

Free University of Bozen-Bolzano  
Personell Office University Staff (call)

19 aprile 2017

## Contents

<b>1</b>	<b>Overview</b>	<b>2</b>
<b>2</b>	<b>List of enclosed documents</b>	<b>2</b>
<b>3</b>	<b>Attachments</b>	<b>4</b>
3.1	Attachment A . . . . .	5
3.2	Attachment B (autocertification) . . . . .	8
3.3	Attachment C (CV) . . . . .	9
3.4	Passport . . . . .	18
3.5	Codice Fiscale . . . . .	19
3.6	PhD Certificate . . . . .	20
3.7	Marie Curie Fellowship Certificate . . . . .	21
<b>4</b>	<b>Publications</b>	<b>22</b>
4.1	Publications . . . . .	23
4.2	Publication . . . . .	65
4.3	Publication . . . . .	147
4.4	Publication . . . . .	164
4.5	Publication . . . . .	335
4.6	Publication . . . . .	346
4.7	Publication . . . . .	356
4.8	Publication . . . . .	367
4.9	Publication . . . . .	378
4.10	Publication . . . . .	401
4.11	Publication . . . . .	409
4.12	Publication . . . . .	419

## 1 Overview

**Application: Selection procedure for one post of University Researcher with Fixed-term Contract**

Faculty .....	Computer Science
Competitive Sector ..	01/B1
S.S.D .....	INF/01
Title of program .....	Intelligent techniques for data and process mangagement
Applicant Surname ..	Rubin
Applicant First Name	Sasha
Applicant Residence .	via Aniello Falcone, 428, Vomero, 80127, Napoli

## 2 List of enclosed documents

- [3.1] Attachment A
- [3.2] Attachment B (autocertification)
- [3.3] Attachment C (CV)
- [3.4] Passport ID
- [3.5] Codice Fiscale
- [3.6] PhD certificate
- [3.7] Marie Curie certificate

- [Pubb. 4.1] Giuseppe De Giacomo, Antonio Di Stasio, Aniello Murano, and Sasha Rubin. “Imperfect information games and generalized planning”. In: *International Joint Conference on Artificial Intelligence (IJCAI 2016)*. 2016.
- [Pubb. 4.2] Francesco Belardinelli, Alessio Lomuscio, Aniello Murano, and Sasha Rubin. “Verification of Multi-agent Systems with Imperfect Information and Public Actions”. In: *Proceedings of the 2017 International Conference on Autonomous Agents & Multiagent Systems, São Paulo, May 8-12, 2017*. 2017.
- [Pubb. 4.3] Raphael Berthon, Bastien Maubert, Aniello Murano, Sasha Rubin, and Moshe Vardi. “Hierarchical Strategic Reasoning”. In: *LICS 2017*. 2017.

- [Pubb. 4.4] Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder. *Decidability of Parameterized Verification*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2015.
- [Pubb. 4.5] Sasha Rubin. "Parameterised Verification of Autonomous Mobile-Agents in Static but Unknown Environments". In: *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015, Istanbul, Turkey, May 4-8, 2015*. 2015, pp. 199–208.
- [Pubb. 4.6] Benjamin Aminof, Vadim Malvone, Aniello Murano, and Sasha Rubin. "Graded Strategy Logic: Reasoning about Uniqueness of Nash Equilibria". In: *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, Singapore, May 9-13, 2016*. 2016, pp. 698–706.
- [Pubb. 4.7] Benjamin Aminof, Aniello Murano, Sasha Rubin, and Florian Zuleger. "Automatic Verification of Multi-Agent Systems in Parameterised Grid-Environments". In: *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, Singapore, May 9-13, 2016*. 2016, pp. 1190–1199.
- [Pubb. 4.8] Benjamin Aminof, Aniello Murano, Sasha Rubin, and Florian Zuleger. "Prompt Alternating-Time Epistemic Logics". In: *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016*. 2016, pp. 258–267.
- [Pubb. 4.9] Benjamin Aminof and Sasha Rubin. "First Cycle Games". In: *Information and Computation*, (2016). doi: <http://dx.doi.org/10.1016/j.ic.2016.10.008>.
- [Pubb. 4.10] Giuseppe De Giacomo, Antonio Di Stasio, Aniello Murano, and Sasha Rubin. "Imperfect information games and generalized planning". In: *International Joint Conference on Artificial Intelligence (IJCAI 2016)*. 2016.
- [Pubb. 4.11] Francesco Belardinelli, Alessio Lomuscio, Aniello Murano, and Sasha Rubin. "Verification of Multi-agent Systems with Imperfect Information and Public Actions". In: *Proceedings of the 2017 International Conference on Autonomous Agents & Multiagent Systems, São Paulo, May 8-12, 2017*. 2017.
- [Pubb. 4.12] Raphael Berthon, Bastien Maubert, Aniello Murano, Sasha Rubin, and Moshe Vardi. "Hierarchical Strategic Reasoning". In: *LICS 2017*. 2017.

19 April 2017



### **3 Attachments**

## **Attachment 'A'**

Free University of Bozen -  
Bolzano  
Personnel Academic Staff  
Universitätsplatz 1  
39100 Bolzano

**Rector's decree dated of 24.02.2017 no. 121**

**Selection procedure for one post of University  
researcher with junior contract Faculty of Computer  
Science**

**Disciplinary-scientific sectors: INF/01 (Computer  
Science)**

**Area of research or title of the research project:  
Intelligent techniques for data and process  
management**

**Supervisor: Dr. Marco Montali**

The undersigned name **Sasha Rubin** requests to be admitted to compete in the above-described selection procedure.

For this purpose, in accordance with the art. 46 e 47 del D.P.R. (Decree of the President of the Republic) of 28th December 2000, n. 445, and aware that falsification of documents and untrue declarations are punished, in accordance with the penal code and with the special laws regarding the matter, according to the provisions referred to in the art. 76 of the D.P.R. of 28<sup>th</sup> December 2000, n. 445, declares under his/her own responsibility that:

- a) he/she was born on **16/02/76** in **Johannesburg, South Africa**;
- b) he/she has the following tax number: **RBNSSH76B16Z347C**
- c) he/she is resident in **Naples** prov./county **Italy** street **Via Aniello Falcone** number **428** post code **80127**;
- d) he/she possesses the following citizenship: **New Zealand**;
- e) he/she is registered on the municipality electoral list of **New**; or, if you are not registered, declare the reasons for the non-r **Zealand** registration or cancellation from such lists; or the foreign nationals must declare to enjoy civil and political rights in the country on which they belong;
- f) not having pending criminal suits;

- g) not having received criminal sentences and not currently undergoing criminal proceedings (otherwise to indicate);
- h) not to exceed the superior limit of twelve years, also non continuous periods, therefore considering the total duration of the contracts under Art. 22 (Research grants) and Art. 24 (Fixed-time researchers) of the law 30th December 2010, no. 240, also between different universities, public, non public or telematic ones, and with bodies as provided in Art. 22, paragraph 1 of the law no. 240/2010, as well as the duration of the contract according to the present call;
- i) to have looked over the provisions cited in the present call and accepts all its provisions;
- j) the content of the digital copies of all documents produced correspond to the content of the copies produced on paper;
- k) not to be a university professor of the first and second rank or a researcher employed with open-ended contract, even if ceased from service;
- l) not having family or kinship relation, up to the fourth degree, with a professor of the Department making the proposal of the activation of the contract, with the Rector, the Director or with a member of the University Council;
- m) not having been dispensed or dismissed from the employment by a Public Administration for persistent insufficient performance, not having been declared lost from other public employment according to art. 127, paragraph 1, letter d) of the T.U. regarding provisions concerning the charter of civil servants, approved with D.P.R. 10th January 1957, no. 3, not having gained the employment through production of faked documents or vitiated by not amendable invalidity, and not having had an employment relationship by a Public Administration which has been terminated for disciplinary reasons, included those provided in art. 21 of the legislative decree 29/1993;
- n) he/she gives the Personnel Office Academic staff the permission to send the scientific curriculum vitae to the mentoring group of the Faculty in order define the optionally payment of the scientific salary increase in case of employment;
- o) ..... (please indicate the kind of activities that are practicing in case the candidate covers other offices or jobs according to art. 12 of the present call)
- p) he/she declares that everything in the curriculum vitae (CV) that has been attached to this application is true and s/he gives the person in charge of the selection procedure the permission to undertake any checks on any substitute declarations that have been presented by the undersigned for the purposes of this selection procedure;
- q) he/she gives his/her permission for the personal details that s/he has supplied to be processed as per Italian legislative decree 196/2003 for any procedures that are connected to this selection procedure and any contracts that may be drawn up as a consequence.
- r) he/she chooses the following address, where communications regarding the present selection procedure can be sent, and undertakes to inform the Administration regarding any subsequent variations:
 

street and number: **428 Via Aniello Falcone** town (post code): **Napoli, 80127**  
 prov./county: **NA**

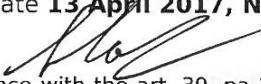
Telephone: **344 281 0361.**

E-mail: **rubin@unina.it**
- s) (For disabled people only): that he/she has the following handicap .....  
 ...., for which reason, in order to sit the

discussion, he/she requires the following assistance ..... as well as the following extra time .....

The undersigned enclose at the application for participation in the selection procedure:

- 1) 1 photocopy of a valid identity document and the fiscal code;
- 2) 1 photocopy of the curriculum vitae regarding the didactic and scientific activity and drawn up according to the model of attachment 'C';
- 3) didactic and scientific qualifications believed useful for the comparative assessment and the corresponding list, if necessary;
- 4) 1 photocopy of the list of the publications believed useful for the comparative assessment and drawn up in accordance with article 5 paragraph 3 of the call;
- 5) 1 photocopy of the list of all documents enclosed at the application, signed and dated at the last page;

Place and date **13 April 2017, Naples**  
Signature 

\* In accordance with the art. 39, pa 1 of the D.P.R. 445/2000 the present declaration requires no authentication of the signature.

## **Attachment 'B'**

**Attention: In case of employment non-EU citizens have to submit certificates as per art. 4 point 7 of the call for qualifications not obtained in Italy or another EU member state**

SUBSTITUTE DECLARATION OF CERTIFICATION  
(Art. 46 D.P.R. 28<sup>th</sup> December 2000, n. 445)

SUBSTITUTE DECLARATION OF ATTESTED AFFIDAVIT REGARDING FACTS, STATUSES, AND PERSONAL QUALITIES DIRECTLY KNOWN TO THE DECLARANT  
(Art. 47 D.P.R. 28th December 2000, n. 445)

The undersigned name **Sasha Rubin**, born **16/02/76** in **Johannesburg, South Africa**, resident in **Naples, Italy**, in street **Via Aniello Falcone, 428**, aware that, in accordance with the artt. 46 e 47 del D.P.R. of 28th December 2000, n. 445, falsification of documents and untrue declarations are punished by the penal code and by the special laws regarding the matter, according to the provisions referred to in the art. 76 del D.P.R. of 28<sup>th</sup> December 2000, n. 445,

DECLARES

under his own responsibility,

that:

- He holds a PhD in Mathematics from the University of Auckland, attained on 06/09/2004
- He holds a MsC in Mathematics from the University of Auckland, attained on 23/04/1999
- He holds a BsC in Mathematics from the University of Cape Town, attained in 1996.
- He held a two-year Marie Curie fellowship with the Istituto Nazionale di Alta Matematica between 03/2015 and 02/2017.
- He held a three-year New Zealand Science and Technology Postdoctoral Fellowship between 2004 and 2007.
- The 12 publications included conform to the originals.
- He was a visiting assistant professor at Cornell University in 2008/2009, where he taught Calculus for Engineers and a graduate course on Random Graphs.
- He held a two year postdoc at TU Wien and IST Austria between 2012-2014
- His 2015 PRIMA paper was awarded "Best Paper".

Town and date **19 April 2017, Naples**  
Signature <sup>(1)</sup>



## **Attachment 'C'**

# **University Academic Curriculum Vitae**

---

### **Personal information**

Name: Sasha Rubin  
Dob: 16/02/76  
Pob: South Africa  
Nationality: New Zealand  
Address: Via Aniello Falcone, 428, Vomero, 80127, Naples  
Telephone number: 344 281 0361  
E-Mail: [rubin@unina.it](mailto:rubin@unina.it)

### **Education since leaving school**

- 1996, Bsc in mathematics,, University of Cape Town
- 1999, MsC in mathematics, University of Auckland
- 2004, PhD in mathematics, "Automatic Structures", University of Auckland

### **Present appointment**

- Fellow of the ASTREA lab
- Start of appointment: 03/2017
- Level: Fellow
- Employer: UNINA "Federico II"
- My present appointment consists of basic research in theoretical computer science and artificial intelligence, helping to supervise undergraduate and PhD students, and teaching short courses.

### **Professional experience**

Chronological list of all previous employments (each with job title, starting and finishing dates, level, employer, responsibilities)

See next page...

From / to	Job title	Name of academic Institution	Academic level	responsibilities
2004/2008	Honorary Research Fellow	U. o. Auckland	PostDoc	Research
2008/2009	Visiting Assistant Professor	Cornell U.	Assistant Professor	Teaching
2009/2010	Visiting Researcher	U. o. Auckland	Visiting Researcher	Research
2010	Visiting Lecturer	U. o. Cape Town	Visiting Lecturer	Teaching
2010/2012	Visiting Researcher	U. o. Auckland	Visiting Researcher	Research
2012/2014	PostDoc	TU Wien, IST Austria	PostDoc	Research
2014/2015	PostDoc	TU Wien, TU Graz	PostDoc	Research
2015/2017	Marie Curie Fellow	UNINA "Federico II"	PostDoc	Research

**Participation in exhibitions (where applicable)**

CONTRIBUTIONS: Primary co-author on all publications.

AWARDS:

- Bsc 1996 Dean's Merit List
- Msc 1998 First Class
- PhD 2004 Prize for best doctoral thesis in the Faculty of Science, and Montgomery memorial prize in logic from the Department of Philosophy

**Experience in academic teaching**

- **Games on graphs**, UNINA "Federico II", Theoretical Computer Science, PhD course.
- According to student evaluations: my teaching is clear, organised, I am proactively willing to help, and motivating (2010). In 2010 I briefly volunteered at a secondary school in Accra, Ghana, teaching, observing and commenting on grade 5mathematics classes. I also briefly volunteered in Khayelitsha, South Africa, helping high-school students prepare for their mathematics exams.

## **Other academic responsibilities**

- **Program Committee member** for IRISA Master Research Internship 2016-2017.
- **Co-chair** of the Italian Conference on Theoretical Computer Science (ICTCS) 2017 , ictcs2017.unina.it, Naples, 26-29 September 2017.
- **Co-chair** of the International Workshop on Strategic reasoning (SR) 2017 (<http://sr2017.csc.liv.ac.uk/>), Liverpool 26-27 July 2017.
- **Co-organiser** of the Italian Conference on Computational Logic (CILC) 2017 (<http://cilc2017.unina.it/>), Naples, 26-29 September 2017.
- **PC member** of the International Joint Conference on Artificial Intelligence (IJCAI) 2017,
- **PC member** of the AAAI Conference on Artificial Intelligence (AAAI) 2017
- **PC member** of the International Workshop of Strategic Reasoning (SR) 2016
- **PC member** of the International Symposium on Games, Automata, Logics and Formal Verification (GandALF) 2016,
- **PC member** of the European Conference on Artificial Intelligence (ECAI) 2016.
- **Co-organiser** of the First Workshop on Formal Methods in AI (FMAI) 2017 (<https://sites.google.com/site/fmai2017homepage/>) 22-24 February 2017, Naples.

## **Memberships**

### **Research and scholarships**

I work in formal methods, a branch of theoretical computer science, and study the power of automata theory (broadly construed) and mathematical logic for describing, reasoning and controlling systems.

During my PhD I (and my co-authors) pioneered the development of the theory of automatic structures. My PhD thesis was awarded the Vice-chancellor's prize for the best doctoral thesis in the Faculty of Science, and Montgomery memorial prize in logic from the Department of Philosophy.

I was then awarded a prestigious New Zealand Science and Technology Postdoctoral Fellowship. During this fellowship, I published a survey and extension of the main results in my thesis in the Bulletin of Symbolic Logic, and I (with a PhD student of Erich Gr\"adel's) solved a 12 year-old conjecture of Courcelle's (STACS'08).

In the last few years, I (with my co-authors) generalised a cornerstone paper on verification of parameterised systems ("Reasoning about Rings", E.A. Emerson, K.S. Namjoshi, {\sc POPL}, 1995) from ring topologies to arbitrary topologies (VMCAI'14). We also completed a book, published by Morgan \&

Claypool in 2015, surveying decidability results in parameterised verification.

Recently, I was awarded a two year Marie-Curie fellowship from the Istituto Nazionale di Alta Matematica to work on formal methods for parameterised light-weight mobile agents. I opened this direction with an article in the premier conference on autonomous and multiagent systems (AAMAS'15). Subsequently (with my co-authors) I continued this direction and published in top rated conferences (AAMAS'16), and won a best-paper award (PRIMA'15) (invited to the premier journal on autonomous and multi-agent systems JAAMAS).

Date granted	Award Holder	Funding Body	Title	Amount received
2004	me	NZ Government	New Zealand Science and Technology Postdoctoral Fellowship, UOAX0413	3 years salary and travel expenses
2015	me	INDAM and the EC	Marie Curie fellow of the National Institute of Higher Mathematics (INdAM ``F. Severi''). INdAM-COFUND-2012, FP7-PEOPLE-2012-COFUND, Proj. ID 600198.	2 years' salary and travel expenses (Eu 43200.00 + RCC2 of Eu2000.00)
15/12/2015	me	GNSAGA	Scientific collaboration with Profs. Lomuscio (Imperial) and Wooldridge (Oxford)	Eu 1050.00
16/03/2016	me	GNSAGA	Attend KR16 and AAMAS16	Eu 1900.00

## Publications

## Publications 2007-2017

1. (\*) Raphael Berthon, Bastien Maubert, Aniello Murano, Sasha Rubin, and Moshe Vardi. "Hierarchical Strategic Reasoning". In: *Logic in Computer Science (LICS 2017)*. 2017.
2. (\*) Verification of Multi-agent Systems with Imperfect Information and Public Actions  
Francesco Belardinelli, Alessio Lomuscio, Aniello Murano,

Sasha Rubin  
*Proceedings of the 2017 International Conference on Autonomous Agents & Multiagent Systems, São Paulo, May 8-12, 2017.*

3. (\*) Imperfect information games and generalized planning  
Giuseppe De Giacomo, and Antonio Di Stasio, Aniello Murano, Sasha Rubin  
*International Joint Conference on Artificial Intelligence (IJCAI 2016),*
4. Decidability in Parameterized Verification  
Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, Josef Widder  
*SIGACT News*, volume 47, number 2, pages 53-64, 2016.
5. (\*) Prompt Alternating-Time Epistemic Logics  
Benjamin Aminof, Aniello Murano, Sasha Rubin, Florian Zuleger  
*Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016.*, pages 258-267, 2016.
6. Model Checking Parameterised Multi-token Systems via the Composition Method  
Benjamin Aminof, Sasha Rubin  
*Automated Reasoning - 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 - July 2, 2016, Proceedings*, pages 499-515, 2016.
7. (\*) Automatic Verification of Multi-Agent Systems in Parameterised Grid-Environments  
Benjamin Aminof, Aniello Murano, Sasha Rubin, Florian Zuleger  
*Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, Singapore, May 9-13, 2016*, pages 1190-1199, 2016.
8. (\*) Graded Strategy Logic: Reasoning about Uniqueness of Nash Equilibria  
Benjamin Aminof, Vadim Malvone, Aniello Murano, Sasha Rubin  
*Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, Singapore, May 9-13, 2016*, pages 698-706, 2016.
9. (\*) First Cycle Games  
Benjamin Aminof, Sasha Rubin  
*Information and Computation*, 2016.
10. Graded Strategy Logic  
Benjamin Aminof, Vadim Malvone, Aniello Murano, Sasha Rubin  
*Proceedings 4th International Workshop on Strategic*

*Reasoning, SR 2016, New York, USA., 2016.*

11. (\*) Decidability of Parameterized Verification  
Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov,  
Sasha Rubin, Helmut Veith, Josef Widder  
2015, Morgan & Claypool Publishers.
12. (\*) Verification of Asynchronous Mobile-Robots in Partially-Known Environments  
Benjamin Aminof, Aniello Murano, Sasha Rubin, Florian Zuleger  
*PRIMA 2015: Principles and Practice of Multi-Agent Systems - 18th International Conference, Bertinoro, Italy, October 26-30, 2015, Proceedings, pages 185-200, 2015*  
[Best-Paper Award]
13. Multi-agent Path Planning in Known Dynamic Environments  
Aniello Murano, Giuseppe Perelli, Sasha Rubin  
*PRIMA 2015: Principles and Practice of Multi-Agent Systems - 18th International Conference, Bertinoro, Italy, October 26-30, 2015, Proceedings, pages 218-231, 2015.*
14. On the Expressive Power of Communication Primitives in Parameterised Systems  
Benjamin Aminof, Sasha Rubin, Florian Zuleger  
*Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24-28, 2015, Proceedings, pages 313-328, 2015.*
15. On CTL\* with Graded Path Modalities  
Benjamin Aminof, Aniello Murano, Sasha Rubin  
*Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24-28, 2015, Proceedings, pages 281-296, 2015.*
16. (\*) Liveness of Parameterized Timed Networks  
Benjamin Aminof, Sasha Rubin, Francesco Spegni, Florian Zuleger  
*Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II, pages 375-387, 2015.*
17. (\*) Parameterised Verification of Autonomous Mobile-Agents in Static but Unknown Environments  
Sasha Rubin  
*Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015, Istanbul, Turkey, May 4-8, 2015, pages 199-208, 2015.*
18. Alternating traps in Muller and parity games  
Andrey Grinshpun, Pakawat Phalitnonkiat, Sasha Rubin,  
Andrei Tarfulea

*Theor. Comput. Sci.*, volume 521, pages 73-91, 2014.

19. First Cycle Games  
Benjamin Aminof, Sasha Rubin  
*Proceedings 2nd International Workshop on Strategic Reasoning, SR 2014, Grenoble, France, April 5-6, 2014.*, pages 83-90, 2014.
20. (\*) Parameterized Model Checking of Token-Passing Systems  
Benjamin Aminof, Swen Jacobs, Ayrat Khalimov, Sasha Rubin  
*Verification, Model Checking, and Abstract Interpretation - 15th International Conference, VMCAI 2014, San Diego, CA, USA, January 19-21, 2014, Proceedings*, pages 262-281, 2014.
21. (\*) Parameterized Model Checking of Rendezvous Systems  
Benjamin Aminof, Tomer Kotek, Sasha Rubin, Francesco Spegni, Helmut Veith  
*CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings*, pages 109-124, 2014.  
2013
22. How to Travel between Languages  
Krishnendu Chatterjee, Siddhesh Chaubal, Sasha Rubin  
*Language and Automata Theory and Applications - 7th International Conference, LATA 2013, Bilbao, Spain, April 2-5, 2013. Proceedings*, pages 214-225, 2013.
23. A Myhill-Nerode theorem for automata with advice  
Alex Kruckman, Sasha Rubin, John Sheridan, Ben Zax  
*Proceedings Third International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2012, Napoli, Italy, September 6-8, 2012.*, pages 238-246, 2012.
24. (\*) Interpretations in Trees with Countably Many Branches  
Alexander Rabinovich, Sasha Rubin  
*LICS 2012, Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, Dubrovnik, Croatia, June 25-28, 2012.*, pages 551-560, 2012.
25. (\*) Automata-based presentations of infinite structures  
Vince Bárány, Erich Grädel, Sasha Rubin  
Chapter in *Finite and Algorithmic Model Theory* (Javier Esparza, Christian Michaux, Charles Steinhorn, eds.), pages 1-76, 2011, Cambridge University Press.  
Note: Cambridge Books Online
26. (\*) Automata Presenting Structures: A Survey of the Finite String Case  
Sasha Rubin  
*Bulletin of Symbolic Logic*, volume 14, number 2, pages 169-209, 2008.

27. (\*) Cardinality and counting quantifiers on omega-automatic structures  
 Lukasz Kaiser, Sasha Rubin, Vince Bárány  
*STACS 2008, 25th Annual Symposium on Theoretical Aspects of Computer Science, Bordeaux, France, February 21-23, 2008, Proceedings*, pages 385-396, 2008.
28. (\*) Order-Invariant MSO is Stronger than Counting MSO in the Finite  
 Tobias Ganzow, Sasha Rubin  
*STACS 2008, 25th Annual Symposium on Theoretical Aspects of Computer Science, Bordeaux, France, February 21-23, 2008, Proceedings*, pages 313-324, 2008.
29. (\*) Automatic Structures: Richness and Limitations  
 Bakhadyr Khoussainov, André Nies, Sasha Rubin, Frank Stephan  
*Logical Methods in Computer Science*, volume 3, number 2, 2007.

### **Publications about the applicant**

#### **Further data**

##### **INVITED:**

- Verification of Multi-Agent Systems with Imperfect Information and Public Actions, FMAI 2017, Naples, Italy
- Removing Partial Observability from Generalized Planning Problems, July 2017, Bozen, Italy (Hosts: Montali, Calvanese)

##### **SELECTED:**

- IJCAI 16: Imperfect information games and generalized planning
- AAMAS 16: Automatic Verification of Multi-Agent Systems in Parameterised Grid-Environments
- KR 16: Prompt Alternating-Time Epistemic Logics
- PRIMA 15: Verification of Asynchronous Mobile-Robots in Partially-Known Environments
- PRIMA 15: Multi-agent Path Planning in Known Dynamic Environments
- AAMAS 15: Parameterised Verification of Autonomous Mobile-Agents in Static but Unknown Environments
- SR 14: First Cycle Games
- VMCAI 14: Parameterized Model Checking of Token-Passing Systems

#### **Statement of interest**

Intelligent techniques for data and process management include identifying expressive yet tractable **languages** in which to express control, data, **interaction**, and **limited visibility** of processes, as well as techniques for **verification**, **monitoring** and **mining** of these processes. My background is suited to

making foundational contributions to this area for four reasons.

First: I have deep and extensive knowledge in automata theory which I can draw upon to formalise and study appropriate **languages** for process management (see, my work on automatic structures between 2002 and 2007).

Second: I have worked in verification of infinite state systems (BPM systems are typically infinite-state because there is no a-priori bound on the variable domains); see my work since 2014 on parameterised model-checking ([4,6,7,11,12, 17,20,21] above).

Third: I have recently worked on strategic reasoning in artificial intelligence [1,2,3,5,8,9,10]; this will allow me to contribute to a deep study of the **interaction** of processes. Concretely, processes may be the result of multiple “players” that may have conflicting goals (e.g., multiple companies want the dominant share of a market); to make sense of such processes one would need to use strategic notions from game-theory (such as equilibria), which, in turn, need to be formalised so that they can be automatically reasoned about for verification, monitoring and mining.

Fourth: I have recently acquired expertise in modeling and reasoning about imperfect and incomplete information [1,2,3,5]. This will allow me to contribute to processes with **limited visibility**.

**Language competence**

English: C2  
Italian: A2-B1  
German: A1  
Portuguese: A1

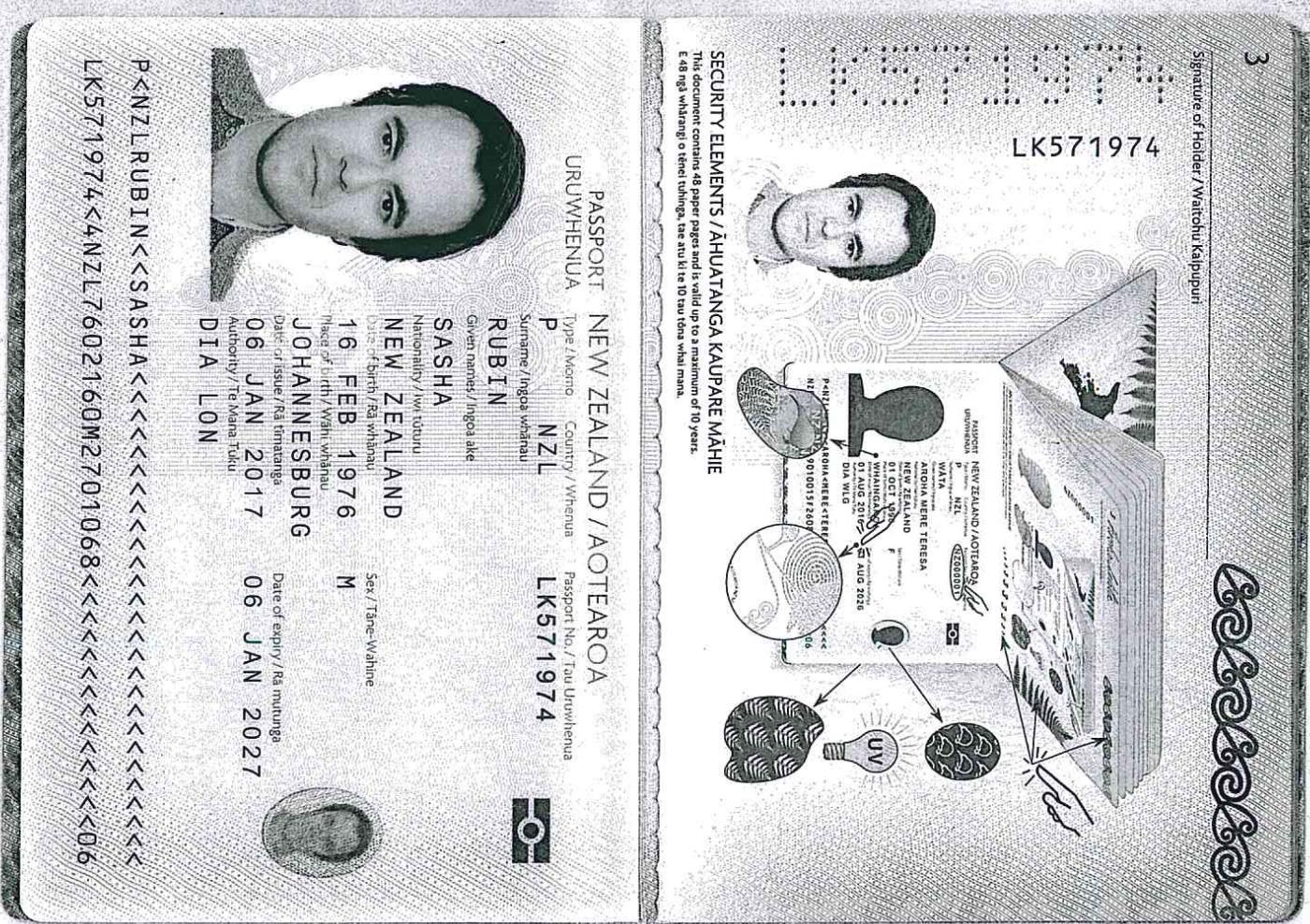
(these are approximate grades; no certificates were obtained)

Date

13 April 2017

Signature

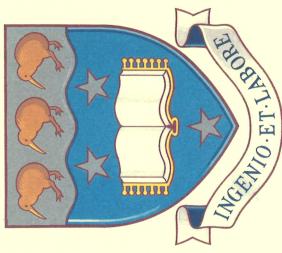






A handwritten signature is written over the date of birth on the card.

13 aprile 2017



# THE UNIVERSITY OF AUCKLAND

THIS IS TO CERTIFY THAT

*Sasha Rubin*

has been admitted to the degree of

## DOCTOR OF PHILOSOPHY

in Mathematics  
and Computer Science

Given under the Seal of the University of Auckland  
this Seventh day of May 2007

*Hugh Fletcher*  
Hugh Fletcher  
CHANCELLOR

*Jeffrey S. Guralnick*  
Jeffrey S. Guralnick  
VICE-CHANCELLOR  
REGISTRAR

# ISTITUTO NAZIONALE DI ALTA MATEMATICA FRANCESCO SEVERI

CITTÀ UNIVERSITARIA - 00185 ROMA

<http://www.altamatematica.it> e-mail: [indam@altamatematica.it](mailto:indam@altamatematica.it)  
Tel. 06.490320 – 06.4440665 – Fax 06.4462293

## TO WHOM IT MAY CONCERN

Dr. Sasha Rubin was awarded the post-doc INdAM-COFUND fellowships in Mathematics and/or Applications for experienced researchers cofunded by Marie Curie actions (Incoming type) for the period of 2 years.

Dr. Sasha Rubin started his work at the University of Naples (Università degli Studi di Napoli Federico II), Italy from Mar. 1, 2015 to Feb. 28, 2017.

Rome, Apr. 18, 2017

Dott. Giovanni Feliciangeli

Uff. Ragioneria INdAM



## **4 Publications**

The rest of this page is intentionally left blank

## **4.1 Publications**

The rest of this page is intentionally left blank

AUTOMATA PRESENTING STRUCTURES:  
A SURVEY OF THE FINITE STRING CASE

SASHA RUBIN

**Abstract.** A structure has a (finite-string) *automatic presentation* if the elements of its domain can be named by finite strings in such a way that the coded domain and the coded atomic operations are recognised by synchronous multitape automata. Consequently, every structure with an automatic presentation has a decidable first-order theory. The problems surveyed here include the classification of classes of structures with automatic presentations, the complexity of the isomorphism problem, and the relationship between definability and recognisability.

CONTENTS

1. Introduction	170
1.1. Summary	172
1.2. Scope and related work	173
1.3. Acknowledgement	174
2. Definitions	174
2.1. Notation	174
2.2. Synchronous finite automata	175
2.3. Automatic presentations	176
2.4. Examples	178
3. Properties	178
3.1. Fundamental properties	178
3.2. Extending first-order	181
3.3. Automatic presentations	188
3.4. Intrinsic regularity	190
3.5. Restriction on growth	191
3.6. Isomorphism problem	195
4. Classifications	197
4.1. Equivalence structures	198
4.2. Linear orders	199

---

Received February 2, 2004.

© 2008, Association for Symbolic Logic  
1079-8986/08/1402-0001/\$5.10

4.3. Trees	202
4.4. Boolean algebras	203
4.5. Groups, rings and fields	205
5. Open problems	206

**§1. Introduction.** Which infinite structures can be stored and manipulated by a computer? At the very least, the structure should be representable in a finite amount of space. Moreover, operations that one would like to perform on the structure, typically logical queries, should be computable.

Finite automata are a robust model of computation that seem well suited to describing infinite structures. Although automata classically recognise sets of strings, they can be generalised to recognise  $n$ -ary relations by introducing  $n$  input-tapes. A (*finite-string*) *automatic presentation* of a structure is a coding of its domain as a set of finite strings so that the domain and each of the atomic operations is recognised by an automaton operating synchronously on its inputs (Definition 2.5)

Automatic presentations are a refinement of computable presentations, and have two salient properties: 1) given an automatic presentation of a structure  $\mathcal{A}$ , every first-order definable relation in  $\mathcal{A}$ , allowing parameters, is computable by a finite automaton (Theorem 3.1); 2) there are automatically presentable structures, such as certain expansions of the standard model of Presburger arithmetic, that interpret every automatically presentable structure (Theorem 3.8).

The effectiveness of using automata to present infinite structures has been amply displayed in the related concept of automatic groups. These are finitely-generated groups whose Cayley graphs (over a certain natural encoding) are recognised by automata. They were introduced by Thurston in 1986 who was motivated by work of Cannon [18] on hyperbolic groups. Automatic groups form a rich collection of finitely presented groups with tractable algorithmic properties that are undecidable in the general case. For instance, in an automatic group the word problem is solvable in quadratic time and the group elements can be efficiently enumerated. Moreover, every automatic group is also finitely presentable (in the group theoretic sense), and a presentation can be extracted from the automata. Software to find and work with automatic groups has been developed, and packaged with the computational algebra tools GAP and MAGMA; see Holt [33]. The standard reference for automatic groups is the book by Cannon, Epstein, Holt, Levy, Paterson, and Thurston [19]. Automatic groups are not covered in this survey.

The connections between automata and logic that are relevant to structures with automatic presentations can be traced to Büchi [16], Elgot [27] and Trahtenbrot [56]. They characterised, in logical terms, the relations over

finite strings recognisable by automata (Theorem 3.8 (iv)). In particular they established the decidability of WS1S: the weak monadic second-order theory of the structure  $\mathbb{N}$  with the successor function. This technique—relating definability and automata to solve decision problems for certain theories—was generalised to automata working on infinite strings (known as  $\omega$ -automata) for the monadic second-order theory of one successor S1S [17], to automata on finite trees for WS2S [54, 25], and to automata on infinite trees for S2S [49]. However, structures presentable by these more general automata have received less study, and so this survey focuses solely on the finite string case.

The importance of these particular theories is that many first-order theories can be interpreted in them. For instance, Büchi proved that the first-order theory of  $(\mathbb{N}, +)$  is decidable by showing that it is interpretable in WS1S. Taking this lead, Hodgson [30, 31, 32] called the first-order-theory of a structure ‘automaton decidable’, if after coding elements of the structure as finite or infinite strings, one can effectively construct automata recognising the encodings of its first-order definable relations. He was interested in the connections between two different ways of proving decidability: via automata and via algebraic constructions (such as direct product and direct power). He observed that certain products of ( $\omega$ -)automatically presentable structures are again ( $\omega$ -)automatically presentable, and thus decidable.<sup>1</sup>

Khoussainov and Nerode [34], independently of Hodgson and inspired by the success of using automata to describe groups as in [19], introduced structures presentable by automata as part of complexity-theoretic model theory (also called feasible model theory). Here is a brief description of this area.

The general idea is to fix a complexity class  $\mathcal{C}$  (such as polynomial time, exponential time, etc.) and study algorithmic properties of  $\mathcal{C}$ -structures: those that can be represented by machines operating with complexity in  $\mathcal{C}$ . A typical question is whether a given infinite structure is isomorphic to a  $\mathcal{C}$ -structure. For instance, in the 1980’s Nerode, Remmel, and Cenzer led the development of polynomial-time structures; see the survey [20]. They prove, for instance, that every computable structure is computably isomorphic to a polynomial-time structure over a binary alphabet.

Structures presentable by automata are part of complexity-theoretic model theory—take the complexity class corresponding to synchronous multitape automata.

Khoussainov and Nerode suggested in [34] that the algebraic-, model theoretic-, and complexity theoretic-properties of automatically presentable structures are amenable to systematic investigation. For instance, they characterised the class of automatically presentable structures via a generalisation of the Myhill-Nerode Theorem for regular languages [34, Theorem 3.3].

---

<sup>1</sup>An  $\omega$ -automatic presentation, also called Büchi-automatic presentation, is similar to an automatic presentation, except that infinite strings rather than finite strings are used to code the domain of the structure.

Blumensath and Grädel [9, 12] introduced automatic presentations of structures to the logic in computer science community. They established many fundamental results, such as the characterisation of automatically presentable structures via interpretability (Theorem 3.8). They discussed the complexity of various problems associated with evaluating formulas, such as model checking various fragments of first-order logic on automatically presentable structures.<sup>2</sup> They also provided the fundamental results for the classes of structures presentable by  $\omega$ -automata and  $(\omega)$ -tree automata.

My co-authors and I have focused on classifying classes of automatically presented structures in terms of classical invariants. For instance, what can be said about the Cantor-normal form of automatically presentable ordinals? (This question was asked by Khoussainov and Nerode [34] and solved by Delhommé [24]). Or what can be said about the Cantor–Bendixson rank of automatically presentable trees or of automatically presentable Boolean algebras? We aim for positive results of the form ‘A structure from a certain class is automatically presentable if and only if it has certain algebraic properties’, and negative results of the form ‘The isomorphism problem for a certain class of automatically presented structures is complete for a certain level of the arithmetic/analytic hierarchy’. These are discussed in detail in Section 4. Since the condition of being automatically presentable is quite strong, it is not surprising that some classes of structures (ordinals and Boolean algebras for instance) are simple, in the sense that they are easy to describe. Surprisingly, some classes turn out to be complex, in the sense that it is hard to detect if two members are isomorphic. Notably, the isomorphism problem for the class of all automatically presented structures, easily seen to be undecidable, is  $\Sigma_1^1$ -complete (Theorem 3.53). Finally, there are many classes (groups, rings, and linear orders for instance) for which it is not yet known whether their automatically presentable members are simple or complex (in the senses described), or somewhere in-between.

A closely related problem is that of providing techniques for showing that a given structure does not have an automatic presentation. For instance, Delhommé [24] provides a necessary condition for a structure to have an automatic presentation, which implies, in particular, that the ordinal  $\omega^\omega$  does not (Corollary 3.52).

The following summary of the rest of this survey should give an indication of the main lines of research in the area.

**1.1. Summary.** Section 2 (Definitions) includes the definitions of regular relations and automatic presentations. The section ends with some common examples.

Section 3 (Properties) first covers general properties of automatically presentable structures, and later some specific topics.

---

<sup>2</sup>The model checking problem is, given a presentation of  $\mathcal{A}$ , a formula  $\phi(\bar{x})$ , and a tuple of parameters  $\bar{a}$  in  $\mathcal{A}$ , to decide whether or not  $\mathcal{A} \models \phi(\bar{a})$ .

Subsection 3.1 ‘Fundamental properties’ includes the decidability result (Theorem 3.2), closure under natural operations (Corollary 3.7), and the logical characterisation of automatically presentable structures (Theorem 3.8).

Subsection 3.2 ‘Extending first-order’ refers to extending the basic decidability result by certain additional quantifiers including ‘there exists infinitely many’ and ‘there exist  $k$  modulo  $m$  many’. This is done in the context of generalised quantifiers and order-invariance.

Subsection 3.3 ‘Automatic presentations’ offers a definition of when two presentations are equivalent, and gives a machine theoretic characterisation.

Subsection 3.4 ‘Intrinsic regularity’ presents the analogue of intrinsically computably enumerable relations (see [1]) for automatically presentable structures.

Subsection 3.5 ‘Restriction on growth’ presents some general ways of proving that a given structure has no automatic presentation. As illustration, the following structures are not automatically presentable: the free semigroup on  $k > 1$  generators, Skolem arithmetic  $(\mathbb{N}, \times)$ , the random graph, and the ordinal  $\omega^\omega$ .

Subsection 3.6 ‘Isomorphism problem’ includes the proof that the isomorphism problem for automatically presented structures is  $\Sigma_1^1$ -complete.

Section 4 (Classifications) is concerned with classifying the automatically presentable members of a given class of structures in relevant algebraic terms. There are many classes for which a complete classification is known for the restricted notion of automaticity requiring that only strings over a unary alphabet are used. In the general (non-unary) case, a complete classification is known for the classes of ordinals, Boolean algebras and fields. However, only partial classifications are known for the following classes: equivalence structures, linear orders, groups, and rings.

Section 5 (Open problems) contains a sample of problems whose solutions will likely require new ideas.

**1.2. Scope and related work.** Familiarity is assumed with the basics of finite automata, formal languages, and logic. Proofs will generally be sketched. The reader is referred to the literature for details: the original article [34]; the article [12] for an excellent reference of the fundamental results; and the theses [9], [51] and [3] which may serve as introductions to the area.

Automatically presentable structures can be generalised in several directions: for instance, by using finite automata on infinite strings [9, 12, 39], finite ranked trees [9, 5], finite unranked trees [40], or WMSO-interpretations (of dimension one) of trees with decidable WMSO-theory [21] (see Definition 3.4). The general theory goes through: decidability of the first-order theory and characterisations via interpretability in some structure. Problems such as classification and techniques for proving non-automaticity in these

more general settings are not dealt with in this survey. Consult [24, 21] for techniques showing that a structure has no finite-tree automatic presentation; and [39, 4] for the addition of generalised quantifiers ‘there exists uncountably many’, ‘there exists countably many’ and ‘there exists  $k$  modulo  $m$  many’ in  $\omega$ -automatically presentable structures; and [21] for intrinsic regularity and proving non-automaticity in WMSO-interpreted structures.

Automatic groups [19] are not in the scope of this survey. See [14] for some remarks relating them to automatically presentable structures.

For the complexity of model-checking and related problems, consult [9, 12] for fragments of FO, and [43] for structures of bounded degree.

Definability issues (VC-dimension, quantifier elimination) in universal string- and tree-automatically presentable structures and their reducts can be found in [40, 5, 6].

**1.3. Acknowledgement.** I thank Valentin Goranko for discussions on intrinsic regularity, Lauri Hella for discussions on generalised quantifiers, André Nies for some corrections, Wolfgang Thomas for clarifying Trahtenbrot’s historical contribution, and Vince Bárány, Michael Benedikt and Alexander Raichev for their comments.

## §2. Definitions.

**2.1. Notation.** Countable means finite or countably infinite. A *relational signature*  $\tau$  is a countable sequence of symbols  $(R_i)_i$  and corresponding arities  $r_i$ . A  $\tau$ -structure  $\mathcal{A} = (A; (R_i^{\mathcal{A}})_i)$  consists of a countable set  $A$ , called the *domain* of  $\mathcal{A}$ , written  $\text{dom}(\mathcal{A})$ , and for each  $i$ , a relation  $R_i^{\mathcal{A}} \subseteq A^{r_i}$ , called an *atomic relation* of  $\mathcal{A}$ . When there is only one structure around, I may drop the superscript with the name of the structure.

Structures are written in script  $\mathcal{A}, \mathcal{B}, \dots$  and their corresponding domains are written in capitals  $A, B, \dots$ . The substructure of relational  $\mathcal{A}$  on set  $B \subseteq A$  is written  $\mathcal{A} \upharpoonright B$  or  $\mathcal{B}$  if there can be no confusion.

Signatures are assumed to be computable (the mappings  $i \mapsto R_i$  and  $i \mapsto r_i$  are computable), and to contain the equality symbol  $=$ , though this symbol may not be explicitly mentioned in the signature. For convenience, I sometimes write a structure containing functions, such as  $(\mathbb{N}, +)$ , but am implicitly referring to its *relational variant* obtained by replacing every function  $f : B^k \rightarrow B$  by its graph  $\{(\bar{x}, y) \in B^{k+1} \mid f(\bar{x}) = y\}$ .

If unspecified, all formulas  $\phi(\bar{x})$  (and associated notions like definability) are first-order and allow parameters. However, we will see extensions  $\mathcal{L}$  of first-order logic, particularly monadic second-order logic and extensions by generalised quantifiers.

An  $\mathcal{A}$ -formula is a formula over the signature of structure  $\mathcal{A}$ . The *relation in  $\mathcal{A}$  defined by  $\Phi(\bar{x}, \bar{y})$  with parameters  $\bar{b}$* , denoted by  $\Phi^{\mathcal{A}}(\cdot, \bar{b})$ , is defined as

$$\{(a_1, \dots, a_m) \mid \mathcal{A} \models \Phi(\bar{a}, \bar{b})\},$$

where  $\mathcal{A}$  is a  $\tau$ -structure,  $\Phi(\bar{x}, \bar{y})$  is an  $\mathcal{A}$ -formula with free variables  $\bar{x} = (x_1, \dots, x_m)$ , and  $\bar{b}$  a tuple from  $\mathcal{A}$ . To ease readability, I often relax the notation and write  $\Phi^{\mathcal{A}}(\bar{b})$  or even  $\Phi^{\mathcal{A}}$ .

Familiarity is assumed with the basics from automata theory. To fix notation: symbols will usually be denoted  $a, b, \dots$ ; a finite alphabet of symbols by  $\Sigma$ ; finite strings by  $u, v, w, \dots$ ; the set of finite strings over  $\Sigma$  by  $\Sigma^*$ ; the set of infinite strings over  $\Sigma$  by  $\Sigma^\omega$ ; the empty string by  $\lambda$ ; concatenation by  $\cdot$  as in  $w \cdot v$ , or simply by juxtaposition, as in  $wv$ ; the concatenation of a symbol  $a$  with itself  $n$  times by  $a^n$  ( $a^0$  is defined as  $\lambda$ ); the length of a string  $w$  by  $|w|$ ; the strings in a set  $A \subseteq \Sigma^*$  of length exactly  $n$  by  $A^{=n}$ , and those of length at most  $n$  by  $A^{\leq n}$ . These should not be confused with the set  $A^n$  of  $n$ -tuples from a set or domain  $A$ . A deterministic finite automaton  $M$  over alphabet  $\Sigma$  is of the form  $(Q, \iota, \Delta, F)$  where  $Q$  is a finite set of states,  $\iota \in Q$  is the initial state,  $\Delta: Q \times \Sigma \rightarrow Q$  is the transition function, and  $F \subseteq Q$  is the set of accepting states.

The logarithmic and exponential functions are taken with base 2. In particular, the functions  $\exp_k: \mathbb{N} \rightarrow \mathbb{N}$  are defined for  $k \in \mathbb{N}$  recursively by  $\exp_0(n) = n$  and  $\exp_{k+1}(n) = 2^{\exp_k(n)}$ . The subscript in  $\exp_1$  will be dropped.

**2.2. Synchronous finite automata.** Consider a finite automaton as a restricted non-deterministic Turing machine: it has a read-only input-tape with one head that only moves in one direction. It has no work tape. By admitting more than one input-tape, say  $n$  many, each with its own head moving independently, the language computed by such a machine is an  $n$ -ary relation. The resulting relations are called *rational relations*. These multi-tape automata do not share the robustness of one-tape automata: they are not closed under the Boolean operations, and the nondeterministic rational relations strictly contain the deterministic rational relations.

This article deals with particular rational relations that do enjoy strong closure properties (Theorem 2.4), namely those recognisable by *synchronous  $n$ -tape automata* (also called letter-to-letter automata). The following informal description follows Eilenberg, Elgot, and Shepherdson [26]. A synchronous  $n$ -tape automaton can be thought of as a one-way Turing machine with  $n$  input-tapes. Each tape is regarded as semi-infinite having written on it a string over the alphabet  $\Sigma$  followed by an infinite succession of blanks,  $\perp$  symbols. The automaton starts in the initial state, reads simultaneously the first symbol of each tape, changes state, reads simultaneously the second symbol of each tape, changes state, etc., until it reads a blank on each tape. The automaton then stops and accepts the  $n$ -tuple of strings if it is in an accepting state. The set of all  $n$ -tuples accepted by the automaton is the relation recognised by the automaton. Since  $n$ -tape automata are simply 1-tape automata over a new alphabet, they can be determinised by the usual subset construction.

Instead of formalising this model of computation, we encode a tuple of strings from  $\Sigma^*$  as a single string over an expanded alphabet. For example, for  $\Sigma = \{1\}$  the expanded alphabet is  $\{\left(\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right), \left(\begin{smallmatrix} 1 \\ \perp \end{smallmatrix}\right), \left(\begin{smallmatrix} \perp \\ 1 \end{smallmatrix}\right), \left(\begin{smallmatrix} \perp \\ \perp \end{smallmatrix}\right)\}$  and the string associated with the tuple  $(1^m, 1^{m+1})$  is  $\left(\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right)^m \left(\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right)$ .

**DEFINITION 2.1.** Write  $\Sigma_\perp$  for  $\Sigma \cup \{\perp\}$ , where  $\perp$  is a symbol not in  $\Sigma$ . The *convolution of a tuple*  $(w_1, \dots, w_n) \in (\Sigma^*)^n$  is the string  $\otimes(w_1, \dots, w_n)$  over alphabet  $(\Sigma_\perp)^n$ , of length  $\max_i |w_i|$ , defined as follows. Its  $k$ th symbol is  $(a_1, \dots, a_n)$  where  $a_i$  is the  $k$ th symbol of  $w_i$  if  $k \leq |w_i|$  and  $\perp$  otherwise. In particular,  $\otimes(\lambda, \dots, \lambda) = \lambda$ .

The *convolution of a relation*  $R \subseteq (\Sigma^*)^n$  is the set  $\otimes R = \{\otimes \bar{w} \mid \bar{w} \in R\}$ ; that is,  $\otimes R \subseteq (\Sigma_\perp)^{n*}$  is the set of convolutions of all the tuples in  $R$ .

**DEFINITION 2.2.** A relation  $R \subseteq (\Sigma^*)^n$  is called *synchronous rational*, or simply *regular*, if there is a finite automaton over alphabet  $(\Sigma_\perp)^n$  recognising the convolution  $\otimes R$ .

In case  $n = 1$ , the convolution  $\otimes R$  equals  $R$ , and so in this case the definition coincides with the traditional class of regular languages.

**EXAMPLES 2.3.** The following relations on  $\Sigma^*$  are regular.

- (i) The prefix relation  $\preceq_p$ .
- (ii) The equal length relation  $\text{el}(w, v)$  if  $|w| = |v|$ .
- (iii) The longest common prefix relation, written functionally  $x \sqcap y = z$ .
- (iv) The prefix-lexicographic ordering (induced by a fixed ordering  $<$  on  $\Sigma$ ) defined by  $x \leq_{\text{lex}} y$  if  $x \preceq_p y$  or otherwise  $za \preceq_p x$  and  $zb \preceq_p y$  and  $a < b$  where  $z = x \sqcap y$ .
- (v) The length-lexicographic ordering  $\leq_{\text{llex}}$  defined by  $x \leq_{\text{llex}} y$  if  $|x| < |y|$  or otherwise  $|x| = |y|$  and  $x \leq_{\text{lex}} y$ .

The synchronous coding ensures that the regular relations have basic closure properties.

**THEOREM 2.4.** Let  $R, S \subseteq \Sigma^{*n}$  be regular relations. Then the following relations are also regular:

- (i) union  $R \cup S$ , intersection  $R \cap S$ , relative complementation  $R \setminus S$ ;
- (ii) projection  $\{\bar{y} \mid (\exists x)R(x, \bar{y})\}$ ;
- (iii) instantiation  $\{\bar{y} \mid R(w, \bar{y})\}$  for fixed  $w \in \Sigma^*$ ;
- (iv) cylindrification  $\{(x, \bar{y}) \mid x \in \Sigma^* \wedge R(\bar{y})\}$ ; and
- (v) permutation of the co-ordinates of  $R$ .

Moreover, there is an effective procedure that given automata for  $\otimes R$  and  $\otimes S$ , constructs an automaton for the convolution of each of the resulting relations.

**2.3. Automatic presentations.** We are ready to define what it means for a structure to be described by a collection of automata. Although the definition is only for relational structures, recall the convention that we implicitly replace functions by their graphs.

**DEFINITION 2.5.** A (*finite-string*) *automatic presentation* of a relational structure  $\mathcal{B} = (B; (R_i^{\mathcal{B}})_i)$  consists of a mapping  $\mu$  and a tuple of automata  $\overline{M} = (M_A, M_=, (M_i)_i)$  so that

- (i)  $M_A$  recognises a set  $A \subseteq \Sigma^*$ ,
- (ii) the mapping  $\mu: A \rightarrow \text{dom}(\mathcal{B})$  is surjective, and
- (iii) for every atomic relation  $R_i^{\mathcal{B}}$  of  $\mathcal{B}$  (say of arity  $r_i$ ), the relation

$$\mu^{-1}(R_i^{\mathcal{B}}) := \{(w_1, \dots, w_{r_i}) \in A^{r_i} \mid R_i^{\mathcal{B}}(\mu(w_1), \dots, \mu(w_{r_i}))\},$$

is regular; and its convolution is recognised by the automaton  $M_i$ .

Since by our convention there is a symbol for equality, the relation

$$\{(w_1, w_2) \in A^2 \mid \mu(w_1) =^{\mathcal{B}} \mu(w_2)\}$$

need also be regular; and its convolution is recognised by the automaton  $M_=$ .

Say that  $\mathcal{B}$  is *automatically presentable*. Depending on the focus, the *automatic presentation* may simply refer to the mapping  $\mu$  or to the tuple of automata  $\overline{M}$ .

The induced quotient structure

$$(A; (\mu^{-1}(R_i^{\mathcal{B}}))_i) / \mu^{-1}(=^{\mathcal{B}})$$

is an *automatic copy* of  $\mathcal{B}$ .

If the signature of  $\mathcal{B}$  is infinite, we also require that the function mapping  $i \in \mathbb{N}$  to (a code for) the automaton  $M_i$  be computable.

If  $|\Sigma| = 1$ , then we speak of a *unary-automatic presentation*.

The idea is that strings from  $A$  code elements of  $B$  (via the mapping  $\mu$ ) so that the induced relations  $\mu^{-1}(R_i^{\mathcal{B}})$  are regular. In general, an element of  $B$  may have more than one code. However, if  $\mu$  is also an injection, then every element of  $B$  is coded by a unique string of  $A$ . In this case,  $\mu$  is called an *injective automatic presentation* of  $\mathcal{B}$ .

**PROPOSITION 2.6.** *Every automatically presentable structure  $\mathcal{B}$  has an injective automatic presentation  $v$ . Moreover,  $v$  can be chosen over a binary alphabet (that is, with  $|\Sigma| = 2$ ).*

**PROOF.** Let  $\mu: A \rightarrow B$  be an automatic presentation of  $\mathcal{B}$  over alphabet  $\Gamma$ . For the first item, it is sufficient to restrict  $A$  to a regular set  $A'$  so that  $\mu: A' \rightarrow B$  is a bijection. To this end, define  $A'$  as the set of strings  $v \in A$  such that  $v$  is the length-lexicographically least string in the set  $\{w \in A \mid \mu(w) = \mu(v)\}$ . It is straightforward to check that  $A'$  is regular.

For the second item, suppose  $k = |\Gamma| > 2$  and let  $\Sigma$  be a binary alphabet. Fix some integer  $l \geq \log_2 k$ . The idea is to code each symbol  $\gamma \in \Gamma$  by a distinct string  $\alpha(\gamma) \in \Sigma^*$  of length  $l$ . Then extend  $\alpha$  to  $\Gamma^*$  in the natural way (define  $\alpha(uv) = \alpha(u)\alpha(v)$ ). The required presentation is  $v: \alpha(A') \rightarrow B$  defined by  $v(\alpha(v)) = \mu(v)$  for  $v \in A'$ .  $\dashv$

#### 2.4. Examples.

- (i) Every finite structure is automatically presentable.
- (ii) For  $\Sigma = \{0, \dots, k-1\}$  ( $k \in \mathbb{N}$ ), define the structure  $\mathcal{W}_k$  as

$$(\Sigma^*; (\sigma_a)_{a \in \Sigma}, \preceq_p, \text{el}).$$

Here  $\sigma_a(w) = wa$ ,  $\preceq_p$  is the prefix relation, and  $\text{el}(w, v)$  if  $|w| = |v|$ . It is automatically presentable.

- (iii) The structure

$$\mathcal{N}_k = (\mathbb{N}; +, |_k)$$

is automatically presentable where  $+$  is the usual addition on the naturals  $\mathbb{N}$ , and  $x|_ky$  means that  $x = k^n$  for some  $n \in \mathbb{N}$  and  $y = mx$  for some  $m \in \mathbb{N}$  (that is,  $x$  is a power of  $k$  and  $x$  divides  $y$ ). One gets an automatic presentation for this structure by coding the natural numbers in base  $k$  (least significant digit first). Indeed, a finite automaton can check addition using the usual carry-bit procedure, and it is similarly straightforward to verify that the relation  $|_k$  is regular.

- (iv) The configuration space of a Turing Machine  $N$ , considered as a directed graph, whose edges represent one step transitions of  $N$ , is automatically presentable. The idea is that an automaton can compute these transitions simply following a window of fixed size around the current position of the read-head.
- (v) The linear ordering of the rationals is automatically presentable. The structure with domain  $\{0, 1\}^*$  and the binary relation  $x \sqsubseteq_Q y$  if  $(x \sqcap y)0$  is a prefix of  $x$  or  $(x \sqcap y)1$  is a prefix of  $y$  (here  $x \sqcap y$  denotes their longest common prefix) is an automatic copy.
- (vi) Every finitely generated Abelian group  $(G; +)$  is automatically presentable. This is straightforward given the classification of these groups as finite sums of  $(\mathbb{Z}, +)$  and finite cyclic groups.
- (vii) Every ordinal  $(L; \leq)$  less than  $\omega^\omega$  is automatically presentable. This is straightforward given the classification of these ordinals as being of the form  $\omega^{n_1} + \dots + \omega^{n_l}$  where  $\omega > n_1 \geq n_2 \geq \dots \geq n_l$ .
- (viii) The Boolean algebra  $(\mathcal{B}_{\text{fin}})^n$  ( $n \in \mathbb{N}$ ) is automatically presentable; here  $\mathcal{B}_{\text{fin}} = (B_{\text{fin}}; \wedge, \vee, \neg)$  is the Boolean algebra of finite or co-finite subset of  $\mathbb{N}$ .

#### §3. Properties.

**3.1. Fundamental properties.** As a consequence of Theorem 2.4 one has the following result that may be called the fundamental theorem of automatically presentable structures.

**THEOREM 3.1 (Definability).** *For every automatic presentation  $\mu$ , of structure  $\mathcal{A}$  say, every first order  $\mathcal{A}$ -formula (allowing parameters) defines a relation  $R$  with  $\mu^{-1}(R)$  regular.*

Moreover there is an algorithm that from the automata of an automatic presentation, and a first-order formula allowing parameters, outputs an automaton for the convolution of the relation defined by the formula.

This is simply proved by structural induction on the formula defining the relation. Consequently:

**THEOREM 3.2 (Decidability).** *The first-order theory (allowing parameters) of each automatically presentable structure is decidable.*

Indeed, from any automatic presentation of  $\mathcal{A}$ , and sentence of the form  $\exists x\Phi(x)$ , one can check effectively whether or not the constructed automaton for  $\Phi^{\mathcal{A}}(x)$  accepts any string at all.

**Remark 3.3.** We shall see in subsection 3.2 that the Definability and Decidability Theorems can be extended to include additional quantifiers such as ‘there exists infinitely many’ and ‘there exists  $k$  modulo  $m$  many’.

Another consequence of the Definability Theorem is that automatically presentable structures are closed under first-order interpretability.

**DEFINITION 3.4.** An interpretation of structure  $\mathcal{B} = (B; (R_i^{\mathcal{B}})_i)$  in structure  $\mathcal{A}$  consists of the following  $\mathcal{A}$ -formulas,

- (i) a domain formula  $\Delta(\bar{x})$ ,
- (ii) a relation formula  $\Phi_{R_i}(\bar{x}_1, \dots, \bar{x}_{r_i})$  for each relation symbol  $R_i$ , and
- (iii) an equality formula  $\epsilon(\bar{x}_1, \bar{x}_2)$ ,

where each  $\Phi_{R_i}^{\mathcal{A}}$  is a relation on  $\Delta^{\mathcal{A}}$ , and  $\epsilon^{\mathcal{A}}$  is a congruence on the structure  $(\Delta^{\mathcal{A}}; (\Phi_{R_i}^{\mathcal{A}})_i)$ , so that  $(\Delta^{\mathcal{A}}; (\Phi_{R_i}^{\mathcal{A}})_i)/\epsilon^{\mathcal{A}}$  and  $\mathcal{B}$  are isomorphic.

Each of the tuples  $\bar{x}_i, \bar{x}$  contain the same number of variables, called the dimension of the interpretation.

We say that  $\mathcal{B}$  is interpretable in  $\mathcal{A}$ .

Also, say that  $\mathcal{B}$  is interpretable in  $\mathcal{A}$  with parameters  $\bar{a}$  if  $\mathcal{B}$  is interpretable in  $(\mathcal{A}, \bar{a})$ .

In case  $\mathcal{B}$  has infinite signature, it is also required that the function sending  $i$  to  $\Phi_{R_i}$  be computable.

To stress that all the formulas are first-order, we say that  $\mathcal{B}$  is *first-order interpretable* in  $\mathcal{A}$ . In this case, every first-order  $\mathcal{B}$ -formula  $\phi$  can be translated into a first-order  $\mathcal{A}$ -formula  $\phi'$  such that for all  $\bar{a} \in \Delta^{\mathcal{A}}$ ,

$$\mathcal{B} \models \phi(\mu(\bar{a})) \iff \mathcal{A} \models \phi'(\bar{a}),$$

where  $\mu$  is the isomorphism from the interpretation. The formula  $\phi'$  is formed from  $\phi$  by relativising the quantification to  $\Delta$ , replacing every relation symbol  $R_i$  by its defining formula  $\Phi_{R_i}$ , and replacing equality by  $\epsilon$ .

We will have brief occasion to consider the case that the formulas are (weak) monadic second-order, whence we say that  $\mathcal{B}$  is (*weak*) *monadic*

*second-order interpretable* in  $\mathcal{A}$ .<sup>3</sup> In this case, elements of  $\mathcal{B}$  are coded by (finite) subsets of  $A$ . As an illustration,  $(\mathbb{N}, +)$  is weak monadic second-order interpretable in  $(\mathbb{N}, S)$  via the map sending a finite set  $X \subset \mathbb{N}$  to the natural number  $\sum_{i \in X} 2^i$ .

EXAMPLE 3.5. The structures  $\mathcal{N}_k$  and  $\mathcal{W}_l$  (see Example 2.4) are first-order interpretable in each other, for every  $k, l \geq 2$ .

The following proposition is immediate from the definitions and Theorem 3.1 [Definability].

PROPOSITION 3.6. [9] *If  $\mathcal{B}$  is first-order interpretable in  $\mathcal{A}$  (possibly with parameters), and  $\mathcal{A}$  is automatically presentable, then  $\mathcal{B}$  is also automatically presentable.*

The following corollary lists some useful closure properties of automatically presentable structures, some of which were already noted in [34]. It is worth mentioning that, more generally, Blumensath and Grädel [9, 12] show that automatically presentable structures are closed under Feferman-Vaught like products.

Recall that definable means first-order definable allowing parameters.

COROLLARY 3.7. *If structures  $\mathcal{A}$  and  $\mathcal{B}$  (with the same signature) are automatically presentable, then the following are also automatically presentable:*

- (i) *the expansion  $(\mathcal{A}, R)$  by any relation definable in  $\mathcal{A}$ ,*
- (ii) *the substructure of  $\mathcal{A}$  with any definable universe,*
- (iii) *the factorisation of  $\mathcal{A}$  by any definable congruence,*
- (iv) *the direct product  $\mathcal{A} \times \mathcal{B}$ , the disjoint union  $\mathcal{A} \cup \mathcal{B}$ ,*
- (v) *the  $\omega$ -fold disjoint union of  $\mathcal{A}$ , written  $\cup_\omega \mathcal{A}$ ,*
- (vi) *the ordered sum  $\mathcal{A} + \mathcal{B}$  and the  $\omega$ -fold ordered sum  $\Sigma_{i < \omega} \mathcal{A}$  (for this case  $\mathcal{A}$  and  $\mathcal{B}$  are ordered structures).*

Next is a logical characterisation of the automatically presentable structures.

THEOREM 3.8. *Let  $\mathcal{B}$  be a structure. Then the following are equivalent:*

- (i)  $\mathcal{B}$  is automatically presentable.
- (ii)  $\mathcal{B}$  is first-order interpretable in  $\mathcal{W}_k$  for some, equivalently all,  $k \geq 2$ .
- (iii)  $\mathcal{B}$  is first-order interpretable in  $\mathcal{N}_k$  for some, equivalently all,  $k \geq 2$ .
- (iv)  $\mathcal{B}$  is weak monadic second-order interpretable in  $(\mathbb{N}, S)$ , where  $S$  is the usual successor function  $n \mapsto n + 1$ .

Here is a brief history of this theorem. The equivalences (i)–(iii) are from Blumensath and Grädel [12] who introduced interpretability to characterise the automatically presentable structures. The main ideas arise from older

---

<sup>3</sup>These are called *(finite) set interpretations* in [21] if the interpretation also has dimension one. The present definition is not to be confused with the case that the formulas are (W)MSO but all free variables are first-order, sometimes also called (W)MSO-interpretations in the literature.

results stated for relations instead of structures: Büchi [16] and Elgot [27] prove that a set of tuples  $(A_1, \dots, A_n)$  of finite sets of natural numbers is weak monadic second-order definable in  $(\mathbb{N}, S)$  if and only if the corresponding  $n$ -ary relation of characteristic strings (a subset of  $\{0, 1\}^{*n}$ ) is synchronous rational. The relationship between weak MSO and finite automata was also realised by Trahtenbrot [56]. Eilenberg, Elgot, and Shepherdson [26] prove that a relation  $R \subseteq (\Sigma^*)^n$  is synchronous rational if and only if  $R$  is first-order definable in  $\mathcal{W}_k$ , where  $k = |\Sigma| \geq 2$ . The Büchi-Bruyère Theorem (proofs of which can be found in [44] and [57]) states that a relation  $R \subseteq \mathbb{N}^n$  (coded in base  $k \geq 2$  least significant digit first) is synchronous rational if and only if it is first-order definable in  $\mathcal{N}_k$ .

The proofs of these results, which are by now standard, usually go as follows. From formulas to automata one proceeds by structural induction on the given formula, using Theorem 2.4 for the logical operations. Conversely, starting with an automaton, one constructs a formula stating the existence of a successful run of the automaton (alternatively, [57] proceeds by induction on regular expressions).

We turn to unary-automatically presentable structures. An important example is the structure  $\mathcal{U} = (\mathbb{N}; \leq, (\equiv_n)_{n \in \mathbb{N}})$  where  $x \equiv_n y$  if  $x$  is congruent to  $y$  modulo  $n$ .

**THEOREM 3.9.** [9, 45] *A structure is automatically presentable over a unary alphabet if and only if it is first-order interpretable (via a 1-dimensional interpretation) in the structure  $(\mathbb{N}; \leq, (\equiv_n)_{n \in \mathbb{N}})$ .*

Blumensath [11] calls a structure  $\mathcal{A}$  *tree-interpretable* if there is a monadic second-order interpretation of dimension 1 of  $\mathcal{A}$  into  $\mathcal{T}_2$  with the restriction that all the free variables in the interpreting formulas are first-order. Here  $\mathcal{T}_2 = (\{0, 1\}^*; \sigma_0, \sigma_1)$  where  $\sigma_\epsilon(w) = w\epsilon$  for all  $w \in \{0, 1\}^*$  and  $\epsilon \in \{0, 1\}$ . Since  $\mathcal{T}_2$  has decidable monadic second-order theory [49], so does  $\mathcal{A}$ .

Since  $\mathcal{U}$  is tree-interpretable, every unary-automatically presentable structure has decidable monadic second-order theory [10, Prop. 5]. Moreover, Blumensath proves that every tree-interpretable structure is automatically presentable (the converse is false because there are automatically presentable structures, such as the infinite grid, with undecidable monadic-second order theory). The tree-interpretable graphs coincide with other classes of infinite graphs that appear in the literature; see [10] for a discussion.

**3.2. Extending first-order.** The Definability Theorem can be strengthened to include order-invariantly definable formulas as well as certain additional quantifiers.

#### *Order-invariance.*

**DEFINITION 3.10.** Fix a signature  $\tau$  and a new symbol  $\leq$ . A formula  $\phi(\bar{x})$  in the signature  $\tau \cup \{\leq\}$ , is called  $\omega$ -order invariant on a  $\tau$ -structure  $\mathcal{A}$ , if for all tuples  $\bar{a}$  from  $A$ , and all linear orders  $\leq_1$  and  $\leq_2$  on  $A$  of order type  $\omega$  (or  $|A|$  if  $A$  is finite), we have that  $(\mathcal{A}, \leq_1) \models \phi(\bar{a})$  if and only if  $(\mathcal{A}, \leq_2) \models \phi(\bar{a})$ .

The *relation in  $\mathcal{A}$  defined by the  $\omega$ -order invariant  $\phi$*  is the set of tuples  $\bar{a}$  from  $A$  such that  $(\mathcal{A}, \leq) \models \phi(\bar{a})$  for some (and hence all) linear orders  $\leq$  on  $A$  of order type  $\omega$  (or  $|A|$  if  $A$  is finite).

Write  $\mathcal{L}_{\omega\text{-inv}}(\mathcal{A})$  for the relations that are definable by  $\omega$ -order invariant  $\mathcal{A}$ -formulas in logic  $\mathcal{L}$ .

*Remark 3.11.* Every automatic presentation  $\mu$  of structure  $\mathcal{A}$  can be expanded to an automatic presentation of an expansion  $(\mathcal{A}, \leq)$  where  $\leq$  linearly orders  $A$ . For instance let  $\mu^{-1}(\leq)$  be the regular linear order  $<_{\text{lex}}$  restricted to the domain  $\mu^{-1}(A)$ . Observe that this linear order has order-type  $\omega$  if  $A$  is infinite.

Thus we can replace  $\text{FO}$  by  $\text{FO}_{\omega\text{-inv}}$  in Theorem 3.1 [Definability].

*Generalised quantifiers.* We briefly recall the definition of generalised quantifiers as introduced by Lindström [41].

**DEFINITION 3.12.** Fix a finite signature  $\tau = (R_i)_{i \leq k}$ , where  $R_i$  has associated arity  $r_i$ . A *quantifier*  $Q$  is a class of  $\tau$ -structures closed under isomorphism. Let  $\sigma$  be another signature. Given  $\sigma$ -formulas  $\Psi_i(\bar{x}_i, \bar{z})$  with  $|\bar{x}_i| = r_i$  ( $i \leq k$ ), the syntax  $Q\bar{x}_1, \dots, \bar{x}_k(\Psi_1, \dots, \Psi_k)$  has the following meaning on a  $\sigma$ -structure  $\mathcal{A}$ :

$$(\mathcal{A}, \bar{a}) \models Q\bar{x}_1, \dots, \bar{x}_k(\Psi_1, \dots, \Psi_k) \text{ iff } (A; \Psi_1^{\mathcal{A}}(\cdot, \bar{a}), \dots, \Psi_k^{\mathcal{A}}(\cdot, \bar{a})) \in Q,$$

where  $\Psi^{\mathcal{A}}(\cdot, \bar{a})$  is the relation defined in  $\mathcal{A}$  by  $\Psi$  with parameters  $\bar{a}$ . The *arity* of a quantifier is the maximum of the  $r_i$ s. A quantifier is  $n$ -ary if its arity is at most  $n$ .

The *relation defined (in  $\mathcal{A}$ ) by this formula* is the set of tuples  $\bar{a}$  from  $A$  such that  $(\mathcal{A}, \bar{a}) \models Q\bar{x}_1, \dots, \bar{x}_k(\Psi_1, \dots, \Psi_k)$ .

A collection of quantifiers is denoted by  $Q$ , and the extension of first-order logic by the quantifiers in  $Q$  is written  $\text{FO}[Q]$ .

As an illustration, ‘there exists’ is given by the unary quantifier

$$\{(A; X) \mid \emptyset \neq X \subseteq A\}.$$

Here are some quantifiers that will feature in this section:

**EXAMPLES 3.13.** (i) The unary quantifier ‘there exists infinitely many’, written  $\exists^\infty$ , is the class of structures  $(A; X)$  where  $X$  is an infinite subset of  $A$ .

(ii) The unary *modulo quantifier* ‘there are  $k$  modulo  $m$  many’ (here  $0 \leq k < m$ ), written  $\exists^{(k,m)}$ , is the class of structures  $(A; X)$  where  $X$  contains  $k$  modulo  $m$  many elements. Write  $\exists^{\text{mod}}$  for the collection of modulo quantifiers.

(iii) The unary *Härtig quantifier* is the class of structures  $(A; P, Q)$  where  $P, Q \subseteq A$  and  $|P| = |Q|$ .

- (iv) The *cardinality quantifiers*: Every set  $C \subseteq (\mathbb{N} \cup \{\infty\})^n$  induces the unary quantifier  $Q_C = \{(A; P_1, \dots, P_n) \mid (|P_1|, \dots, |P_n|) \in C\}$ . In fact, a given unary quantifier over signature  $(R_i)_{i \leq k}$  is identical to some cardinality quantifier with  $n = 2^k$ .
- (v) The binary *reachability quantifier* is the class of structures of the form  $(A; E, \{c_s\}, \{c_f\})$  where  $E \subseteq A^2$ ,  $c_s, c_f \in A$ , and there is a path in the directed graph  $(A; E)$  from  $c_s$  to  $c_f$ .
- (vi) The  $k$ -ary *Ramsey quantifier*  $\exists^{k\text{-ram}}$  is the class of structures  $(A; E)$ ,  $E \subseteq A^k$ , for which there is an infinite  $X \subseteq A$  such that for all pairwise distinct  $x_1, \dots, x_k \in X$ ,  $E(x_1, \dots, x_k)$ .

Of course the generalised quantifiers that interest us most are the ones, like  $\forall$  and  $\exists$ , that preserve regularity; meaning, loosely, that quantifying over regular relations yields regular relations.

**DEFINITION 3.14.** Let  $Q$  be a quantifier with signature  $\tau = (R_i)_{i \leq k}$ , where  $R_i$  has associated arity  $r_i$ . Say that the quantifier *preserves regularity* if for every  $n \in \mathbb{N}$ , and every automatic presentation  $\mu$  of a structure  $\mathcal{A}$ , every formula

$$Q\bar{x}_1, \dots, \bar{x}_k(\Psi_1^{\mathcal{A}}(\bar{x}_1, \bar{z}), \dots, \Psi_k^{\mathcal{A}}(\bar{x}_k, \bar{z}))$$

defines a relation  $R$  in  $\mathcal{A}$  so that  $\mu^{-1}(R)$  is regular (here  $\bar{z} = (z_1, \dots, z_n)$  and the  $\Psi_i$ s are first-order  $\mathcal{A}$ -formulas).

Moreover, say that  $Q$  *preserves regularity effectively* if an automaton for  $\mu^{-1}(R)$  can effectively be constructed from the automata of the presentation and the formulas  $\Psi_i$ .

**Remark 3.15.** Since every automatic presentation restricts (effectively) to an injective automatic presentation (proof of Proposition 2.6), to show that  $Q$  preserves regularity (effectively) it is sufficient to satisfy the above definition with ‘injective automatic presentation’ replacing ‘automatic presentation’.

**EXAMPLE 3.16. [9]** The quantifier  $\exists^\infty$  preserves regularity effectively. Let  $\mu$  be an injective automatic presentation of structure  $\mathcal{A}$ , and  $\Psi(x, \bar{z})$  a first-order  $\mathcal{A}$ -formula. Then the relation defined in  $\mathcal{A}$  by  $(\exists^\infty x)\Psi(x, \bar{z})$  is equal to the relation defined by the  $\omega$ -order invariant formula

$$(\forall w)(\exists x)[w < x \wedge \Psi(x, \bar{z})].$$

Here is a non-example.

**EXAMPLE 3.17.** The reachability quantifier is not regularity preserving. For otherwise, by Example 2.4 (4), the set of starting configurations that drive a given Turing Machine to a halting state would be regular, and hence computable.

*Remark 3.18.* In our new terminology, Theorem 3.1 [Definability] says that  $\exists$  and  $\forall$  preserve regularity effectively. Moreover if every quantifier in a collection  $\mathcal{Q}$  preserves regularity effectively, then we can replace FO by  $\text{FO}[\mathcal{Q}]$  in Theorem 3.1 [Definability] and Theorem 3.2 [Decidability]. If quantifiers in  $\mathcal{Q}$  simply preserve regularity, then we can replace FO by  $\text{FO}[\mathcal{Q}]$  in Proposition 3.6, Corollary 3.7 and parts two and three of Theorem 3.8.

**THEOREM 3.19.** [36] *Every quantifier in  $\exists^{\text{mod}}$  preserves regularity effectively.*

**PROOF.** Let  $\mu$  be an injective automatic presentation of structure  $\mathcal{D}$ , and let  $\Phi(y_1, \dots, y_n)$  denote

$$\exists^{(k,m)} x \Psi(x, y_1, \dots, y_n),$$

for fixed  $k, m \in \mathbb{N}$  and first-order  $\mathcal{D}$ -formula  $\Psi$ . Let  $\mathcal{A}$ , with  $A \subseteq \Sigma^*$  be the automatic copy of  $\mathcal{D}$  induced by the presentation. The aim is to show that the relation defined in  $\mathcal{A}$  by  $\Phi$  is regular, and to construct the automaton effectively from the automata of the presentation.

First, the required automaton should not accept those  $\bar{y}$  for which there are infinitely many  $x$  with  $\mathcal{A} \models \Psi(x, \bar{y})$ . So, define  $\Psi_f(x, \bar{y})$  to be

$$\Psi(x, \bar{y}) \wedge \neg \exists^\infty x \Psi(x, \bar{y}).$$

Now, it is sufficient to construct an automaton, which will be called  $\mathbf{B}'$ , that recognises the relation defined in  $\mathcal{A}$  by  $\exists^{(k,m)} x \Psi_f(x, \bar{y})$ . The following property of  $\Psi_f$  will be used: For every tuple  $\bar{y}$ , there are finitely many  $x$  with  $\mathcal{A} \models \Psi_f(x, \bar{y})$ .

Let  $\mathbf{B} = (Q, i, \delta, F)$  be a deterministic automaton recognising the convolution of  $(\Psi_f)^\mathcal{A}$ . The idea is that on input  $\bar{y}$ , the required automaton,  $\mathbf{B}'$ , will count, modulo  $m$ , the number of accepting paths of  $\mathbf{B}$  on inputs of the form  $(x, \bar{y})$  for  $x \in \Sigma^*$ . Here is some notation to help describe the construction.

*Notation:* For states  $q, q' \in Q$ , and input  $\bar{y} \in \Sigma^{*n}$  define  $\#(q, \otimes \bar{y}, q')$  to be the number of strings  $x \in \Sigma^*$  with  $|x| = |\otimes \bar{y}|$ , such that there is a path in  $\mathbf{B}$  labeled  $\otimes(x, \bar{y})$  from state  $q$  to state  $q'$ . For  $S \subseteq Q$  define  $\#(S, \otimes \bar{y}, q')$  to be the sum  $\sum_{q \in S} \#(q, \otimes \bar{y}, q')$ .

More to the point then, the states of  $\mathbf{B}'$  are of the form  $(S_1, \dots, S_m)$  where  $S_i \subseteq Q$ . If  $\mathbf{B}'$  is in state  $(S_1, \dots, S_m)$  after reading input  $\bar{y}$ , then  $S_i$  will consist of those states  $q$  of  $\mathbf{B}$  for which  $\#(i, \otimes \bar{y}, q)$  is congruent to  $i$  modulo  $m$ . Finally, once  $\otimes \bar{y}$  has been completely read, the automaton needs to account for more input to  $\mathbf{B}$  of the form  $\otimes(x', \lambda, \dots, \lambda)$ .

*Definition:* The required automaton  $\mathbf{B}' = (Q', i', \Delta', F')$  over alphabet  $(\Sigma_\perp)^n$  is defined as follows.

- (i) The state set  $Q'$  is  $\prod_{1 \leq i \leq m} \mathbb{P}(Q)$ , where  $\mathbb{P}(Q)$  denotes the powerset of  $Q$ .
- (ii) The initial state  $i'$  is  $\{i\} \times \prod_{2 \leq i \leq m} \{\emptyset\}$ .

- (iii) For a state  $T = (T_1, \dots, T_m) \in Q'$  and input symbol  $\sigma \in (\Sigma_\perp)^n$  define the transition function  $\Delta'(T, \sigma) = (S_1, \dots, S_m)$  as follows. For every set  $S_i$  and  $q \in Q$ , let  $q \in S_i$  if and only if

$$\sum \{j \times \#(T_j, \sigma, q) \mid 1 \leq j \leq m\} \equiv i \pmod{m}.$$

- (iv) Let  $(S_1, \dots, S_m) \in F'$  if and only if

$$\sum_{f \in F} \sum_{r < |Q|} \sum_{1 \leq j \leq m} j \times \#(S_j, (\{\perp\}^n)^r, f) \equiv k \pmod{m}.$$

Here  $(\{\perp\}^n)^r$  is the concatenation of  $r$  copies of the new symbol  $(\perp, \dots, \perp) \in (\Sigma_\perp)^n$ .

Note that the automaton is deterministic.

*Correctness:* Let  $w = \otimes \bar{y}$  be the input to  $\mathcal{B}'$  and let  $(S_1, \dots, S_m)$  be the state  $\Delta'(t', w)$ . The following can be proved by induction on  $|w|$ :

- (i) If  $|w| \geq 1$ , then for each  $S_i$ ,

$$q \in S_i \text{ if and only if } \#(t, w, q) \equiv i \pmod{m}.$$

- (ii)  $(S_1, \dots, S_m) \in F'$  if and only if

$$\sum_{f \in F} \sum_{r \in \mathbb{N}} \#(t, w \cdot (\{\perp\}^n)^r, f) \equiv k \pmod{m}.$$

Finally, note that the index  $r \in \mathbb{N}$  can be restricted to  $0 \leq r < |Q|$ , since the number of strings  $x$  with  $\Psi_f(x, \bar{y})$  is finite.  $\dashv$

A result in [37] says that the set of extendible nodes of an automatically presentable tree  $(T; \preceq)$  is regular. We adapt the proof for the next Theorem.

**THEOREM 3.20.** *Each  $k$ -ary Ramsey quantifier preserves regularity effectively.*

**PROOF.** Note that the 1-Ramsey quantifier is identical to ‘there exists infinitely many’. We illustrate the proof for  $k = 2$ , the general case being similar.

Let  $\mu$  be an injective automatic presentation of some structure  $\mathcal{D}$ , and  $\Psi(x, y, z_1, \dots, z_n)$  a first-order  $\mathcal{D}$ -formula. Let  $\mathcal{A}$  be an automatic copy of  $\mathcal{D}$  with  $A \subseteq \Sigma^*$ . The aim is to show constructively that the binary relation defined in  $\mathcal{A}$  by the formula

$$\exists^{\text{2-ram}} xy \Psi(x, y, \bar{z}),$$

is regular.

Now,  $\mathcal{A} \models \exists^{\text{2-ram}} xy \Psi(x, y, \bar{z})$  if and only if there exists an infinite string  $\alpha \in \Sigma^\omega$  and an infinite set  $I \subseteq \mathbb{N}$ :

- (i) for every  $i \in I$ , there is a string  $w_i := p_i t_i$  where  $p_i$  is the initial prefix of  $\alpha$  of length  $i$ , and  $t_i$  is a non-empty string;
- (ii) and for every  $i \neq j \in I$ , it holds that  $w_i \neq w_j$  and  $\Psi(w_i, w_j, \bar{z})$ .

Since this expression quantifies over infinite strings, we will use automata operating over infinite strings, so called  $\omega$ -automata or Büchi-automata. Here is a brief reminder of what this means. A Büchi automaton has the same form  $(Q, \iota, \Delta, F)$  as a non-deterministic finite automaton. A run on an infinite string is accepting if some accepting state occurs infinitely in the run. A relation  $R \subseteq (\Sigma^\omega)^m$  is *Büchi recognisable* if there is a Büchi automaton recognising the convolution of  $R$ , namely the relation  $\otimes R \subseteq (\Sigma^m)^\omega$  where the  $i$ th symbol of  $\otimes R$  is defined as  $(\sigma_1, \dots, \sigma_m)$  where  $\sigma_j$  is the  $i$ th symbol of the  $j$ th component of  $R$ . The languages recognised by Büchi automata satisfy an analogous version of Theorem 2.4. For details, see for instance [55].

We need to mark the boundaries as given by  $I$ . Introduce a new symbol  $\wr$  for this purpose, and let  $\Delta = \Sigma \cup \{\wr\}$ . Call a pair  $(s, t)$  of infinite strings  $s, t \in \Delta^\omega$  *good* if  $s$  can be expressed as  $s_0 \wr s_1 \wr s_2 \wr \dots$  and  $t$  as  $t_0 \wr t_1 \wr t_2 \wr \dots$ , where  $s_i, t_i \in \Sigma^*$  and  $|s_i| = |t_i| > 0$  for each  $i$ . In this case, say that every string of the form  $s_0 s_1 \dots s_l t_{l+1}$ , for  $l \in \mathbb{N}$ , is *on* the good pair  $(s, t)$ .

Now define the language  $R \subseteq (\Delta^\omega)^{2+n}$  as consisting of tuples of the form  $(s, t, u_1, \dots, u_n)$  with the following properties:

- (i)  $(s, t)$  is a good pair;
- (ii) every  $u_i$  is of the form  $z_i \{\wr\}^\omega$  for some  $z_i \in \Sigma^*$ ;
- (iii) and if  $x$  and  $y$  are distinct strings on  $(s, t)$ , then  $\Psi(x, y, z_1, \dots, z_n)$  holds.

Then  $\mathcal{A} \models \exists^{2\text{-ram}} xy\Psi(x, y, \bar{z})$  if and only if there exists  $(s, t)$  so that  $R(s, t, z_1 \{\wr\}^\omega, \dots, z_n \{\wr\}^\omega)$ .

To finish the proof, one can show that  $R$  is Büchi recognisable. Moreover, since Büchi automata are closed under projection, there is a Büchi automaton that accepts  $(z_1 \{\wr\}^\omega, \dots, z_n \{\wr\}^\omega)$  if and only if  $\mathcal{A}$  models  $\exists^{2\text{-ram}} xy\Psi(x, y, \bar{z})$ . This automaton can be transformed into a finite automaton recognising the relation defined in  $\mathcal{A}$  by  $\exists^{2\text{-ram}} xy\Psi(x, y, \bar{z})$ .  $\dashv$

As an aside, we now show that one can extract from this proof an automatic version of Ramsey's Theorem.

Recall that the infinitary version of Ramsey's Theorem (for  $k$ -tuples and two colours) says that if  $D$  is infinite and  $E \subseteq D^k$ , then there exists an infinite monochromatic set  $X \subseteq D$ ; namely, either  $E(\bar{x})$  for all pairwise distinct  $x_1, \dots, x_k \in X$ , or  $\neg E(\bar{x})$  for all pairwise distinct  $x_1, \dots, x_k \in X$ .

**PROPOSITION 3.21.** *Let  $\mu$  be an automatic presentation of a structure  $\mathcal{D} = (D; E)$ , with  $D$  infinite and  $E \subseteq D^k$ . Then there exists an infinite monochromatic set  $X \subseteq D$  with  $\mu^{-1}(X)$  regular.*

**PROOF.** For simplicity we consider the case  $k = 2$ . By Ramsey's Theorem there is a monochromatic set  $X \subseteq D$ . Suppose that  $E(x, y)$  for all  $x \neq y \in X$  (the other case is symmetric).

Let  $\mathcal{A} = (A; E)$  be the automatic copy of  $\mathcal{D}$  induced by  $\mu$ . By the previous proof, the set of good pairs  $(s, t)$  such that  $x \neq y \in A$  on  $(s, t)$  implies

$E(x, y)$ , is a Büchi recognisable relation, say  $R$ . If  $R(s, t)$  then the set of strings  $w \in A$  that are on  $(s, t)$  forms an infinite monochromatic set.

By assumption  $R(\cdot, \cdot)$  is non-empty. Since every Büchi automaton accepting some string also accepts some ultimately periodic string, we get that  $R$  contains some ultimately periodic  $\otimes(s', t') \in (\Delta^2)^\omega$ . So both  $s'$  and  $t'$  are ultimately periodic. This pair  $(s', t')$  can be transformed into a finite automaton accepting all the strings  $w \in A$  that are on  $(s', t')$ .  $\dashv$

**DEFINITION 3.22.** Write  $\mathcal{Q}_n^{\text{reg}}$  for the collection of  $n$ -ary generalised quantifiers that preserve regularity.

For instance,  $\mathcal{Q}_1^{\text{reg}}$  contains the usual quantifiers  $\exists, \forall$ , as well as the modulo quantifiers  $\exists^{\text{mod}}$  and  $\exists^\infty$ ; and  $\mathcal{Q}_n^{\text{reg}}$  contains  $\exists^{\text{n-ram}}$ .

**Problem 3.23.** Classify the quantifiers in  $\mathcal{Q}_n^{\text{reg}}$ , for each  $n$ .

The following general definition will allow us to compare the expressive strength of quantifiers.

**DEFINITION 3.24.** Let  $\mathcal{Q}$  be a quantifier,  $\mathcal{Q}$  a collection of quantifiers, and  $\tau$  the signature of  $\mathcal{Q}$ . Say that  $\mathcal{Q}$  is definable in  $\mathcal{Q}$  if there is a sentence  $\theta$  over the signature  $\tau$  in the logic  $\text{FO}[\mathcal{Q}]$  with  $\mathcal{Q} = \{\mathcal{A} \mid \mathcal{A} \models \theta\}$ .

For instance, a structure  $(A; X)$  satisfies the sentence  $\neg[\exists^{(0,2)}zX(z) \vee \exists^{(1,2)}zX(z)]$  if and only if  $X$  is infinite. So,  $\exists^\infty$  is definable in  $\{\exists^{(0,2)}, \exists^{(1,2)}\}$ .

As a first step, we characterise the unary quantifiers that preserve regularity.

**PROPOSITION 3.25.** Every quantifier in  $\mathcal{Q}_1^{\text{reg}}$  is definable in  $\exists^{\text{mod}}$ .

**PROOF.** Every unary quantifier is equal to some cardinality quantifier where  $C \subseteq (\mathbb{N} \cup \{\infty\})^r$  (Example 3.13 (4)). So suppose that  $\mathcal{Q}_C$  preserves regularity. Since  $\exists^\infty$  is definable in  $\exists^{\text{mod}}$ , we may assume that  $C \subseteq \mathbb{N}^r$ . Let  $\Sigma = \{1, \dots, r\}$ . Write  $\#_i(z)$  for the number of occurrences of the symbol  $i$  in the string  $z \in \Sigma^*$ . Then the set

$$C' = \{z \in \Sigma^* \mid (\#_1(z), \dots, \#_r(z)) \in C\}$$

is regular since it is definable as  $\mathcal{Q}_C \bar{x}(M_1(x_1, z), \dots, M_r(x_r, z))$ , where for each  $i \leq r$ ,  $M_i$  is the regular binary relation of those pairs  $(x, z)$  such that  $x \in 11^*$  and  $z$  has the symbol  $i$  in position  $|x|$ .

Now if  $C'$  is regular, then  $C$  is recognisable - meaning that  $C$  is the union of classes of a congruence on  $(\mathbb{N}^r, +)$  of finite index.

By Mezei's description of the recognisable subsets of a product of monoids (see for instance [8]),  $C$  is recognisable if and only if it is a finite union of sets of the form  $X_1 \times \dots \times X_r$ , where each  $X_i \subseteq \mathbb{N}$  is ultimately periodic. In other words,  $\mathcal{Q}_C$  is definable in  $\exists^{\text{mod}}$ .  $\dashv$

**EXAMPLE 3.26.** The Härtig quantifier does not preserve regularity. Indeed, the previous proof says that this is because one could use it to define, in a

suitable automatically presentable structure, the (non-regular) set of strings  $x \in \{0, 1\}^*$  for which  $x$  has the same number of 0s as 1s.

What about  $\mathcal{Q}_n^{\text{reg}}$  for  $n \geq 2$ ? I do not know of a characterisation as for  $n = 1$ . Here is a simple example separating  $\mathcal{Q}_2^{\text{reg}}$  from  $\mathcal{Q}_1^{\text{reg}}$ .

EXAMPLE 3.27. Define the quantifier  $\exists^\rho$  stating that  $(A; \psi(\cdot, \cdot, \bar{z}))$  is an equivalence structure with infinitely many infinite classes'.

Then  $\exists^\rho$  is in  $\mathcal{Q}_2^{\text{reg}}$ , since it is  $\text{FO}_{\omega-\text{inv}}(\exists^\infty)$  definable.

However, it is not expressible in first-order plus all unary quantifiers over the class of automatically presentable structures. This can be seen by checking that for every  $r \in \mathbb{N}$ , the Duplicator has a winning strategy in the 1-bijective  $r$ -round game<sup>4</sup> between the following structures:  $\mathcal{A}_r$  comprises exactly  $r$  infinite classes, and  $\mathcal{B}_r$  comprises infinitely many infinite classes.

**3.3. Automatic presentations.** An automatically presentable structure  $\mathcal{B}$  has many automatic presentations. Here we look at the relationship amongst these presentations, and in the next subsection at their relationship to definability in the structure.

For a given automatic presentation  $\mu: A \rightarrow B$  of  $\mathcal{B}$ , write  $\mu(\text{Reg})$  for the collection of relations

$$\{\mu(R) \mid R \subseteq A^r \text{ is a regular relation, } r \in \mathbb{N}\}.$$

DEFINITION 3.28. [2] Let  $\mathcal{B}$  be an automatically presentable structure. Call two automatic presentations  $\mu, \nu$  of  $\mathcal{B}$  *equivalent* if  $\mu(\text{Reg}) = \nu(\text{Reg})$ .

EXAMPLE 3.29. Fix an automatic presentation  $\mu$  of an infinite structure  $\mathcal{A}$ . There are infinitely many equivalent presentations  $\nu$  so that the binary relation  $\{(u, v) \mid \mu(u) = \nu(v)\}$  that translates between the presentations is not regular. For instance, for  $c \notin \Sigma$  and  $n \in \mathbb{N}$ , let  $\nu_n$  be the presentation sending  $a_0 c^n a_1 c^n \dots a_k c^n$  to  $\mu(a_0 a_1 \dots a_k)$  for  $a_0 \dots a_k \in \text{dom}(\mu)$ . Here  $c^n$  denotes the concatenation of  $n$  many  $c$  symbols.

Proposition 2.6 says that we can transform an arbitrary automatic presentation into one that is injective and over a binary alphabet. Its proof yields an equivalent presentation.

PROPOSITION 3.30. *Let  $\mu$  be an automatic presentation of  $\mathcal{B}$ . Then there is an equivalent presentation  $\nu$  of  $\mathcal{B}$  with the following properties:*

- (i)  $\nu$  is injective, and
- (ii)  $\nu$  is a presentation over a binary alphabet  $|\Sigma| = 2$ .

---

<sup>4</sup>The  $n$ -bijective games, introduced by Hella [29], are Ehrenfeucht–Fraïssé like games characterising definability in first-order logic extended by all  $n$ -ary generalised quantifiers. Here is a brief description: in round  $i$  the Duplicator chooses a bijection  $b_i: A \rightarrow B$ , and the spoiler answers with a set  $C_i \subseteq A$  where  $|C_i| \leq n$ . Duplicator wins the  $r$ -round game if  $\cup_{j \leq r} b_j \upharpoonright C_j$  is a partial isomorphism from  $\mathcal{A}$  to  $\mathcal{B}$ .

Bárány has characterised equivalence of presentations in automata theoretic terms. For the sake of completeness, here is the definition of semi-synchronous rational relation, which can be thought of as a relation recognised by a multi-tape automaton where each read-head advances at a different, but still constant, speed.

**DEFINITION 3.31.** Fix a finite alphabet  $\Sigma$  and a vector of positive integers  $\underline{m} = (m_1, \dots, m_r)$ . Let  $\perp$  be a symbol not in  $\Sigma$ . For each component  $m_i$  introduce the alphabet  $(\Sigma_\perp)^{m_i}$ . The  $\underline{m}$ -convolution of a tuple  $(w_1, \dots, w_r) \in (\Sigma^*)^r$  is formed as follows: First, consider the intermediate string  $(w_1\perp^{a_1}, \dots, w_r\perp^{a_r})$  where the  $a_i$  are minimal such that there is some  $k \in \mathbb{N}$  so that for all  $i$ ,  $|w_i| + a_i = km_i$ . Second, partition each component  $w_i\perp^{a_i}$  into  $k$ -many blocks of size  $m_i$ , and view each block as an element of  $(\Sigma_\perp)^{m_i}$ . Thus the string  $\otimes_{\underline{m}}(w_1, \dots, w_r)$  is formed over alphabet  $(\Sigma_\perp)^{m_1} \times \dots \times (\Sigma_\perp)^{m_r}$ .

The  $\underline{m}$ -convolution of a relation  $R \subseteq (\Sigma^*)^r$  is the set  $\otimes_{\underline{m}}R$  defined as  $\{\otimes_{\underline{m}}\overline{w} \mid \overline{w} \in R\}$ .

Call  $R$   $\underline{m}$ -synchronous rational if there is a finite automaton recognising  $\otimes_{\underline{m}}R$ .

Call  $R$  semi-synchronous if it is  $\underline{m}$ -synchronous rational for some  $\underline{m}$ .

In particular, a  $(1, \dots, 1)$ -synchronous rational relation  $R$  is synchronous rational (Definition 2.2).

**EXAMPLE 3.32.** In the proof of Proposition 2.6, the coding  $\alpha \subseteq \Gamma^* \times \Sigma^*$  is semi-synchronous. Similarly, the translation in Example 3.29 is semi-synchronous.

**THEOREM 3.33.** [2] Two automatic presentations  $\mu$  and  $\nu$  of a structure  $\mathcal{B}$  are equivalent if and only if the relation  $\{(u, v) \mid \mu(u) = \nu(v)\}$  is semi-synchronous.

Here are a few words about the proof. The ‘if’ part follows from the basic property that the composition of a semi-synchronous relation with a regular relation is regular. On the other hand, for the ‘only if’ part it is sufficient to consider injective presentations, and prove that the translation  $\mu^{-1}\nu: \text{dom}(\nu) \rightarrow \text{dom}(\mu)$ , which by assumption preserves the regularity and non-regularity of relations on  $\text{dom}(\nu)$ , is semi-synchronous. This is achieved by a series of transformations, and is nicely presented in [3].

- EXAMPLES 3.34.**
- (i) With a little more work, one can show that each of the structures  $\mathcal{W}_k$  and  $\mathcal{N}_k$  (for  $k \geq 2$ ) has exactly one presentation up to equivalence [2, 3].
  - (ii) Presburger arithmetic  $(\mathbb{N}; +)$  has infinitely many non-equivalent presentations. This follows from a result of Büchi [16] saying that the set of powers of  $n \geq 2$ , written in base  $m \geq 2$  coding, is regular if and only if  $n^p = m^q$  for some positive integers  $p$  and  $q$ .

**3.4. Intrinsic regularity.** This subsection looks at the relationship between regularity and definability in automatically presentable structures. The following definition parallels that of the intrinsically computably enumerable relations; see Ash and Nerode [1]. Loosely, a relation  $R \subseteq B^n$  (not assumed to be an atomic relation of  $\mathcal{B}$ ) is called *intrinsically regular in  $\mathcal{B}$*  if it is regular in every automatic copy of  $\mathcal{B}$ .

**DEFINITION 3.35.** Fix an automatically presentable structure  $\mathcal{B}$ . Call a relation  $R \subseteq B^n$  *intrinsically regular* in  $\mathcal{B}$  if for every automatic presentation  $\mu$  of  $\mathcal{B}$ , the relation  $\mu^{-1}(R)$  is regular (or in the terminology of the previous section, that  $R \in \mu(\text{Reg})$ ).

Denote by  $\text{IR}(\mathcal{B})$  the set of intrinsically regular relations in  $\mathcal{B}$ .

**Problem 3.36.** Can we capture intrinsic regularity using definability? For instance, by Remarks 3.11 and 3.18, for every automatically presentable structure  $\mathcal{B}$  we see that  $\text{FO}[\cup_n Q_n^{\text{reg}}]_{\omega-\text{inv}}(\mathcal{B}) \subseteq \text{IR}(\mathcal{B})$ . Is there equality here?

**IR in some specific structures.** We can describe  $\text{IR}(\mathcal{B})$  for some specific automatically presentable  $\mathcal{B}$ s.

**PROPOSITION 3.37.**

- (i)  $\text{IR}(\mathbb{N}, +, |_m) = \text{FO}(\mathbb{N}, +, |_m)$ , for  $m > 1$ .
- (ii)  $\text{IR}(\mathbb{N}, +) = \text{FO}(\mathbb{N}, +)$ .
- (iii)  $\text{IR}(\mathbb{N}, \leq) = \text{FO}[\exists^{\text{mod}}](\mathbb{N}, \leq)$ .

The first item follows from the fact that (the base  $m$  coding of) a relation is regular only if it is first-order definable in  $(\mathbb{N}, +, |_m)$  (see Theorem 3.8).

The second item follows from the Cobham-Semenov Theorem:  $R$  is first-order definable in  $(\mathbb{N}, +)$  if for some  $m, n \in \mathbb{N}$ ,  $R$  in base  $m$  is regular (as a subset of  $\{0, 1, \dots, m-1\}^*$ ) and  $R$  in base  $n$  is regular, and  $m^p \neq n^q$  for all positive integers  $p, q$  (see [15] for a discussion).

The third item follows from the characterisation of the structures with unary-automatic presentations (Theorem 3.9).

The proof of the next theorem, due to Frank Stephan, is technical and can be found in [51].

**THEOREM 3.38.** *For every  $k \geq 2$ , there is an automatic copy of  $(\mathbb{N}, S)$  in which the image of the set  $\{n \in \mathbb{N} \mid k \text{ divides } n\}$  is not regular.*

Here  $S$  is the usual successor function  $n \mapsto n+1$ . A little work establishes the following corollary.

**COROLLARY 3.39.** *A unary relation  $R \subseteq \mathbb{N}$  is intrinsically regular for the structure  $(\mathbb{N}, S)$  if and only if  $R$  is in  $\text{FO}(\mathbb{N}, S)$ .*

The proof of Theorem 3.38 may be adapted to yield automatic presentations with pathological properties.

A *cut* of the structure  $(\mathbb{Z}, S)$  is a set of the form  $\{x \in \mathbb{Z} \mid x \geq n\}$  where  $n \in \mathbb{Z}$  is fixed.

COROLLARY 3.40. *There is an automatic copy of  $(\mathbb{Z}, S)$  in which no cut is regular.*

COROLLARY 3.41. *There is an automatic copy of a graph with exactly two connected components, each isomorphic to  $(\mathbb{N}, S)$ , and neither regular.*

**3.5. Restriction on growth.** How can one prove that a given structure is not automatically presentable? This section provides some properties of automatically presentable structures, that in the contrapositive can be used to establish that a given structure has no automatic presentation. I first mention two methods based on results that we have already seen.

First, Theorem 3.2 says that the first-order theory of an automatically structure is decidable. This settles, for instance, that arithmetic  $(\mathbb{N}, +, \times)$  is not automatically presentable. A closer approach would be to note that the proof of the theorem gives an upper bound of  $\exp_{k-1}(c^n)$  for the space complexity of deciding the  $\Sigma_k$  (or  $\Pi_k$ ) fragment of the first-order theory of an automatically presentable structure (here  $c$  is the size of the automata in the presentation, and  $n$  the size of the input sentence).

Second, if it has already been established that  $\mathcal{A}$  is not automatically presentable, then no structure that interprets  $\mathcal{A}$  is automatically presentable (Proposition 3.6).

We now turn to some finer techniques. These share the idea that in an automatically presentable structures, certain functions that count (elements, definable sets, etc.) cannot grow too fast.

A *locally-finite* relation  $R$  parameterised by  $k, l \in N$ , is one such that  $R \subseteq A^{k+l}$  and for every tuple  $\bar{x}$  of size  $k$  there are at most a finite number of tuples  $\bar{y}$  of size  $l$  such that  $(\bar{x}, \bar{y}) \in R$ . The canonical example of a locally-finite relation is the graph of a function.

The following proposition says, in a simple case, that if (the graph of) a function  $f$  is regular, then for every  $x$ , the value  $|f(x)| - |x|$  is bounded above by the number of states of an automaton for  $f$ . Although the proposition has an easy proof, it is an important tool.

PROPOSITION 3.42. *Suppose that locally-finite  $R \subseteq A^{k+l}$  is regular. There exists a constant  $p$ , that depends only on the automaton for  $R$ , such*

$$\max\{|y| \mid y \in \bar{y}\} - \max\{|x| \mid x \in \bar{x}\} \leq p$$

for every  $(\bar{x}, \bar{y}) \in R$ .

PROOF. Take the simple case that  $R$  is the graph of a (partial) function  $f: A \rightarrow A$ . Suppose an automaton recognising  $\otimes R$  has  $p$  states. Assume, aiming for a contradiction, that  $|f(x)| > |x| + p$  for some  $x$  in the domain of  $f$ . Then the run of  $\otimes(x, f(x))$ , after reading the first  $|x|$  symbols, must repeat a state. This implies that the automaton also accepts infinitely many strings of the form  $\otimes(x, \cdot)$ .  $\dashv$

We will now see a useful way of iterating Proposition 3.42.

**DEFINITION 3.43.** Start with an automatic copy  $\mathcal{A}$  of a structure that includes functions  $f_1, \dots, f_k$  of arities  $r_1, \dots, r_k$  respectively. Let  $D \subseteq A$  be regular, listed as  $d_i$  so that  $|d_i| \leq |d_{i+1}|$ .

Define the *n<sup>th</sup> growth level*, written  $G_n(D)$ , inductively by  $G_1(D) = \{d_1\}$  and

$$\begin{aligned} G_{n+1}(D) = & \{d_{n+1}\} \cup G_n(D) \cup \\ & \cup_{i \leq k} \{f_i(x_1, \dots, x_{r_i}) \mid x_j \in G_n(D) \text{ for } 1 \leq j \leq r_i\}. \end{aligned}$$

We are interested in how fast  $|G_n(D)|$  grows as a function of  $n$ . For example, consider the free semigroup  $(\Sigma^*, \cdot)$  with generating set  $\Sigma = \{d_1, \dots, d_m\}$ . For  $m \geq 2$ , since  $G_m(\Sigma) \supseteq \Sigma$ , the set  $G_{m+n}(\Sigma)$  includes all strings over  $\Sigma$  of length at most  $2^n$ ; thus the cardinality of  $G_{m+n}(\Sigma)$  is at least a double exponential, namely  $\exp_2(n)$ .

**PROPOSITION 3.44** ([12], cf. [34]). *Let  $\mathcal{A}$  and  $G_n(D)$  be as in Definition 3.43. Then there is a linear function  $t: \mathbb{N} \rightarrow \mathbb{N}$  so that every string of  $G_n(D)$  has length at most  $t(n)$ .*

In other words, if  $|\Sigma| = 1$  then  $|G_n(D)| = O(n)$ , else  $|G_n(D)| = |\Sigma|^{O(n)}$ .

**COROLLARY 3.45.** *The following structures do not have automatic presentations.*

- (i) *The free semigroup  $(\Sigma^*, \cdot)$  on  $|\Sigma| > 1$  generators [34].*
- (ii) *Any term algebra generated by finitely many constants and at least one non-unary atomic function [34].*
- (iii)  *$(\mathbb{N}, \times)$ , multiplication of natural numbers [12].*
- (iv)  *$(\mathbb{N}, \langle \cdot, \cdot \rangle)$ , where  $\langle \cdot, \cdot \rangle: \mathbb{N}^2 \rightarrow \mathbb{N}$  is a bijection [12].*

A straightforward application of Proposition 3.42 is the following result. It can be proved by induction on  $m$ .

**PROPOSITION 3.46.** *Let  $(M; \cdot)$  be an automatic copy of a semigroup. Then there is a constant  $e$ , that depends only on an automaton for  $\cdot$ , so that for every  $x_1, \dots, x_m \in M$ ,*

$$|\prod_{i=1}^m x_i| \leq \max\{|x_i| : 1 \leq i \leq m\} + e \log m.$$

This proposition will be used in Section 4 in the classification of the automatically presentable Boolean algebras and for proving certain groups do not have automatic presentations.

A different technique, independently reported by Delhommé [23] and Stephan [52], is used to prove that certain universal homogeneous structures have no automatic presentation. The basic ideas are presented below, following [35], although it is worth mentioning that Delhommé [24] presents a more general result, as well stating a corresponding condition for structures with tree-automatic presentations.

Recall  $A^{\leq n}$  denotes the strings of  $A$  of length at most  $n$ .

**DEFINITION 3.47.** Suppose that the structure  $\mathcal{A}$  contains an atomic binary relation  $E$  and that  $A \subseteq \Sigma^*$ . For  $n \in \mathbb{N}$  and  $y \in A$  define the set  $\text{tp}_n(y) \subseteq A^{\leq n}$  as those  $x \in A^{\leq n}$  for which  $\mathcal{A} \models E(x, y)$ .

Write  $\#\text{tp}_n$  for the cardinality of the set  $\{\text{tp}_n(y) \mid y \in A\}$ . Note that the set  $\text{tp}_n$  and the number  $\#\text{tp}_n$  are defined with respect to  $A$  and  $E$ . I may write  $\text{tp}_{n,A,E}$  to stress this.

In general  $\#\text{tp}_n \leq |\Sigma|^{|A^{\leq n}|}$ , however if  $\mathcal{A}$  is an automatic copy of some structure then we can say more.

**THEOREM 3.48.** *If  $\mathcal{A}$  is an automatic copy of a structure containing an atomic binary relation  $E$ , then  $\#\text{tp}_{n,A,E} \leq k|A^{\leq n}|$  for some constant  $k \in \mathbb{N}$  that depends on the automata for  $A$  and  $E$ .*

**PROOF.** One proves that there is a constant  $c$ , depending on the number of states of the automata for  $A$  and  $E$ , so that for every  $n \in \mathbb{N}$  and  $y \in A$ , there is a  $y' \in A^{\leq n+c}$  with  $\text{tp}_n(y) = \text{tp}_n(y')$ . Now apply the fact that  $|A^{\leq n+c}| \leq k(|A^{\leq n}|)$  where the constant  $k \in \mathbb{N}$  depends on the number of states of the automaton for  $A$ .  $\dashv$

**COROLLARY 3.49.** *The following structures do not have automatic presentations.*

- (i) *The random graph.*
- (ii) *The universal, homogeneous partial order.*
- (iii) *The  $K_p$ -free random graph for every  $p > 2$  ( $K_p$  is the complete graph on  $p$  vertices).*

**PROOF.** The first case is illustrated. Suppose  $(A; E)$  is an automatic copy of the random graph over a binary alphabet. The random graph has the following property. For every finite subset  $F$  of  $A$ , and every partition  $X_1, X_2$  of  $F$ , there is a vertex  $x \in A$  such that  $(x, x_1) \in E$  for every  $x_1 \in X_1$  and  $(x, x_2) \notin E$  for every  $x_2 \in X_2$ . Hence taking  $F$  to be  $A^{\leq n}$  it holds that  $\#\text{tp}_{n,A,E} = \exp(|A^{\leq n}|)$  contradicting the theorem.  $\dashv$

We finish with another necessary condition of having an automatic presentation, due to Delhomme [24].

**DEFINITION 3.50.** Say that a structure  $\mathcal{B}$  is a *sum-augmentation* of a set of structures  $\mathcal{S}$  (each having the same signature as  $\mathcal{B}$ ) if there is a finite partition of  $B = B_1 \cup \dots \cup B_n$  such that for each  $i$  the substructure  $\mathcal{B} \upharpoonright B_i$  is isomorphic to some structure in  $\mathcal{S}$ .

**THEOREM 3.51.** *Suppose  $\mathcal{A}$  has finite signature. If  $\mathcal{A}$  is an automatically presentable then for every  $\mathcal{A}$ -formula  $\phi(x, \bar{y})$ , there is a finite set of structures  $\mathcal{S}$  so that for every tuple of elements  $\bar{b}$  from  $A$ , the substructure  $\mathcal{A} \upharpoonright \phi^{\mathcal{A}}(\cdot, \bar{b})$  is a sum-augmentation of  $\mathcal{S}$ .*

**PROOF.** Say  $\mathcal{A} = (A; R_1^{\mathcal{A}}, \dots, R_r^{\mathcal{A}})$ ,  $A \subseteq \Sigma^*$ , is an automatic copy of a structure. For any given  $\mathcal{A}$ -formula  $\psi$ , fix a deterministic automaton

$(Q_\psi, \iota_\psi, \Delta_\psi, F_\psi)$  recognising  $\psi^A$ , and write  $\Gamma_\psi(w)$  for  $\Delta_\psi(\iota_\psi, w)$ . We will use the following property  $(P_\psi)$ : For all strings  $c_i, d_i$  with the  $c_i$ s all the same length,

$$\Delta_\psi(\Gamma_\psi(\otimes(c_1, \dots, c_k)), \otimes(d_1, \dots, d_k)) \in F_\psi.$$

if and only if  $\psi(c_1d_1, \dots, c_kd_k)$  holds in  $A$ .

Now, given an  $A$ -formula  $\phi(x, y_1, \dots, y_l)$  as in the hypothesis, and tuple  $\bar{b}$ , observe that for  $m = \max\{|b_i|\}$ , we can partition  $\phi^A(\cdot, \bar{b})$  into the finitely many singletons  $\{c\}$  for  $\phi(c, \bar{b})$  with  $|c| < m$ , and the finitely many sets  $\phi^{a\Sigma^*}(\cdot, \bar{b}) := \{aw \in A \mid A \models \phi(aw, \bar{b}), w \in \Sigma^*\}$  for  $|a| = m$ . Since the signature is assumed to be finite, there are finitely many isomorphism types amongst substructures of the form  $A \upharpoonright \{a\}$ , for  $a \in A$ . So, it is sufficient to show, that as we vary  $(a, \bar{b})$  subject to  $|a| = \max\{|b_i|\}$ , there are finitely many isomorphism types amongst substructures of the form  $A \upharpoonright \phi^{a\Sigma^*}(\cdot, \bar{b})$ .

The idea is to bound this number of isomorphism types in terms of the number of states of the automata involved. To this end, define a function  $f_\phi$  as follows. Its domain consists of tuples  $(a, \bar{b})$  where  $|a| = \max\{|b_i|\}$ ; and  $f_\phi$  sends this tuple to the tuple of states

$$(\Gamma_\phi(\otimes(a, b_1, \dots, b_l)), \Gamma_A(a), (\Gamma_{R_i^A}(\otimes(a, \dots, a)))_{i \leq r}).$$

The range of  $f_\phi$  is bounded by  $|Q_\phi| \times |Q_A| \times \prod_{i \leq r} |Q_{R_i^A}|$ . In particular, the range is finite.

To finish the proof, one needs to argue that the isomorphism type of the substructure  $A \upharpoonright \phi^{a\Sigma^*}(\cdot, \bar{b})$  depends only on the value  $f_\phi(a, \bar{b})$ . This follows from the fact that if  $f_\phi(a, \bar{b}) = f_\phi(a', \bar{b}')$ , then the corresponding substructures are isomorphic via the mapping  $I: aw \mapsto a'w$  ( $w \in \Sigma^*$ ). Indeed, by property  $(P_{x \in A})$  we get that  $aw \in A$  if and only if  $a'w \in A$ , for every  $w$ . This means that  $I$  is a bijection between the sets  $A \cap a\Sigma^*$  and  $A \cap a'\Sigma^*$ . Similarly, by the properties  $(P_\psi)$  where  $\psi$  is taken to be  $\bar{x} \in R_i^A$ , we get that  $I$  is an isomorphism between the substructures  $A \upharpoonright a\Sigma^*$  and  $A \upharpoonright a'\Sigma^*$ . Finally, by  $(P_\phi)$  we get that  $I$  is an isomorphism between the substructures  $A \upharpoonright \phi^{a\Sigma^*}(\cdot, \bar{b})$  and  $A \upharpoonright \phi^{a'\Sigma^*}(\cdot, \bar{b})$ .  $\dashv$

Here is an illustration of the theorem.

**COROLLARY 3.52.** [22] *The ordinal  $(\omega^\omega, \leq)$  is not automatically presentable.*

**PROOF.** Suppose for a contradiction that  $(\omega^\omega, \leq)$  has an automatic presentation. In Theorem 3.51, take  $\phi(x, y)$  to be  $x < y$  and consider the following fact (proved by induction): If the domain of a well-order, isomorphic to some ordinal of the form  $\omega^n$  for  $n \in \mathbb{N}$ , is partitioned into finitely many pieces  $\{B_i\}_i$ , then there is some  $i$  so that the substructure on domain  $B_i$  is isomorphic to  $\omega^n$ .

This means that the set of structures  $\mathcal{S}$  must contain  $(\omega^n, <)$  for every  $n \in \mathbb{N}$ , contradicting the finiteness of  $\mathcal{S}$ .  $\dashv$

**3.6. Isomorphism problem.** Let  $C$  be a class of structures (over a finite signature) closed under isomorphism. Write  $C^{\text{aut}}$  for the (codes of the) tuples of automata  $\bar{M} = (M_A, M_-, (M_i)_i)$  in automatic presentations of members of  $C$ .

The *isomorphism problem for the automatically presentable members of  $C$*  is the set

$$\{\langle \bar{M}, \bar{M}' \rangle \mid \bar{M}, \bar{M}' \in C^{\text{aut}} \text{ present isomorphic structures}\}.$$

Note that the  $C^{\text{aut}}$  may be computable (for instance, if  $C$  are the Boolean algebras, linear orderings, or finite graphs). However this does not mean that we can describe the isomorphism types of the automatically presentable members of  $C^{\text{aut}}$ . Indeed, the isomorphism problem for the automatically presentable directed graphs is undecidable [13]:

**PROOF.** We encode the halting problem into this isomorphism problems as follows. For a given a Turing machine  $N$ , construct an equivalent reversible Turing machine  $N_r$ . In particular, this means that its configuration space is a disjoint union of *chains* (these are graphs isomorphic to initial segments of  $(\mathbb{N}, S)$  where  $S: n \mapsto n + 1$ ). Such a machine can be constructed by introducing to  $N$  a tape that records the sequence of transitions that it takes, see Bennett [7].

We may assume that  $N$  (and hence also  $N_r$ ) loops indefinitely instead of halting in a reject state. So, the configuration space of  $N_r$  consists of a finite chain for every accepting computation of  $N_r$ , an infinite chain for every rejecting computation of  $N_r$ , and possibly some other chains that do not correspond to valid computations of  $N_r$  (the *junk*).

The configuration space of a Turing machine  $N_r$  is automatically presentable (Example 2.4 (4)). It can be massaged so that its only finite chains are those that correspond to *valid* accepting computations of  $N_r$ . Consider the set of configurations  $I$  of  $N_r$  which have no predecessor in  $N_r$ , but are not valid initial configurations (in other words, consider the roots of chains from the junk). Write  $(\mathbb{N}, P)$  for the graph with domain  $\mathbb{N}$  and edge relation  $(n+1, n)$  for every  $n \in \mathbb{N}$ . For each element  $i \in I$ , attach the chain with root  $i$  to a copy of  $(\mathbb{N}, P)$  at 0. Here *attach  $H$  to  $G$*  means take the disjoint union of  $H$  and  $G$  and add an edge from the distinguished node of  $g$  to that of  $h$ . Call the resulting graph  $N'$ . It has the property that every invalid computation is isomorphic to either  $(\mathbb{N}, P)$  (if it were a terminating computation) or to  $(\mathbb{Z}, S)$  (if it were non-terminating).

Denote by  $\mathcal{J}$  an automatic copy of the graph consisting of infinitely many disjoint copies of each of the following structures:  $(\mathbb{N}, S)$ ,  $(\mathbb{N}, P)$  and  $(\mathbb{Z}, S)$ . Denote by  $N''$  the disjoint union of the graph  $N'$  and  $\mathcal{J}$ . Then  $N$  rejects every input string if and only if  $N''$  is isomorphic to  $\mathcal{J}$ .

It is straightforward to check that  $N''$  is automatically presentable, and that automata presenting it can be computed from  $N$ . ⊣

In the previous proof the configuration space of  $N_r$  is *locally-finite*, namely the degree of every vertex in the underlying undirected graph is finite. The isomorphism problem for the locally-finite automatically presentable directed graphs is  $\Pi_3^0$ -complete [51]. So, at first sight, the following result seems surprising.

**THEOREM 3.53.** [35] *The isomorphism problem for the class of automatically presentable directed graphs is  $\Sigma_1^1$ -complete.*

**PROOF.** For hardness, we encode the isomorphism problem for computable subtrees of  $\omega^{<\omega}$ . A proof of its  $\Sigma_1^1$ -completeness can be found in Goncharov and Knight [28, Theorem 4.4(b)] who attribute it to the folklore. Let us briefly describe this problem.

Here  $\omega^{<\omega}$  is the set of all finite sequences from the set of natural numbers  $\omega$ . Implicitly, there is an edge from  $x$  to  $x \cdot n$  where  $x \in \omega^{<\omega}$  and  $n \in \omega$ ; this is called the *immediate successor* relation. A *subtree* of  $\omega^{<\omega}$  is a subset  $T$  that is downward closed with respect to the immediate successor relation: for  $x \in \omega^{<\omega}$  and  $n \in \omega$ , if  $x \cdot n \in T$  then  $x \in T$ . A subtree  $T$  is *computable* if there is an algorithm that on input  $x \in \omega^{<\omega}$  decides whether or not  $x$  is in  $T$ . Two subtrees are isomorphic if there is a bijection respecting the immediate successor relation. The isomorphism problem for computable subtrees of  $\omega^{<\omega}$  is the set of pairs  $\langle n, m \rangle$  for which  $n$  and  $m$  are indices of computable subtrees, say  $T_n$  and  $T_m$  respectively, and  $T_n$  is isomorphic to  $T_m$ .

Our aim is to exhibit, for each computable subtree  $T$ , an automatically presentable directed graph  $\mathcal{G}_T$ , so that computable subtrees  $T_1$  and  $T_2$  are isomorphic if and only if the directed graphs  $\mathcal{G}_{T_1}$  and  $\mathcal{G}_{T_2}$  are isomorphic. Moreover, given an index for  $T$ , we effectively construct automata presenting  $\mathcal{G}_T$ .

To begin,  $\omega^{<\omega}$  has an automatic copy  $\mathcal{W}$ : the domain  $W$  is  $\{\lambda\} \cup \{0, 1\}^*1$  and immediate successor is given by  $x \prec_p y \wedge \neg \exists z x \prec_p z \prec_p y$ . So replacing  $\omega^{<\omega}$  with  $\mathcal{W}$ , we talk instead about (*computable*) *subtrees* of  $\mathcal{W}$ . Now, from a computable subtree  $T$  of  $\mathcal{W}$ , construct a reversible Turing machine  $N_T$  with the property that  $w \in T$  if and only if the computation of  $N_T$  starting on input  $w$  converges (compare with the previous proof). Writing  $\mathcal{G}'_T$  for the configuration space of  $N_T$ , note that  $\mathcal{G}'_T$  consists of disjoint unions of chains. Abuse notation and identify the initial configuration of  $\mathcal{G}'_T$  on input  $w \in W$  with the string  $w$ .

We need an auxilliary graph  $\mathcal{O}$  formed from a copy of  $\mathcal{W}$  by attaching to every  $w \in W$ , infinitely many finite chains of every size and exactly one infinite chain.

Form  $\mathcal{G}''_T$  from  $\mathcal{G}'_T$  by attaching infinitely many copies of  $\mathcal{O}$  to every  $w \in W \subseteq \mathcal{G}'_T$ .

Finally, form the graph  $\mathcal{G}_T$  by taking the disjoint union of  $\mathcal{G}''_T$  and infinitely many chains of every size (finite and infinite). This last step takes care of the non-valid computations of  $N_T$ . It is straightforward to see that  $\mathcal{G}_T$  is automatically presentable.

For every  $w \in W \subseteq G_T$ ,

- (i)  $w \in T$  if and only if there is not an isolated infinite chain in  $\mathcal{G}_T$  whose root is an immediate successor of  $w$  (*isolated* means that every element of this chain has at most one immediate successor in  $\mathcal{G}_T$ );
- (ii) if  $w \notin T$  then the tree in  $\mathcal{G}_T$  rooted at  $w$  is isomorphic to  $\mathcal{O}$  (this is because  $T$  is downward closed).

The first property ensures that every isomorphism  $f : \mathcal{G}_{T_1} \cong \mathcal{G}_{T_2}$  restricts to an isomorphism between  $T_1$  and  $T_2$ , since  $f$  maps an isolated chain of size  $\kappa \leq \omega$  to an isolated chain of the same size.

Conversely, in order to extend an isomorphism  $g : T_1 \cong T_2$  to  $\mathcal{G}_{T_1} \cong \mathcal{G}_{T_2}$ , note that every  $w \in W \subseteq G_T$  has infinitely many immediate successors  $v \in W$  that are the roots in  $\mathcal{G}_T$  of trees isomorphic to  $\mathcal{O}$ . This mitigates against the case that the cardinality of  $S(w) \setminus T_1$  is different from the cardinality of  $S(g(w)) \setminus T_2$ , where  $S(\cdot) \subseteq W$  is the function denoting the set of immediate successors in the tree  $\mathcal{W}$ .  $\dashv$

This theorem can be extended to other classes of automatically presentable structures.

If  $(T; \preceq)$  is a tree, then call  $(T; S_\preceq)$  a *successor tree*, where  $S_\preceq(x)$  is defined as the set of immediate  $\preceq$ -successors of  $x \in T$ . The proof of Theorem 3.53 can easily be adapted to show that the isomorphism problem for the class of automatically presentable successor trees is  $\Sigma_1^1$ -complete. We can get the result for undirected graphs by attaching a gadget (such as a cycle of length 3) to identify the root of the successor tree, and then considering the underlying graph.

Nies [46] observes that since the class of undirected graphs is bi-interpretable in a variety of other classes, one gets  $\Sigma_1^1$ -completeness for these classes as well.

**COROLLARY 3.54.** *The isomorphism problem for each of the following classes is  $\Sigma_1^1$ -complete.*

- (i) *The automatically presentable successor trees.*
- (ii) *The automatically presentable undirected graphs.*
- (iii) *The automatically presentable commutative monoids.*
- (iv) *The automatically presentable partial orders.*
- (v) *The automatically presentable lattices of height 4.*
- (vi) *The automatically presentable algebras consisting of two 1-ary functions.*

**§4. Classifications.** Fix a class  $C$  of structures (say the class of Boolean algebras, or linear orders). Which members of  $C$  are automatically presentable? This section is concerned with describing, in terms of classical invariants, the isomorphism types of the automatic members of  $C$ .

Each subsection begins with the classification of the unary-autonomically presentable structures in the given class. These are usually based on the

following proposition, whose proof follows from an analysis of the structure of the corresponding automata.

**PROPOSITION 4.1.** [9, 51] *If a graph  $\mathcal{G} = (G; E)$  is a unary-automatically presentable, then it contains a finite number of infinite connected components, and a finite bound on the sizes of the finite connected components.*

For the non-unary case, Proposition 2.6 says that it is sufficient to consider  $|\Sigma| = 2$ .

**Remark 4.2.** Each of the following subsections indicate that the classification of classes of unary-automatically presentable structures is considerably simpler than the classification of classes of automatically presentable structures over a binary alphabet.

**4.1. Equivalence structures.** An *equivalence structure*  $(E; \rho)$  is one where  $\rho$  is an equivalence relation on the set  $E$ . Each equivalence structure is characterised up to isomorphism by the number of equivalence classes of every size. To this end, define the *height function*  $h_{\mathcal{E}} : \mathbb{N} \cup \{\infty\} \rightarrow \mathbb{N} \cup \{\infty\}$  of an equivalence structure  $\mathcal{E}$  as  $h(n)$  being the number (possibly infinite) of equivalence classes of size  $n$  (possibly infinite). Then  $\mathcal{E}_1$  is isomorphic to  $\mathcal{E}_2$  if and only if  $h_{\mathcal{E}_1} = h_{\mathcal{E}_2}$ .

The unary case follows immediately from Proposition 4.1.

**THEOREM 4.3.** [9, 51] *An equivalence structure  $(E; \rho)$  has a unary-automatic presentation if and only if  $h_{\mathcal{E}}(\infty)$  is finite and  $h_{\mathcal{E}}(n) = 0$  for all but finitely many  $n \in \mathbb{N}$ .*

We now turn to the non-unary case. Consider an automatically presentable equivalence structure  $\mathcal{E}$ . The set of elements of  $E$  in infinite classes is definable using  $\exists^\infty$ , and so we can compute how many infinite classes there are. So to characterise the automatically presentable equivalence structures it is sufficient to consider those with no infinite classes. To this end, write  $\mathcal{H}$  for the set of height functions of automatically presentable equivalence structures with no infinite classes. Adopt the convention that  $m + n = m \times n = \infty$  if at least one of  $m$  or  $n$  is  $\infty$ .

**PROPOSITION 4.4.** [51]

- (i)  $\mathcal{H}$  contains all functions of the form  $h_f : \mathbb{N} \rightarrow \mathbb{N}$  defined by  $h(f(n)) = 1$ , where  $f : \mathbb{N} \rightarrow \mathbb{N}$  is either a polynomial (with coefficients in  $\mathbb{N}$ ) or an exponential  $k^{an+b}$  (for some  $k, a, b \in \mathbb{N}$ ).
- (ii) If  $h, h' \in \mathcal{H}$  then so is their
  - (a) sum  $(h + h')(n) = h(n) + h'(n)$ ,
  - (b) Dirichlet convolution  $(h \star h')(n) = \sum_{ab=n} h(a)h'(b)$ , and
  - (c) Cauchy product  $(h \# h')(n) = \sum_{a+b=n} h(a)h'(b)$ .

**PROOF.** For the first item, consider automatic copy  $(R; \text{el})$  of an equivalence structure where  $R \subseteq \Sigma^*$  and  $\text{el}$  is the equal length predicate. Thus the equivalence classes are of the form  $R^{=n}$  for  $n \in \mathbb{N}$ . But for every polynomial

or exponential function  $f$  as in the statement of the proposition, there is a regular set  $R_f \subseteq \Sigma^*$  with  $f(n) = |(R_f)^{=n}|$  (see [53] or [51, Lemma D.2.3]).

For the second item, sum corresponds to disjoint union of equivalence structures, and Dirichlet convolution corresponds to direct product. Cauchy product is only slightly more involved.  $\dashv$

So in particular there is an automatically presentable equivalence structure whose height function has unbounded range. As far as I know, neither is there a classification of  $\mathcal{H}$ , nor is it known whether the isomorphism problem for automatically presentable equivalence relations, seen to be  $\Pi_1^0$ , is decidable.

**4.2. Linear orders.** An excellent reference for linear orders is [50]. The classical ranking of linear orders  $\mathcal{L} = (L; \leq)$  is based on iteratively factoring  $\mathcal{L}$  by the equivalence relation  $c$  stating that  $x$  is equivalent to  $y$  if the number of elements between  $x$  and  $y$  is finite.

For ease of reference, here is the formal definition. For every ordinal  $\alpha$  define an equivalence relation  $\sim_\alpha$  on  $\mathcal{L}$  inductively:  $x \sim_\alpha y$  if  $x = y$  or for some  $\beta < \alpha$ , the number of elements in  $\mathcal{L}/\sim_\beta$  between  $[x]_{\sim_\beta}$  and  $[y]_{\sim_\beta}$  is finite. Here  $\mathcal{L}/\sim_\beta$  is the linear ordering defined as follows: its domain is the collection of non-empty  $\sim_\beta$ -equivalence classes  $[x]_{\sim_\beta}$  ( $x \in L$ ). The order is defined by  $[x]_{\sim_\beta}$  less than  $[y]_{\sim_\beta}$  if  $a <_L b$  for every  $a \in [x]_{\sim_\beta}, b \in [y]_{\sim_\beta}$ .

The FC-rank of  $\mathcal{L}$  is defined as the least ordinal  $\alpha$  so that for every  $\beta < \alpha$  and every  $x \in L$ , we have  $[x]_{\sim_\beta} = [x]_{\sim_\alpha}$ . For example, the FC-rank of the ordinal  $\omega^\alpha$  is  $\alpha$ .

Here FC is an acronym for ‘finite condensation’, hinting that elements a finite distance apart are condensed together.

A linear order  $\mathcal{L}$  is *dense* if for every distinct  $a, b \in L$ , there is a  $z \in L$  with  $a < z < b$ . Note that the linear order with exactly one element is dense. A linear order is *scattered* if none of its suborderings are both dense and infinite. The *ordered sum* of the orderings  $\mathcal{A}_b$  indexed by the linear order  $B$  is the result of replacing each  $b \in B$  by a copy of  $\mathcal{A}_b$ , and is written  $\Sigma_B \mathcal{A}_b$ .

*Fact 4.5.* Every linear order  $\mathcal{L}$  is a dense sum of scattered linear orders. That is,  $\mathcal{L}$  can be expressed as  $\Sigma_D \mathcal{L}_d$  for some dense  $D$  and scattered  $\mathcal{L}_d$ s,  $d \in D$ .

I do not know of a non-machine theoretic classification of the automatically presentable linear orders, or even the FC-rank 1 linear orders (it is straightforward to exhibit a linear order of FC-rank 1 with undecidable first-order theory). The next result describes the unary case.

**THEOREM 4.6.** [9, 51] *A linear order  $(L; \leq)$  has a unary-automatic presentation if and only if it is a finite sum of scattered orders of FC-rank at most 1.*

In other words, the linear orders with unary-automatic presentations are finite sums of linear orders amongst the set  $\omega$ ,  $\omega^*$ , and  $\mathbf{n}$ , for  $n < \omega$ . In

particular, the order type of the rationals is not automatically presentable over a unary alphabet, and the least ordinal without a unary automatic presentation is  $\omega^2$ .

Regarding the general (non-unary) case, Khoussainov and Nerode asked for the least ordinal without an automatic presentation [34] (it is easy to see that the automatically presentable ordinals form an initial segment of the countable ordinals). In Corollary 3.52, we saw that the condition of Theorem 3.51 implies that the answer is  $\omega^\omega$ .

Similarly, we can use the condition of Theorem 3.51 to get a generalisation to linear orders [37].

**THEOREM 4.7.** *The FC-rank of every automatically presentable linear order is finite.*

**PROOF.** In Theorem 3.51, let  $\mathcal{L}$  be an automatically presentable linear order, and  $\phi(x, y_1, y_2)$  be the relation  $y_1 \leq x \leq y_2$ . Then there is a finite set of structures  $\mathcal{S}$ , so that for every  $a_1 \leq a_2 \in L$ , there is a partition of the domain of  $[a_1, a_2]$  into finitely many pieces  $\{A_i\}$ , so that every  $\mathcal{L} \upharpoonright A_i$  is isomorphic to some structure in  $\mathcal{S}$ .

Just as in Corollary 3.52, we look for a decomposition property such as: If the domain of an automatically presentable linear order  $\mathcal{L}$  is partitioned into finitely many pieces  $\{B_i\}_i$ , then some  $\mathcal{L} \upharpoonright B_i$  has the same FC-rank as  $\mathcal{L}$ . Of course this condition fails horribly if  $\mathcal{L}$  is not scattered, because  $\mathcal{L}$  then embeds orders of arbitrary countable FC-rank. However, in the scattered case it just falls short; for instance, partition  $\mathcal{L} = \omega + \omega$  into the first copy of  $\omega$  ( $B_1 = \{0, 1, \dots\}$ ) and the second copy ( $B_2 = \{\omega, \omega + 1, \dots\}$ ). Then  $\text{FC}(\mathcal{L}) = 2$ , but  $\text{FC}(B_i) = 1$ .

By slightly altering the notion of rank we can achieve the required decomposition result for scattered linear orders. Define the  $\text{FC}_*$ -rank of a scattered linear order  $\mathcal{L}$  as the least ordinal  $\alpha$  such that  $\mathcal{L}$  can be expressed as a finite sum of orders of FC-rank at most  $\alpha$ . Then  $\text{FC}_*(\mathcal{L}) \leq \text{FC}(\mathcal{L}) \leq \text{FC}_*(\mathcal{L}) + 1$ .

The following decomposition property can be proved by induction on  $\text{FC}_*$ -rank: If the domain of a scattered linear order  $\mathcal{L}$  is partitioned into finitely many pieces  $\{B_i\}_i$ , then there is some  $i$  with  $\text{FC}_*(\mathcal{L} \upharpoonright B_i) = \text{FC}_*(\mathcal{L})$ .

Combining this with the statement in the first paragraph, we see that for every  $a_1 \leq a_2 \in L$ , if  $\mathcal{L} \upharpoonright [a_1, a_2]$  is scattered, say with  $\text{FC}_*$ -rank  $\alpha$ , then  $\mathcal{S}$  contains a scattered linear order of  $\text{FC}_*$ -rank  $\alpha$ . Moreover,  $\alpha$  is finite; for otherwise, using elementary properties of rank,  $\mathcal{L} \upharpoonright [a_1, a_2]$  would contain infinitely many closed scattered subintervals having pairwise distinct  $\text{FC}_*$ -ranks, contradicting the finiteness of  $\mathcal{S}$ . Thus there is a uniform finite bound, say  $k$ , on the FC-rank of every closed scattered subinterval of  $\mathcal{L}$ .

Now if  $\mathcal{L}$  is scattered, then every two elements  $a_1, a_2 \in L$  are condensed within  $k$  steps; thus  $\text{FC}(\mathcal{L}) \leq k$ . Incase  $\mathcal{L}$  is not scattered, by Fact 4.5,  $\mathcal{L}$  is the sum of scattered orders  $\mathcal{L}_d$  ( $d \in \mathcal{D}$ ,  $\mathcal{D}$  dense). Consequently, each  $\mathcal{L}_d$  has FC-rank at most  $k$ , and so  $\mathcal{L}$  has FC-rank at most  $k$ .  $\dashv$

Some decidability results for automatically presentable linear orders now follow. Since the axioms stating that  $\leq$  linearly orders  $L$  are first-order, it is decidable whether a given automatically presentable structure  $(L; \leq)$  is a linear order or not.

**COROLLARY 4.8.** [37] *Let  $\mathcal{L} = (L; \leq)$  be an automatically presentable linear order.*

- (i) *It is decidable whether or not  $\mathcal{L}$  is scattered. In case  $\mathcal{L}$  is not scattered, we can effectively compute an automatically presentable dense subordering  $\mathcal{D}$  and automatically presentable scattered suborderings  $\mathcal{L}_d$  for which  $\mathcal{L} = \Sigma_{\mathcal{D}} \mathcal{L}_d$ .*
- (ii) *It is decidable whether or not  $\mathcal{L}$  is isomorphic to an ordinal.*
- (iii) *The isomorphism problem for automatically presentable ordinals is decidable. In fact the Cantor-normal-form may be extracted from an automatic copy of an ordinal.*

It is not known whether the isomorphism problem for automatically presentable linear orderings is decidable.

*Cantor's theorems.* One of Cantor's theorems says that every countable linear ordering embeds in the rational ordering  $\mathbb{Q}$ .

There are, potentially, a variety of possible automatic versions. The following proposition is the best known.

**PROPOSITION 4.9.** [38] *Every automatic copy  $\mathcal{M}$  of a linear order can be embedded into some automatic copy of  $\mathbb{Q}$  by a function  $f: \mathcal{M} \rightarrow \mathbb{Q}$  with the following properties:*

- (i) *The function  $f$  is continuous with respect to the order topology.*
- (ii) *The graph of  $f$  is regular.*

**PROOF.** We mainly present the definition of the required automatic copy of  $\mathbb{Q}$ . Let  $\mathcal{M} = (M; \leq_M)$  be an automatic copy of a linear order, where  $M \subseteq \Delta^*$  and  $0, 1 \notin \Delta$ . Define  $M_L \subseteq M$  as the set of elements of  $\mathcal{M}$  that cannot be approximated from the left. That is,  $w \in M_L$  if and only if

$$(\exists u)[u <_M w \wedge \neg(\exists z)(u <_M z <_M w)] \vee \neg(\exists u)[u <_M w].$$

Similarly define  $M_R \subseteq M$  as the set of elements that cannot be approximated from the right. The required automatic copy of  $\mathbb{Q}$  has domain

$$M \cup M_L \cdot 0\{0, 1\}^* \cup M_R \cdot 1\{0, 1\}^*$$

and relation  $uw \leq u'w'$  (here  $u, u' \in M$  and  $w, w' \in \{0, 1\}^*$ ) if  $u \leq_M u'$  or otherwise  $u = u'$  and  $w \sqsubseteq_Q w'$ . Here  $(\{0, 1\}^*, \sqsubseteq_Q)$  is the automatic copy of  $\mathbb{Q}$  from Example 2.4 (5). The required function  $f$  maps  $u \mapsto u$  for  $u \in M$ .  $\dashv$

It is not known whether there is a single automatic copy of  $\mathbb{Q}$  that embeds, in the sense above, all automatic copies of all automatically presentable linear orders  $\mathcal{M}$ .

Cantor also proved that  $\mathbb{Q}$  is homogeneous: For every two tuples  $x_1 < \dots < x_m$  and  $y_1 < \dots < y_m$  there is an automorphism  $f : \mathbb{Q} \rightarrow \mathbb{Q}$  with  $f(x_i) = y_i$  for  $i \leq m$ . Again there might be a number of automatic variations. Call an automatic copy of  $\mathbb{Q}$  *automatically homogeneous* if for every two tuples there is an automorphism as above that is also regular.

**PROPOSITION 4.10.** [38] *The automatic copy of  $\mathbb{Q}$  in Example 2.4(5) is automatically homogeneous. There exists an automatic copy of  $\mathbb{Q}$  that is not automatically homogeneous.*

**4.3. Trees.** A tree  $(T; \preceq)$  is a partial order with a smallest element and the property that for every element  $v \in T$ , the set  $\{w \in T \mid w \preceq v\}$  is finite and linearly ordered by  $\preceq$ .

For  $u \in T$ , write  $S(u)$  for the set of  $\prec$ -immediate successors of  $u$ ; namely,

$$S(u) = \{v \in T \mid u \prec v \wedge \forall z[u \prec z \preceq v \rightarrow z = v]\}.$$

A tree is *finitely-branching* if  $S(u)$  is finite for every  $u \in T$ . An *infinite path* in a tree is a sequence of elements  $(w_i)_{i \in \mathbb{N}}$  with  $w_{i+1} \in S(w_i)$  for every  $i$ .

The complexity of the isomorphism problem for automatically presentable trees, over the signature of partial orders, is not known. However, another measure of the complexity of a tree is its Cantor–Bendixson rank. Before giving the definition, we need some preliminary concepts. Define the extendible part  $E(\mathcal{T})$  of  $\mathcal{T}$  as those  $x \in T$  that are on some infinite path of  $T$ . Define  $d(\mathcal{T})$ , the *derivative* of  $\mathcal{T}$ , as the subtree of  $\mathcal{T}$  restricted to those elements that are on two distinct infinite paths, namely:

$$\{x \in T \mid \exists z, z' \in E(\mathcal{T}), z, z' \succ x \text{ and neither } z \preceq z' \text{ nor } z' \preceq z\}.$$

For each ordinal  $\alpha$  define the iterated operation  $d^\alpha(\mathcal{T})$  inductively by

$$d^\alpha(\mathcal{T}) = \{x \in T : \forall \beta < \alpha [x \text{ is an element of the tree } d(d^\beta(\mathcal{T}))]\}.$$

Note that for  $\alpha = 0$  the range of the universal quantifier is void, and therefore  $d^0(\mathcal{T})$  is just  $T$ .

**DEFINITION 4.11.** The *Cantor–Bendixson rank*  $\text{CB}(\mathcal{T})$  of a tree  $\mathcal{T}$  is the least ordinal  $\alpha$  such that  $d^\alpha(\mathcal{T}) = d^{\alpha+1}(\mathcal{T})$ .

Since  $\mathcal{T}$  is countable,  $\text{CB}(\mathcal{T})$  is a countable ordinal. Also, if  $\mathcal{T}$  contains countably many infinite paths, then  $d^{\text{CB}(\mathcal{T})}$  is the empty tree.

**THEOREM 4.12.** [37] *The CB-rank of every automatically presentable tree  $\mathcal{T} = (T; \preceq)$  is finite and computable from  $\mathcal{T}$ .*

The main idea of the proof is to associate to each automatically presentable tree  $\mathcal{T}$  an automatically presentable linear ordering, the Kleene–Brouwer ordering for instance, whose FC-rank can be used to bound the CB-rank of  $\mathcal{T}$ . By Theorem 4.7 this FC-rank is finite.

*König's Lemma.* König's Lemma says that an infinite finitely-branching tree has an infinite path. Here is an automatic analogue.

**THEOREM 4.13.** *If  $\mathcal{T} = (T; \preceq)$  is an automatic copy of an infinite finitely-branching tree, then  $\mathcal{T}$  has a regular infinite path. That is, there exists a regular set  $P \subseteq T$  where  $P$  is an infinite path of  $\mathcal{T}$ .*

**PROOF.** Define a set  $P$  as those elements  $x$  such that  $\exists^\infty w[x \prec w]$  and for which every  $y \prec x$  satisfies that

$$\forall z, z' \in S(y)[z \preceq x \Rightarrow z \leq_{\text{lex}} z'].$$

Then  $P$  is the length-lexicographically least infinite path of  $\mathcal{T}$  (in the ordering induced by the finite strings presenting the tree).  $\dashv$

However, using the 2-Ramsey quantifier we can do more.

**THEOREM 4.14.** *If  $\mathcal{T}$  is an automatic copy of a tree with countably many infinite paths, then every infinite path is regular.*

**PROOF.** Denote by  $E(\mathcal{T}) \subseteq T$  the set of elements of a tree  $\mathcal{T}$  that are on infinite paths. It is definable in  $\mathcal{T}$  using the 2-Ramsey quantifier, so Theorem 3.20 gives that  $E(\mathcal{T})$  is regular. Then every isolated path of  $\mathcal{T}$  is regular, since it is definable as  $\{x \in E(\mathcal{T}) \mid p \preceq x\} \cup \{x \in E(\mathcal{T}) \mid x \prec p\}$ , for suitable  $p \in E(\mathcal{T})$ . Replace  $\mathcal{T}$  by its derivative  $d(\mathcal{T})$ , which is also automatically presentable. Since the CB-rank of  $\mathcal{T}$  is finite (Theorem 4.12), and  $d^{\text{CB}}(\mathcal{T})$  is the empty tree, every infinite path is defined in this way.  $\dashv$

The trees considered above are partial orders. Instead we may consider successor trees: structures of the form  $(T; S, r)$  where  $S$  is the immediate successor relation and  $r$  is the root. These behave quite differently from trees in the signature of partial orders. By Theorem 3.53, the isomorphism problem for successor trees is  $\Sigma_1^1$ -complete. A slight modification of Corollary 3.41 yields the following pathology.

**PROPOSITION 4.15.** *There is an automatic copy  $(T; S, r)$  of a successor tree with exactly two infinite paths, neither of which is a regular subset of  $T$ .*

**4.4. Boolean algebras.** Boolean algebras are considered in the signature  $(\vee, \wedge, \neg)$ . Write  $0$  for the bottom element,  $1$  for the top and  $\subseteq$  for the associated partial order.

The following lemma, which will be useful for classifying the automatically presentable Boolean algebras, is a direct application of Proposition 3.46.

**LEMMA 4.16.** *Suppose  $\mathcal{B}$  is an automatic copy of a Boolean algebra. There exists a constant  $e \in \mathbb{N}$ , that depends only on the automaton recognising  $\vee$ , such that for every set  $S$  of  $n$  elements of  $\mathcal{B}$ , the length of the element  $\bigvee S$  is at most*

$$\max_{s \in S}\{|s|\} + e \log n.$$

**PROPOSITION 4.17.** *The countable atomless Boolean algebra is not automatically presentable.*

PROOF. Suppose  $\mathcal{B} = (B; \vee, \wedge, \neg)$  is an automatic copy of the countable atomless Boolean algebra. The idea is that, starting with the top element  $\mathbf{I}$ , we can generate too many pairwise disjoint elements by repeated splitting.

For non-zero  $x \in B$ , there is exists a non-zero element  $a < x$ , because  $x$  is not an atom. Thus there are two disjoint non-zero elements  $(x \wedge a)$  and  $(x \wedge \neg a)$  below  $x$ . Define  $a(x)$  as the length-lexicographically least element  $a$  with this property. Expand the presentation of  $\mathcal{B}$  to include the functions  $f_l: x \mapsto x \wedge a(x)$  and  $f_r: x \mapsto x \wedge \neg a(x)$ . Apply Proposition 3.44 with respect to the functions  $f_l, f_r$  and with  $D = \{\mathbf{I}\}$ : There exists a linear function  $t$  so that every string in  $G_n(\mathbf{I})$  has length at most  $t(n)$ .

The subalgebra  $\mathcal{A}$  of  $\mathcal{B}$  generated by  $G_n(\{\mathbf{I}\})$  has  $\exp(n)$  atoms. By Lemma 4.16, there is a constant  $e$ , so that every element of  $\mathcal{A}$  has length at most  $t(n) + en$ . So, there are only  $|\Sigma|^{t(n)+en}$  available strings to code for elements of  $\mathcal{A}$ ; contradicting the fact that  $\mathcal{A}$  has  $\exp_2(n)$  elements.  $\dashv$

Consequently, no automatically presentable Boolean algebra has the property that the countable atomless boolean algebra is definable in it. In particular, if an automatically presentable Boolean algebra has finitely many atoms, then it is finite.

**COROLLARY 4.18.** *There are no infinite unary-automatically presentable Boolean algebras.*

PROOF. Suppose  $\mathcal{B}$  is a unary-automatic copy of a Boolean algebra with infinitely many atoms. The set of atoms  $B_A$ , being first-order definable in  $\mathcal{B}$ , is regular. Apply Proposition 3.44 to the functions  $\vee, \wedge$ , and  $\neg$ , and the set  $B_A$ . For  $n \geq 3$ , the set  $G_{n+2}(B_A)$  contains every element generated by the first  $n$  atoms. So  $|G_{n+2}(B_A)| \geq 2^n$ , which exceeds the linear bound.  $\dashv$

Recall that  $\mathcal{B}_{\text{fin}}$  is the Boolean algebra of finite or co-finite subsets of  $\mathbb{N}$ . Write  $\mathcal{B}_{\text{fin}}^n$  for the  $n$ -fold power. A refinement of the proof of Proposition 4.17 is used to classify the automatically presentable Boolean algebras in the non-unary case.

**THEOREM 4.19.** [35] *An infinite Boolean algebra is automatically presentable if and only if it is isomorphic to  $\mathcal{B}_{\text{fin}}^n$  for some  $n \in \mathbb{N}$ .*

**COROLLARY 4.20.** *The isomorphism problem for automatically presentable Boolean algebras is decidable.*

PROOF. Start with an automatic copy of an infinite Boolean algebra  $\mathcal{B}$ . Recall that the Frechét congruence on  $\mathcal{B}$  consists of those pairs whose symmetric difference is either  $\mathbf{0}$  or a meet of a finite number of atoms of  $\mathcal{B}$ . This congruence is first-order definable in  $\mathcal{B}$  with the additional quantifier  $\exists^\infty$ , and so is regular. Hence the quotient of  $\mathcal{B}$  by its Frechét congruence is automatically presentable. Compute the least number of times one has to factor  $\mathcal{B}$  until one reaches the two element algebra, say  $n$  times. This value defines  $\mathcal{B}$  up to isomorphism; indeed,  $\mathcal{B}$  is isomorphic to  $\mathcal{B}_{\text{fin}}^n$ .  $\dashv$

**4.5. Groups, rings and fields.** By algebraically characterising the relations recognised by automata over a unary alphabet, Blumensath establishes the following theorem.

**THEOREM 4.21.** [9] *There are no infinite unary-automatically presentable groups  $(G; \cdot)$ , rings, or fields.*

Let us consider the non-unary cases. Recall that an integral domain  $(D; +, \theta, \cdot, I)$  is a commutative ring with identity with the property that if  $x \cdot y = \theta$  then  $x = \theta$  or  $y = \theta$ . With a little work, the ideas in Section 3.5 (Restriction on growth) can be used to prove the next result.

**THEOREM 4.22.** [35] *There are no infinite automatically presentable integral domains. In particular, there are no infinite automatically presentable fields.*

However, the cases of groups and rings are open. The current state of affairs is nicely detailed in Nies [46]. I only mention a sample here.

For  $k \geq 2$ , write  $\mathbb{Z}[1/k]$  for the additive group of rationals of the form  $z/k^i$ , where  $z \in \mathbb{Z}$  and  $i \in \mathbb{N}$ . It is straightforward to show that  $\mathbb{Z}[1/k]$  and  $\mathbb{Z}[1/k]/\mathbb{Z}$  have automatic presentations.

**PROPOSITION 4.23.** [35] *The following groups do not have automatic presentations:*

- (i) *The positive rationals under multiplication.*
- (ii) *The direct sum of infinitely many copies of  $\mathbb{Z}[1/k]$ , for a fixed  $k$ .*
- (iii) *The direct sum of infinitely many copies of the Prüfer group  $\mathbb{Z}[1/k]/\mathbb{Z}$ , for a fixed  $k$ .*

**PROOF.** The idea, illustrated here in the proof of the first item, is to define sets  $F_n$  of lots of distinct elements using the primes of length at most  $n$ , and then show, as before using Proposition 3.46, that the lengths of these elements are too ‘short’.

Suppose that  $(Q^+, \times)$  is automatically presentable; say an automatic copy is  $(D; \times)$ . Let  $P$  be the elements of  $D$  that correspond to primes. Recall  $P^{\leq n}$  are the elements of  $P$  of length at most  $n$ . Let

$$F_n = \left\{ \prod_{p \in P^{\leq n}} p^{\beta_p} : 0 \leq \beta_p \leq 2^n \right\}.$$

Then  $F_n$  contains  $\exp(nr_n)$  distinct elements, where  $r_n = |P^{\leq n}|$ .

On the other hand, we compute an upperbound on the size of  $F_n$ . By successive applications of Proposition 3.46, there is a constant  $e$ , so that  $|p^\beta| \leq n + en \leq n(e+1)$  and so each string has length at most  $n(e+1) + e \log r_n$ . This places an upper bound of  $\exp(n(e+1) + e \log r_n)$  on the size of  $F_n$ , contradicting that  $r_n$  goes to infinity.  $\dashv$

Using a mixture of algebra, coding techniques, and growth arguments, Nies and Thomas [47] prove the next result (the first item was already established for finitely generated groups  $\mathcal{G}$  by Oliver and Thomas [48]).

- THEOREM 4.24.** (i) *Let  $\mathcal{G}$  be an automatically presentable infinite group. Then every finitely generated subgroup of  $\mathcal{G}$  has an Abelian subgroup of finite index.*  
(ii) *Every finitely generated subring of an automatically presentable ring is finite.*

**§5. Open problems.** This work has only dealt with automata operating on finite strings. The fundamental result that makes the theory work is Theorem 3.1 [Definability]. Analogues of this theorem hold for other models of finite automata: automata operating on finite trees (ranked or unranked), infinite strings, and infinite trees. Each model yields a class of ‘automatically presentable structures’ where the basic theory goes through: particularly, a logical characterisation via interpretability similar to Theorem 3.8. Of course, the problems considered in this survey—such as classification, intrinsic regularity, and proving non-automaticity—can be asked in these more general settings, see for instance [24, 39, 21].

However, there are still many problems in the finite string case. Here are some problems whose solutions will likely require new techniques.

*Problem 5.1.* Find new ways to prove that a structure does not have an automatic presentation. For instance, do the following structures have automatic presentations?

- (i) The additive group of rationals  $(\mathbb{Q}; +)$ . This structure is WMSO-interpretable in a certain infinite string (compare [46]). This limits the kind of proof that could be used to show that  $(\mathbb{Q}; +)$  has no finite-string automatic presentation.
- (ii) The graph generated by the ground term rewriting system with one constant  $a$ , one binary function  $f$ , initial term  $f(a, a)$  and rewriting rule  $a \mapsto f(a, a)$ . In other words, the structure whose domain consists of all finite binary branching trees  $t$ , and for which there is an edge from  $t$  to  $t'$  if  $t'$  extends  $t$  by exactly one node. This is a candidate for separating the class of graphs generated by ground term rewriting systems from the finite-string automatic graphs (see [42]).

*Problem 5.2.* Investigate the complexity in the arithmetic/analytic hierarchy of the isomorphism problem for classes of automatically presentable structures. For instance, what is the complexity of the isomorphism problem for finite-string automatically presentable linear orders, or equivalence structures?

*Problem 5.3.* Can we capture intrinsic regularity using definability?

For instance, is it the case that for every finite-string automatically presentable structure  $\mathcal{A}$ ,

$$\text{FO}[\cup_n \mathcal{Q}_n^{\text{reg}}]_{\omega-\text{inv}}(\mathcal{A}) = \text{IR}(\mathcal{A})?$$

## REFERENCES

- [1] C. J. ASH and A. NERODE, *Intrinsically recursive relations*, *Aspects of effective algebra (clayton, 1979)*, Upside Down A Book Co., Yarra Glen, 1981, pp. 26–41.
- [2] V. BÁRÁNY, *Invariants of automatic presentations and semi-synchronous transductions*, *STACS 2006* (B. Durand and W. Thomas, editors), Lecture Notes in Computer Science, vol. 3884, Springer, 2006, pp. 289–300.
- [3] ———, *Automatic presentations of infinite structures*, Ph.D. thesis, RWTH Aachen, 2007.
- [4] V. BÁRÁNY, L. KAISER, and S. RUBIN, *Countable omega-automatic structures and their presentations*, *STACS 2008* (S. Albers, P. Weil, and C. Rochange, editors), Dagstuhl Seminar Proceedings, vol. 08001, Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2008.
- [5] M. BENEDIKT and L. LIBKIN, *Tree extension algebras: logics, automata, and query languages*, *LICS 2002*, IEEE Computer Society, 2002, pp. 203–212.
- [6] M. BENEDIKT, L. LIBKIN, T. SCHWENTICK, and L. SEGOUFIN, *A model-theoretic approach to regular string relations*, *LICS 2001*, IEEE Computer Society, 2001, pp. 431–440.
- [7] C. H. BENNETT, *Logical reversibility of computation*, *IBM Journal of Research and Development*, vol. 17 (1973), no. 6, pp. 525–532.
- [8] JEAN BERSTEL, *Transductions and context-free languages*, Leitfäden der Angewandten Mathematik und Mechanik [Guides to Applied Mathematics and Mechanics], vol. 38, B. G. Teubner, Stuttgart, 1979.
- [9] A. BLUMENSATH, *Automatic structures*, Diploma thesis, RWTH Aachen, 1999.
- [10] ———, *Prefix-recognisable graphs and monadic second-order logic*, Technical report, RWTH Aachen, 2001.
- [11] ———, *Axiomatising tree-interpretable structures*, *STACS 2002* (H. Alt and A. Ferreira, editors), Lecture Notes in Computer Science, vol. 2285, Springer, 2002, pp. 596–607.
- [12] A. BLUMENSATH and E. GRÄDEL, *Automatic structures*, *LICS 2000*, IEEE Computer Society, 2000, pp. 51–62.
- [13] ———, *Finite presentations of infinite structures: Automata and interpretations*, *Proceedings of the 2nd International Workshop on Complexity in Automated Deduction, CiAD 2002*, 2002.
- [14] ———, *Finite presentations of infinite structures: Automata and interpretations*, *Theory of Computing Systems*, vol. 37 (2004), pp. 641–674.
- [15] V. BRUYÈRE, G. HANSÉ, C. MICHAUX CHRISTIAN, and R. VILLEMAIRE, *Logic and p-recognizable sets of integers*, *Bulletin of the Belgian Mathematical Society. Simon Stevin*, vol. 1 (1994), no. 2, pp. 191–238, Journées Montoises (Mons, 1992).
- [16] J. R. BÜCHI, *Weak second-order arithmetic and finite automata*, *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, vol. 6 (1960), pp. 66–92.
- [17] ———, *On a decision method in restricted second order arithmetic*, *Logic, methodology and philosophy of science*, Stanford University Press, 1962, pp. 1–11.
- [18] J. W. CANNON, *The combinatorial structure of cocompact discrete hyperbolic groups*, *Geometriae Dedicata (Historical Archive)*, vol. 16 (1984), no. 2, pp. 123–148.
- [19] J. W. CANNON, D. B. H. EPSTEIN, D. F. HOLT, S. V. F. LEVY, M. S. PATERSON, and W. P. THURSTON, *Word processing in groups*, Jones and Bartlett, 1992.
- [20] D. CENZER and J. B. REMMEL, *Complexity-theoretic model theory and algebra*, *Handbook of recursive mathematics*, vol. 1 (Yu. L. Ershov, S. S. Goncharov, A. Nerode, and J. B. Remmel, editors), Studies in Logic and the Foundations of Mathematics, vol. 138, North-Holland, Amsterdam, 1998, pp. 381–513.
- [21] T. COLCOMBET and C. LÖDING, *Transforming structures by set interpretations*, Technical Report AIB-2006-07, RWTH Aachen, 2006.

- [22] C. DELHOMMÉ, *Non-automaticity of  $\omega^\omega$* , 2001, manuscript.
- [23] ———, *The Rado graph is not automatic*, 2001, manuscript.
- [24] ———, *Automaticité des ordinaux et des graphes homogènes*, *Comptes Rendus Mathématique*, vol. 339 (2004), no. 1, pp. 5–10.
- [25] J. DONER, *Tree acceptors and some of their applications*, *Journal of Computer and System Sciences*, vol. 4 (1970), pp. 406–451.
- [26] S. EILENBERG, C. C. ELGOT, and J. C. SHEPHERDSON, *Sets recognised by  $n$ -tape automata*, *Journal of Algebra*, vol. 13 (1969), no. 4, pp. 447–464.
- [27] C. C. ELGOT, *Decision problems of finite automata design and related arithmetics*, *Transactions of the American Mathematical Society*, vol. 98 (1961), pp. 21–51.
- [28] S. S. GONCHAROV and J. F. KNIGHT, *Computable structure and non-structure theorems*, *Algebra and Logic*, vol. 41 (2002), no. 6, pp. 351–373.
- [29] L. HELLA, *Definability hierarchies of generalized quantifiers*, *Annals of Pure and Applied Logic*, vol. 43 (1989), no. 3, pp. 235–271.
- [30] B. R. HODGSON, *Théories décidables par automate fini*, Ph.D. thesis, University of Montréal, 1976.
- [31] ———, *On direct products of automaton decidable theories*, *Theoretical Computer Science*, vol. 19 (1982), pp. 331–335.
- [32] ———, *Decidabilite par automate fini*, *Annales de Sciences Mathématiques du Québec*, vol. 7 (1983), pp. 39–57.
- [33] D. F. HOLT, *The Warwick automatic groups software*, *Geometric and computational perspectives on infinite groups (Minneapolis, MN and New Brunswick, NJ, 1994)*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 25, American Mathematical Society, 1996, pp. 69–82.
- [34] B. KHOSSAINOV and A. NERODE, *Automatic presentations of structures*, Lecture Notes in Computer Science, vol. 960, 1995.
- [35] B. KHOSSAINOV, A. NIES, S. RUBIN, and F. STEPHAN, *Automatic structures: Richness and limitations*, *Logical Methods in Computer Science*, vol. 3 (2007), no. 2, pp. 0–0.
- [36] B. KHOSSAINOV, S. RUBIN, and F. STEPHAN, *Definability and regularity in automatic structures*, *STACS 2004* (V. Diekert and M. Habib, editors), Lecture Notes in Computer Science, vol. 2996, Springer, 2004, pp. 440–451.
- [37] ———, *Automatic linear orders and trees*, *ACM Transactions on Computational Logic*, vol. 6 (2005), no. 4, pp. 675–700.
- [38] D. KUSKE, *Is Cantor's theorem automatic?*, *LPAR 2003* (M. Y. Vardi and A. Voronkov, editors), Lecture Notes in Artificial Intelligence, vol. 2850, Springer, 2003, pp. 332–345.
- [39] D. KUSKE and M. LOHREY, *First-order and counting theories of omega-automatic structures*, Fakultätsbericht Nr. 2005/07, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, September 2005.
- [40] L. LIBKIN and F. NEVEN, *Logical definability and query languages over unranked trees*, *LICS 2003*, IEEE Computer Society, 2003, pp. 178–187.
- [41] PER LINDSTRÖM, *First order predicate logic with generalized quantifiers*, *Theoria*, vol. 32 (1966), pp. 186–195.
- [42] CHRISTOF LÖDING, *Infinite graphs generated by tree rewriting*, Ph.D. thesis, RWTH Aachen, 2003.
- [43] M. LOHREY, *Automatic structures of bounded degree*, *LPA 2003* (M. Y. Vardi and A. Voronkov, editors), Lecture Notes in Artificial Intelligence, vol. 2850, Springer, 2003, pp. 344–358.
- [44] C. MICHAUX and F. POINT, *Les ensembles  $k$ -reconnaissables sont définissables dans  $\langle N, +, V_k \rangle$* , *Comptes Rendus des Séances de l'Académie des Sciences. Série I. Mathématique*, vol. 303 (1986), no. 19, pp. 939–942.

- [45] A. A. NABEBIN, *Multitape automata in a unary alphabet*, *Trudy Moskovskogo Ordona Lenina Ènergeticheskogo Instituta*, vol. 292 (1976), pp. 7–11.
- [46] A. NIES, *Describing Groups*, this BULLETIN, vol. 13 (2007), no. 3, pp. 305–339.
- [47] A. NIES and R. THOMAS, *FA-presentable groups and rings*, 2005, to appear.
- [48] G. OLIVER and R. THOMAS, *Automatic presentations of finitely generated groups*, *STACS 2005* (V. Diekert and B. Durand, editors), Lecture Notes in Computer Science, vol. 3404, Springer, 2005.
- [49] M. O. RABIN, *Decidability of second-order theories and automata on infinite trees*, *Transactions of the American Mathematical Society*, vol. 141 (1969), pp. 1–35.
- [50] J. G. ROSENSTEIN, *Linear orderings*, Academic Press, 1982.
- [51] S. RUBIN, *Automatic structures*, Ph.D. thesis, University of Auckland, 2004.
- [52] F. STEPHAN, *The random graph is not automatically presentable*, 2002, manuscript.
- [53] A. SZILARD, S. YU, K. ZHANG, and J. SHALLIT, *Characterizing regular languages with polynomial densities*, *MFCS '92* (I. M. Havel and V. Koubek, editors), Lecture Notes in Computer Science, vol. 629, Springer, 1992, pp. 494–503.
- [54] J. W. THATCHER and J. B. WRIGHT, *Generalized finite automata theory with an application to a decision problem of second-order logic*, *Mathematical Systems Theory*, vol. 2 (1968), pp. 57–81.
- [55] W. THOMAS, *Automata on infinite objects*, *Handbook of theoretical computer science* (J. van Leeuwen, editor), vol. B: Formal models and semantics, Elsevier, 1990, pp. 133–191.
- [56] B. A. TRAHENBROT, *Finite automata and the logic of one-place predicates*. Russian, *Siberian Mathematical Journal*, vol. 3 (1962), pp. 103–131, English translation: *American Mathematical Society Translations, Series 2*, vol. 59 (1966), pp. 23–55.
- [57] R. VILLEMAIRE, *The theory of  $\langle N, +, V_k, V_l \rangle$  is undecidable*, *Theoretical Computer Science*, vol. 106 (1992), pp. 337–349.

DEPARTMENT OF COMPUTER SCIENCE  
 UNIVERSITY OF AUCKLAND, NEW ZEALAND  
*E-mail:* rubin@cs.auckland.ac.nz

## **4.2 Publication**

The rest of this page is intentionally left blank

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/228594972>

# Automata-based presentations of infinite structures

Article · January 2009

DOI: 10.1017/CBO9780511974960.002

---

CITATIONS

10

---

READS

35

3 authors, including:



Vince Bárány

Google Inc.

22 PUBLICATIONS 203 CITATIONS

[SEE PROFILE](#)



Erich Graedel

RWTH Aachen University

158 PUBLICATIONS 3,780 CITATIONS

[SEE PROFILE](#)

All content following this page was uploaded by [Erich Graedel](#) on 12 April 2017.

The user has requested enhancement of the downloaded file. All in-text references [underlined in blue](#) are added to the original document and are linked to publications on ResearchGate, letting you access and read them immediately.

# Automata-based presentations of infinite structures

Vince Bárány<sup>1</sup>and Erich Grädel<sup>2</sup>and Sasha Rubin<sup>3</sup>

<sup>1</sup> Oxford University Computing Laboratory  
Wolfson Building, Parks Road, Oxford OX1 3QD, United Kingdom  
[vbarany@logic.rwth-aachen.de](mailto:vbarany@logic.rwth-aachen.de)

<sup>2</sup> Mathematical Foundations of Computer Science  
RWTH Aachen, D-52056 Aachen, Germany  
[graedel@logic.rwth-aachen.de](mailto:graedel@logic.rwth-aachen.de)

<sup>3</sup> Department of Mathematics and Applied Mathematics  
University of Cape Town, Private Bag, Rondebosch 7701, South Africa  
[srubin@math.cornell.edu](mailto:srubin@math.cornell.edu)



# 1

## Automata-based presentations of infinite structures

Vince Bárány<sup>1</sup> and Erich Grädel<sup>2</sup> and Sasha Rubin<sup>3</sup>

### 1.1 Finite presentations of infinite structures

The model theory of finite structures is intimately connected to various fields in computer science, including complexity theory, databases, and verification. In particular, there is a close relationship between complexity classes and the expressive power of logical languages, as witnessed by the fundamental theorems of descriptive complexity theory, such as Fagin’s Theorem and the Immerman-Vardi Theorem (see [78, Chapter 3] for a survey).

However, for many applications, the strict limitation to finite structures has turned out to be too restrictive, and there have been considerable efforts to extend the relevant logical and algorithmic methodologies from finite structures to suitable classes of infinite ones. In particular this is the case for databases and verification where infinite structures are of crucial importance [130]. *Algorithmic model theory* aims to extend in a systematic fashion the approach and methods of finite model theory, and its interactions with computer science, from finite structures to finitely-presentable infinite ones.

There are many possibilities to present infinite structures in a finite manner. A classical approach in model theory concerns the class of *computable structures*; these are countable structures, on the domain of nat-

<sup>1</sup> Oxford University Computing Laboratory  
Wolfson Building, Parks Road, Oxford OX1 3QD, United Kingdom  
[vbarany@logic.rwth-aachen.de](mailto:vbarany@logic.rwth-aachen.de)

<sup>2</sup> Mathematical Foundations of Computer Science  
RWTH Aachen, D-52056 Aachen, Germany  
[graedel@logic.rwth-aachen.de](mailto:graedel@logic.rwth-aachen.de)

<sup>3</sup> Department of Mathematics and Applied Mathematics  
University of Cape Town, Private Bag, Rondebosch 7701, South Africa  
[srubin@math.cornell.edu](mailto:srubin@math.cornell.edu)

ural numbers, say, with a finite collection of computable functions and relations. Such structures can be finitely presented by a collection of algorithms, and they have been intensively studied in model theory since the 1960s. However, from the point of view of algorithmic model theory the class of computable structures is problematic. Indeed, one of the central issues in algorithmic model theory is the effective evaluation of logical formulae, from a suitable logic such as first-order logic (FO), monadic second-order logic (MSO), or a fixed point logic like LFP or the modal  $\mu$ -calculus. But on computable structures, only the quantifier-free formulae generally admit effective evaluation, and already the existential fragment of first-order logic is undecidable, for instance on the computable structure  $(\mathbb{N}, +, \cdot)$ .

This leads us to the central requirement that for a suitable logic  $L$  (depending on the intended application) the model-checking problem for the class  $\mathcal{C}$  of finitely presented structures should be algorithmically solvable. At the very least, this means that the  $L$ -theory of individual structures in  $\mathcal{C}$  should be decidable. But for most applications somewhat more is required:

**Effective semantics:** There should be an algorithm that, given a finite presentation of a structure  $\mathfrak{A} \in \mathcal{C}$  and a formula  $\psi(\bar{x}) \in L$ , expands the given presentation to include the relation  $\psi^{\mathfrak{A}}$  defined by  $\psi$  on  $\mathfrak{A}$ .

This also implies that the class  $\mathcal{C}$  should be closed under some basic operations (such as logical interpretations). Thus we should be careful to restrict the model of computation. Typically, this means using some model of *finite automata* or a very restricted form of rewriting.

In general, the finite means for presenting infinite structures may involve different approaches: logical interpretations; finite axiomatisations; rewriting of terms, trees, or graphs; equational specifications; the use of synchronous or asynchronous automata, etc. The various possibilities can be classified along the following lines:

**Internal:** a set of finite or infinite words or trees/terms is used to represent the domain of (an isomorphic copy of) the structure. Finite automata/rewriting-rules compute the domain and atomic relations (eg. prefix-recognisable graphs, automatic structures).

**Algebraic:** a structure is represented as the least solution of a finite set of recursive equations in an appropriately chosen algebra of finite and countable structures (eg. VR-equational structures).

**Logical:** structures are described by interpreting them, using a finite collection of formulae, in a fixed structure (eg. tree-interpretable structures). A different approach consists in (recursively) axiomatising the isomorphism class of the structure to be represented.

**Transformational:** structures are defined by sequences of prescribed transformations, such as graph-unraveling, or Muchnik's iterations, applied to certain fixed initial structures (which are already known to have a decidable theory). Transformations can also be transductions, logical interpretations, etc. [23]

The last two approaches overlap somewhat. Also, the algebraic approach can be viewed *generatively*: convert the equational system into an appropriate *deterministic grammar* generating the solution of the original equations [44]. The grammar is thus the finite presentation of the graph. One may also say that internal presentations and generating grammars provide descriptions of the *local structure* from which the whole arises, as opposed to descriptions based on *global symmetries* typical of algebraic specifications.

### Prerequisites and notation

We assume rudimentary knowledge of finite automata on finite and infinite words and trees, their languages and their correspondence to monadic second-order logic (MSO) [133, 79]. Undefined notions from logic and algebra (congruence on structures, definability, isomorphism) can be found in any standard textbook. We mainly consider the following logics  $\mathcal{L}$ : first-order (FO), monadic second order (MSO), and weak monadic second-order (wMSO) which has the same syntax as MSO, but the intended interpretation of the set variables is that they range over *finite* subsets of the domain of the structure under consideration.

We mention the following to fix notation: infinite words are called  $\omega$ -words and infinite trees are called  $\omega$ -trees (to distinguish them from finite ones); relations computable by automata will be called *regular*; the domain of a *structure*  $\mathfrak{B}$  is usually written  $B$  and its relations are written  $R^{\mathfrak{B}}$ . An MSO-formula  $\phi(X_1, \dots, X_j, x_1, \dots, x_k)$  interpreted in  $\mathfrak{B}$  *defines* the set  $\phi^{\mathfrak{B}} := \{(B_1, \dots, B_j, b_1, \dots, b_k) \mid B_i \subset B, b_i \in B, \mathfrak{B} \models \phi(B_1, \dots, B_j, b_1, \dots, b_k)\}$ . A wMSO-formula is similar except that the  $B_i$  range over finite subsets of  $B$ . The *full binary tree*  $\mathfrak{T}_2$  is defined as the structure

$$(\{0, 1\}^*, \mathbf{suc}_0, \mathbf{suc}_1)$$

where the successor relation  $\mathbf{suc}_i$  consists of all pairs  $(x, xi)$ . Tree automata operate on  $\Sigma$ -labelled trees  $T : \{0, 1\}^* \rightarrow \Sigma$ . Such a tree is identified with the structure

$$(\{0, 1\}^*, \mathbf{suc}_0, \mathbf{suc}_1, \{T^{-1}(\sigma)\}_{\sigma \in \Sigma}).$$

Rabin proved the decidability of the MSO-theory of  $\mathfrak{T}_2$  and the following fundamental correspondence between MSO and tree automata (see [132] for an overview):

For every monadic second-order formula  $\varphi(\bar{X})$  in the signature of  $\mathfrak{T}_2$  there is a tree automaton  $\mathcal{A}$  (and vice versa) such that

$$L(\mathcal{A}) = \{T_{\bar{X}} \mid \mathfrak{T}_2 \models \varphi(\bar{X})\} \quad (1.1)$$

where  $T_{\bar{X}}$  denotes the tree with labels for each  $X_i$ .

Similar definitions and results hold for  $r$ -ary trees, in which case the domain is  $[r]^*$  where  $[r] := \{0, \dots, r - 1\}$ , and finite trees.

In section 1.2.2 and elsewhere we do not distinguish between a term and its natural representation as a tree. Thus we may speak of infinite terms. We consider countable, vertex- and edge-labelled graphs possibly having distinguished vertices (called sources), and no parallel edges of the same label. A graph is *deterministic* if each of its vertices is the source of at most one edge of each edge label.

### Interpretations

Interpretations allow one to define an isomorphic copy of one structure in another. Fix a logic  $\mathcal{L}$ . A  $d$ -dimensional  $\mathcal{L}$ -interpretation  $\mathcal{I}$  of structure  $\mathfrak{B} = (B; (R_i^{\mathfrak{B}})_i)$  in structure  $\mathfrak{A}$ , denoted  $\mathfrak{B} \leq_{\mathcal{L}}^{\mathcal{I}} \mathfrak{A}$ , consists of the following  $\mathcal{L}$ -formulas in the signature of  $\mathfrak{A}$ ,

- a domain formula  $\Delta(\bar{x})$ ,
- a relation formula  $\Phi_{R_i}(\bar{x}_1, \dots, \bar{x}_{r_i})$  for each relation symbol  $R_i$ , and
- an equality formula  $\epsilon(\bar{x}_1, \bar{x}_2)$ ,

where each  $\Phi_{R_i}^{\mathfrak{A}}$  is a relation on  $\Delta^{\mathfrak{A}}$ , each of the tuples  $\bar{x}_i, \bar{x}$  contain the same number of variables,  $d$ , and  $\epsilon^{\mathfrak{A}}$  is a congruence on the structure  $(\Delta^{\mathfrak{A}}, (\Phi_{R_i}^{\mathfrak{A}})_i)$ , so that  $\mathfrak{B}$  is isomorphic to

$$(\Delta^{\mathfrak{A}}, (\Phi_{R_i}^{\mathfrak{A}})_i) / \epsilon^{\mathfrak{A}}.$$

If  $\mathcal{L}$  is FO then the free  $\bar{x}$  are FO and we speak of a *FO interpretation*. If  $\mathcal{L}$  is MSO (wMSO) but the free variables are FO, then we speak of a (*weak*) *monadic second-order interpretation*.

We associate with  $\mathcal{I}$  a transformation of formulas  $\psi \mapsto \psi^{\mathcal{I}}$ . For illustration we define it in the first-order case: the variable  $x_i$  is replaced by the  $d$ -tuple  $\bar{y}_i$ ,  $(\psi \vee \phi)^{\mathcal{I}}$  by  $\psi^{\mathcal{I}} \vee \phi^{\mathcal{I}}$ ,  $(\neg\psi)^{\mathcal{I}}$  by  $\neg\psi^{\mathcal{I}}$ ,  $(\exists x_i \psi)^{\mathcal{I}}$  by  $\exists \bar{y}_i \Delta(\bar{y}_i) \wedge \psi^{\mathcal{I}}$ , and  $(x_i = x_j)^{\mathcal{I}}$  is replaced by  $\epsilon(\bar{y}_i, \bar{y}_j)$ . Thus one can translate  $\mathcal{L}$  formulas from the signature of  $\mathfrak{B}$  into the signature of  $\mathfrak{A}$ .

**Proposition 1.1.1** *If  $\mathfrak{B} \leq_{\mathcal{L}}^{\mathcal{I}} \mathfrak{A}$ , say the isomorphism is  $f$ , then for every formula  $\psi(x_1, \dots, x_k)$  in the signature of  $\mathfrak{B}$  and all  $k$ -tuples  $\bar{b}$  of elements of  $\mathfrak{B}$  it holds that*

$$\mathfrak{B} \models \psi(b_1, \dots, b_k) \iff \mathfrak{A} \models \psi^{\mathcal{I}}(f(b_1), \dots, f(b_k))$$

In particular, if  $\mathfrak{A}$  has decidable  $\mathcal{L}$ -theory, then so does  $\mathfrak{B}$ .

### Set interpretations

When  $\mathcal{L}$  is MSO (wMSO) and the free variables are MSO (wMSO) the interpretation is called a *(finite) set interpretation*. In this last case, we use the notation  $\mathfrak{B} \leq_{\text{set}}^{\mathcal{I}} \mathfrak{A}$  or  $\mathfrak{B} \leq_{\text{fset}}^{\mathcal{I}} \mathfrak{A}$ . We will only consider (finite) set interpretations of dimension 1.

If finiteness of sets is MSO-definable in some structure  $\mathfrak{A}$  (as for linear orders or for finitely branching trees) then every structure  $\mathfrak{B}$  having a finite-set interpretation in  $\mathfrak{A}$  can also be set interpreted in  $\mathfrak{A}$ .

**Example 1.1.2** An interpretation  $(\mathbb{N}, +) \leq_{\text{fset}}^{\mathcal{I}} (\mathbb{N}, 0, \text{suc})$  based on the binary representation is given by  $\mathcal{I} = (\varphi(X), \varphi_+(X, Y, Z), \varphi_=(X, Y))$  with  $\varphi(X)$  always true,  $\varphi_+$  the identity, and  $\varphi_+(X, Y, Z)$  is

$$\exists C \forall n [(Zn \leftrightarrow Xn \oplus Yn \oplus Cn) \wedge (C(\text{suc}n) \leftrightarrow \mu(Xn, Yn, Cn)) \wedge \neg C0]$$

where  $C$  stands for carry,  $\oplus$  is exclusive or, and  $\mu(x_0, x_1, x_2)$  is the majority function, in this case definable as  $\bigvee_{i \neq j} x_i \wedge x_j$ .

To every (finite) subset interpretation  $\mathcal{I}$  we associate, as usual, a transformation of formulas  $\psi \mapsto \psi^{\mathcal{I}}$ , in this case mapping first-order formulas to (weak) monadic second-order formulas.

**Proposition 1.1.3** *Let  $\mathfrak{B} \leq_{(\text{f})\text{set}}^{\mathcal{I}} \mathfrak{A}$  be a (finite) subset interpretation with isomorphism  $f$ . Then to every first-order formula  $\psi(x_1, \dots, x_k)$  in the signature of  $\mathfrak{B}$  one can effectively associate a (weak) monadic second-order formula  $\psi^{\mathcal{I}}(X_1, \dots, X_k)$  in the signature of  $\mathfrak{A}$  such that for all  $k$ -tuples  $\bar{b}$  of elements of  $\mathfrak{B}$  it holds that*

$$\mathfrak{B} \models \psi(b_1, \dots, b_k) \iff \mathfrak{A} \models \psi^{\mathcal{I}}(f(b_1), \dots, f(b_k)) .$$

Consequently, if the (weak) monadic-second order theory of  $\mathfrak{A}$  is decidable then so is the first-order theory of  $\mathfrak{B}$ .

For more on subset interpretations we refer to [23].

## 1.2 A hierarchy of finitely presentable structures

This section provides an overview of some of the prominent classes of graphs and their various finite presentations.

These developments are the product of over two decades of research in diverse fields. We begin our exposition with the seminal work of Muller and Schupp on context-free graphs, we mention prefix-recognisable structures, survey hyperedge-replacement and vertex-replacement grammars and their corresponding algebraic frameworks leading up to equational graphs in algebras with asynchronous or synchronous product operation. These latter structures are better known in the literature by their automatic presentations, and constitute the topic of the rest of this survey.

As a unifying approach we discuss how graphs belonging to individual classes can be characterised as least fixed-point solutions of finite systems of equations in a corresponding algebra of graphs. We illustrate on examples how to go from graph grammars through equational presentations and interpretations to internal presentations and vice versa.

We briefly summarise key results on Caucal's pushdown hierarchy and more recent developments on simply-typed recursion schemes and collapsible pushdown automata.

Figure 1.1 provides a summary of some of the graph classes discussed in this section together with the boundaries of decidability for relevant logics. Rational graphs and automatic graphs featured on this diagram are described in detail in Section 1.3.

### 1.2.1 From context-free graphs to prefix-recognisable structures

Context-free graphs were introduced in the seminal papers [110, 111, 112] of Muller and Schupp. There are several equivalent definitions. The objects of study are countable directed edge-labelled, finitely branching graphs. An *end* is a maximal connected<sup>4</sup> component of the induced subgraph obtained by removing, for some  $n$ , the  $n$ -neighbourhood of a fixed

<sup>4</sup> connectedness is taken with respect to the underlying undirected graph.

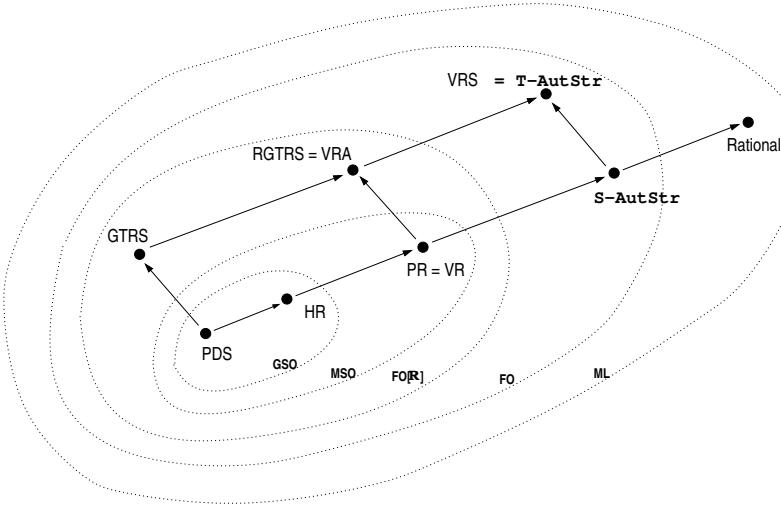


Figure 1.1 Relationship of graph classes and logical decidability boundaries.

vertex  $v_0$ . A vertex of an end is on the *boundary* if it is connected to a vertex in the removed neighbourhood. Two ends are end-isomorphic if there is a graph isomorphism (preserving labels as well) between them that is also a bijection of their boundaries. A graph is *context-free* if it is connected and has only *finitely many ends* up to end-isomorphism. This notion is independent of the  $v_0$  chosen.

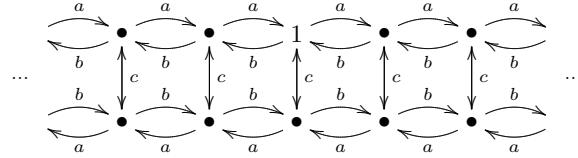
A graph is context-free if and only if it is isomorphic to the connected component of the configuration graph of a pushdown automaton (without  $\epsilon$ -transitions) induced by the set of configurations that are reachable from the initial configuration [112].

A *context-free group* is a finitely generated group  $G$  such that, for some set  $S$  of semigroup generators of  $G$ , the set of words  $w \in S^*$  representing the identity element of  $G$  forms a context-free language. This is independent of the choice of  $S$ . Moreover, a group is context-free if and only if its Cayley graph for some (and hence all) sets  $S$  of semigroup generators is a context-free graph. Finally, a finitely generated group is context-free if and only if it is *virtually free*, that is, if it has a free subgroup of finite index [111].<sup>5</sup>

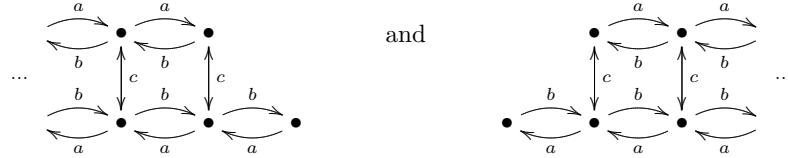
<sup>5</sup> Originally [111] proved this under the assumption of *accessibility*, a notion related to group decompositions introduced by Wall who conjectured that all

Muller and Schupp have further shown that context-free graphs have a decidable MSO-theory. Indeed, every context-free graph can be MSO-interpreted in the full binary tree.

**Example 1.2.1** Consider the group  $G$  given by the finite presentation  $\langle a, b, c \mid ab, cc, acac, bcbc \rangle$ . The Cayley graph  $\Gamma(G, S)$  of  $G$  with respect to the set of semigroup generators  $S = \{a, b, c\}$  is depicted below.



Notice that  $\Gamma(G, S)$  has two ends, for any  $n$ -neighbourhood of the identity with  $n > 1$ . These are



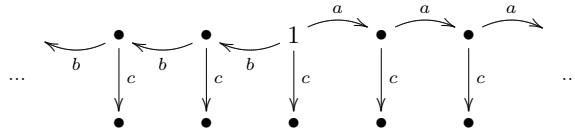
A word  $w \in \{a, b, c\}^*$  represents the identity of  $G$  if, and only if,  $w$  has an even number of  $c$ 's and the number of  $a$ 's equals the number of  $b$ 's. We present a pushdown automaton  $\mathcal{A}$  which recognises this set of words and, moreover, has a configuration graph that is isomorphic to  $\Gamma(G, S)$ . The states of  $\mathcal{A}$  are  $Q = \{1, c\}$  with  $q_0 = 1$  as the initial state, the stack alphabet is  $\Gamma = \{a, b\}$ , the input alphabet is  $\{a, b, c\}$  and  $\mathcal{A}$  has the following transitions:

$$\begin{array}{lll} \text{internal: } & 1\theta & \xrightarrow{c} c\theta \\ \text{internal: } & c\theta & \xrightarrow{c} 1\theta \\ \text{push: } & q\sigma\theta & \xrightarrow{\sigma} q\sigma\sigma\theta \quad \text{for } q = 1, c \text{ and } \sigma = a, b \\ \text{push: } & q\perp & \xrightarrow{\sigma} q\sigma\perp \quad \text{for } q = 1, c \text{ and } \sigma = a, b \\ \text{pop: } & q\sigma\theta & \xrightarrow{\bar{\sigma}} q\theta \quad \text{for } q = 1, c \text{ and } \{\sigma, \bar{\sigma}\} = \{a, b\} \end{array}$$

finitely generated groups would have this property. Muller and Schupp conjectured every context-free group to be accessible, but it was not until Dunwoody [64] proved that all finitely presentable groups are accessible that this auxiliary condition could be dropped from the characterisation of [11]. Unfortunately, many sources forgot to note this fact. Later Dunwoody also gave a counterexample refuting Wall's conjecture.

Here  $\theta$  is the stack content written with its top element on the left and always ending in the special symbol  $\perp$  marking the bottom of the stack.

In every deterministic edge-labelled connected graph and for any ordering of the edge labels one obtains a spanning tree by taking the shortest path with the lexicographically least labeling leading to each node from a fixed source. Take such a spanning tree  $T$  for the example graph  $\Gamma(G, S)$  with root  $1_G$ . Observe that  $T$  is regular, having only finitely many subtrees (ends) up to isomorphism. The ordering  $a < b < c$  induces the spanning tree depicted below. The Cayley graph  $\Gamma(G, S)$  is MSO-interpretable in this regular spanning tree by defining the missing edges using the relators from the presentation of the group.



In particular  $\Gamma(G, S)$  is MSO-interpretable in the full binary tree, and hence has decidable MSO.

A mild generalisation of pushdown transitions, *prefix-rewriting* rules, take the form  $uz \mapsto vz$  where  $u$  and  $v$  are fixed words and  $z$  is a variable ranging over words. As in the previous example, pushdown transitions are naturally perceived as prefix-rewriting rules affecting the state and the top stack symbols. Conversely, Caucal [40] has shown that connected components of configuration graphs of prefix-rewriting systems given by finitely many prefix-rewriting rules are effectively isomorphic to connected components of pushdown graphs. Later, Caucal introduced *prefix-recognisable graphs* as a generalisation of context-free graphs and showed that these are MSO-interpretable in the full binary tree and hence have a decidable MSO-theory [42].

**Definition 1.2.2** (Prefix-recognisable relations) Let  $\Sigma$  be a finite alphabet. The set  $\text{PR}(\Sigma)$  of prefix-recognisable relations over  $\Sigma^*$  is the smallest set of relations such that

- every regular language  $L \subseteq \Sigma^*$  is a prefix-recognisable unary relation;
- if  $R, S \in \text{PR}$  (arities  $r$  and  $s$ ) and  $L$  is regular then  $L \cdot (R \times S) = \{(uv_1, \dots, uv_r, uw_1, \dots, uw_s) \mid u \in L, \bar{v} \in R, \bar{w} \in S\} \in \text{PR}$ ;
- if  $R \in \text{PR}$  of arity  $m > 1$  and  $\{i_1, \dots, i_m\} = \{1, \dots, m\}$ ,  
then  $R^{(\bar{i})} = \{(u_{i_1}, \dots, u_{i_m}) \mid (u_1, \dots, u_m) \in R\} \in \text{PR}$ ;
- if  $R, S \in \text{PR}$  are of the same arity, then  $R \cup S \in \text{PR}$ .

**Example 1.2.3** Consider the lexicographic ordering  $<_{\text{lex}}$  on an ordered alphabet  $\Sigma$ . It is prefix-recognisable being the union of

$$\Sigma^* \cdot (\{\varepsilon\} \times \Sigma^+) \quad \text{and} \quad \Sigma^* \cdot (a\Sigma^* \times b\Sigma^*) \quad \text{for all } a < b \in \Sigma.$$

Following [22] we say that a structure  $\mathfrak{A} = (A, \{R_i\}_i)$  is *prefix-recognizable* if  $A$  is a regular set of words over some finite alphabet  $\Sigma$  and each of the relations  $R_i$  is in  $\text{PR}(\Sigma)$ . Prefix-recognisable structures can be characterized in terms of interpretations. On the basis of tree automata, it is relatively straightforward to show that the prefix-recognisable structures coincide with the structures that are MSO-interpretable in the binary tree  $\mathfrak{T}_2$  [97, 42, 22]. This result has been strengthened by Colcombet [51] to first-order interpretability in the expanded structure  $(\mathfrak{T}_2, \prec)$  (note that the prefix relation  $\prec$  is MSO-definable but not FO definable in  $\mathfrak{T}_2$ ). Colcombet proved that MSO-interpretations and FO-interpretations in  $(\mathfrak{T}_2, \prec)$  have the same power, which gives a new characterisation of prefix-recognisable structures. We summarize these results as follows.

**Theorem 1.2.4** *For every structure  $\mathfrak{A}$ , the following are equivalent.*

- (1)  $\mathfrak{A}$  is isomorphic to a prefix-recognisable structure;
- (2)  $\mathfrak{A}$  is MSO-interpretable in the full binary tree  $\mathfrak{T}_2$ ;
- (3)  $\mathfrak{A}$  is FO-interpretable in  $(\mathfrak{T}_2, \prec)$ .

*In particular, every prefix-recognisable structure has a decidable MSO-theory.*

Below we discuss further characterisations of prefix-recognisable structures in terms of vertex-replacement grammars, or as least solutions of VR-equational systems.

### 1.2.2 Graph grammars and graph algebras

In this section we consider vertex- and edge-labelled graphs. In formal language theory grammars generate sets of finite words. Similarly, context-free graph grammars produce sets of finite graphs - start from an initial nonterminal and rewrite nonterminal vertices and edges according to the derivation rules. Just as for languages, the set of valid derivation trees, or parse trees, forms a regular set of trees labelled by derivation rules of the graph grammar. Conversely, consider a collection  $\Theta$  of graph operations — such as disjoint union, recolourings, etc. — as primitives. Every closed  $\Theta$ -term  $t$  evaluates to a finite graph  $\llbracket t \rrbracket$ , and similarly every

$\Theta$ -term  $t(\bar{x})$  evaluates to a finite graph  $\llbracket t(\bar{x}) \rrbracket$  with non-terminal (hyper)-edges and/or vertices. Formally, evaluation is the unique homomorphism from the initial algebra of  $\Theta$ -terms to the  $\Theta$ -algebra of finite graphs with non-terminals. Each regular tree language  $L$  of closed terms thus represents a family of finite graphs  $\{\llbracket t \rrbracket \mid t \in L\}$ . For a concise treatment of graph grammars and finite graphs we refer to the surveys [69, 59] and the book [53].

Our focus here is on individual countable graphs generated by *deterministic* grammars via ‘complete rewriting’. A suitable framework for formalising complete rewriting, in the context of term rewriting, is convergence in complete partial orders (cpo’s). Since no classical order- or metric-theoretic notion of limit seems to exist for graphs, we use the more general categorical notion of colimit [11]. We outline this framework in which an infinite term (over the graph operations  $\Theta$ ) yields a countable graph; details may be found in [55, 11, 53].

In the category  $\mathbb{G}$  of graphs and their homomorphisms every diagram of the form

$$G_0 \xrightarrow{f_0} G_1 \xrightarrow{f_1} G_2 \xrightarrow{f_2} \dots \xrightarrow{f_{n-1}} G_n \xrightarrow{f_n} G_{n+1} \xrightarrow{f_{n+1}} \dots$$

has a colimit  $G$ , i.e. a kind of least common extension  $G$  of the  $G_n$ s with homomorphisms  $g_n : G_n \rightarrow G$  such that  $g_n = g_{n+1}f_n$  for all  $n$ .<sup>6</sup> We assume that the graph operations in  $\Theta$  determine endofunctors of  $\mathbb{G}$  that are cocontinuous i.e. colimit preserving.

On the other side, take the cpo of finite and infinite terms over the signature  $\Theta \cup \{\perp\}$ , with the empty term  $\perp$  and the extension ordering  $s \sqsubseteq t$ . We may turn it into a category  $\mathbb{T}_\Theta$  with each relation  $s \sqsubseteq t$  inducing a unique arrow  $s \rightarrow t$ . Moreover, in this category, colimits (of diagrams as above) exist and an infinite term  $t$  is the colimit of approximations  $t_0 \rightarrow t_1 \rightarrow \dots$  (think that  $t_i$  is the restriction of  $t$  to the first  $i$  levels). The evaluation mapping  $\llbracket \cdot \rrbracket$  has a unique cocontinuous extension, also denoted  $\llbracket \cdot \rrbracket$ , mapping infinite terms to colimits of graphs.

This completes the basic description. Now consider a grammar  $\mathcal{G}$  whose derivation rules  $(X_i \mapsto t_i(\bar{X}))$  can be expressed by  $\Theta$ -terms. These terms determine cocontinuous endofunctors in the category of terms  $\mathbb{T}_\Theta$ . By the Knaster-Tarski theorem the functors have a least fixed-point  $\bar{G}$ , which by Kleene’s Theorem is attained as the colimit of the chain

<sup>6</sup> There are examples of ascending chains  $G_0 \xrightarrow{f_0} G_1 \xrightarrow{f_1} \dots$  and  $G_0 \xrightarrow{g_0} G_1 \xrightarrow{g_1} \dots$  with identical graphs but different embeddings yielding different colimits, whence there is no apparent canonical way of defining a limit knowing only that each  $G_n$  is embeddable into  $G_{n+1}$ .

$\langle \gamma^n(\bar{\emptyset}) \rangle_n$  with the natural homomorphisms. The graph *generated* by the grammar from the corresponding non-terminal  $X_i$  is defined to be the component  $G_i$  of the colimit  $\overline{G}$ .

Equivalently, given the system of equations  $\mathcal{E}_G = \langle X_i = t_i(\bar{X}) \rangle$  one can construct a syntactic (uninterpreted) solution of  $\mathcal{E}_G$  by ‘unraveling’ these equations from the initial non-terminal  $X_0$  of the grammar. This results in a possibly infinite regular term  $t_G$ , which is precisely the least fixed-point solution for  $X_0$  in  $\mathbb{T}_\Theta$ . By cocontinuity of the evaluation mapping  $\llbracket t_G \rrbracket$  is isomorphic to the least fixed-point solution of  $\mathcal{E}_G$  in  $\mathbb{G}$ , that is to the graph generated by  $\mathcal{G}$ .

In what follows we focus on different sets of graph operations  $\Theta$  (namely, HR, VR and some extensions). It has been observed that for suitable choices of operations, most notably avoiding products, the evaluation mapping can be realised as a monadic second-order interpretation or transduction [11, 60]. Consequently every interpretation  $\llbracket t \rrbracket \leq_{\text{MSO}}^\tau t$  naturally translates to an internal presentation of  $\llbracket t \rrbracket$  using tree automata. Moreover, for a regular term  $t$  the MSO-theory of  $\llbracket t \rrbracket$  is decidable by Rabin’s Theorem.

Finally we mention that all this smoothly extends to solutions of infinite sets of equations [33]. Although unravelling might not result in a regular solution term, as long as it has a decidable MSO-theory so does the solution graph.

### Equational graphs and hyperedge-replacement grammars

Hyperedge-replacement (HR) grammars are a very natural generalisation of context-free grammars from formal language theory. Every HR-grammar defines a ‘language’ of finite graphs just as context-free grammars define languages of finite words. The class of graph languages defined by HR-grammars possesses many structural properties akin to those well-known for context-free languages. The interested reader is referred to the monograph [80].

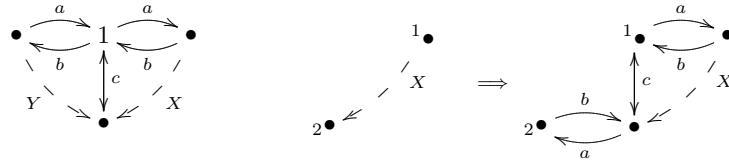
An HR-grammar is given as a finite collection of rules that allow the replacement of any hyperedge of a hypergraph bearing a non-terminal label by the right hand side of a matching rule, which is a given finite hypergraph with a number of distinguished vertices equal to the arity of the hyperedge to be replaced. A copy of the right-hand side of a matching rule is then glued to the original hypergraph precisely at these distinguished vertices and corresponding end vertices of the hyperedge being replaced. Derivation begins with a distinguished non-terminal.

As outlined at the start of section 1.2.2, each *deterministic* HR-grammar

determines a unique countable graph constructed from the initial graph by complete rewriting in the course of which every non-terminal hyperedge is eventually replaced by the right-hand side of the unique matching rule. A countable graph is **HR-equational**, or simply *equational*, if it is generated by a deterministic HR grammar [55]. The class of equational graphs will be denoted by **HR**. Equational graphs constitute a proper extension of the class of context-free graphs [41].

**Proposition 1.2.5** *A connected graph is context-free if, and only if, it is equational and of finite degree.*

**Example 1.2.6** To generate the context-free graph of Example 1.2.1 with a deterministic HR grammar we take as our initial graph the 1-neighbourhood of the root node (labelled with 1 above) and attach to it non-terminal hyperedges labelled with  $X$  and with  $Y$ , respectively, whose vertices enumerate the boundaries of either ends. Similarly, the 1-neighbourhood of the boundary of each end, that is the vertices of the corresponding non-terminal hyperedge, constitutes the right-hand side of the matching rule. Again, non-terminal hyperedges are attached to mark the new boundary. The initial graph and the rule for the non-terminal  $X$  obtained this way are pictured below.



Notice how the linearity of the generated graph is reflected in the linearity of the replacement rules each having only a single non-terminal hyperedge on the right. In the next example a non-linear rule is used to generate a tree, which is not context-free.

**Example 1.2.7** The complete bipartite graph  $K_{1,\omega}$  and the full  $\omega$ -branching tree  $\mathfrak{T}_\omega$  (in the signature of graphs) are not context-free, but can be generated by the following rules from the initial graph  $\bullet \xrightarrow{X} \bullet$ .



The HR-algebra of finite and countable graphs corresponding to hyperedge-replacement grammars is a many-sorted algebra defined as follows.

For each  $n$  there is a separate sort  $\mathbb{G}_n$  of graphs with  $n$  *sources*. These are distinguished vertices, though not necessarily distinct, named  $v_1, \dots, v_n$ . There are constants of each sort  $\mathbb{G}_n$ : these are hypergraphs having at most one hyperedge, exactly  $n$  vertices, each vertex a distinct source. The HR-algebra is built on the following operations: *disjoint union*  $\oplus$ , *renaming of sources*  $\text{rename}_{c \leftrightarrow c'}$ , and *fusion of sources*  $\text{fuse}_{\approx}$  according to an equivalence  $\approx$  on source names. By convention  $\oplus$  is understood to automatically shift the source names of its second argument by the maximum of the source names used in the first to avoid naming conflict. Also  $\text{fuse}$  assigns the least source name of a class to each fused node while dropping the others.

It is intuitively clear how a hyperedge-replacement step can be expressed using disjoint union with the right-hand side of the rule followed by a fusion and renaming of sources. Formally, one transforms an HR-grammar  $\mathcal{G}$  into a system of finitely many equations  $X_i = t_i(\bar{X})$  where variables play the role of non-terminals of the grammar and the terms  $t_i$  are chosen such that, when variables are interpreted as individual hyperedges,  $\llbracket t_i(\bar{X}) \rrbracket$  is the right hand side of the matching rule for a hyperedge labelled  $X_i$ .

**Example 1.2.8** The equation corresponding to the single rule of the HR grammar of Example 1.2.7 generating  $\mathfrak{T}_\omega$  is

$$X = \text{rename}_{0 \leftrightarrow 0, 1 \leftrightarrow 1}(\text{fuse}_{\{0,2\}, \{1,4\}}(\overset{0}{\bullet} \rightarrow \overset{1}{\bullet} \oplus X \oplus X)).$$

Note that the source names of the first and second occurrences of  $X$  are shifted by 2 and by 4, respectively, while forming their disjoint union. Thus, after fusion we obtain precisely the right hand side of the HR-rule generating  $\mathfrak{T}_\omega$ , however, with additional source names. The renaming operation in this term has the effect of forgetting the source names 2 and above. So the least solution of this equation is indeed  $\mathfrak{T}_\omega$  with its root labelled 0 and one of its children with 1.

The generating power of HR-grammars is limited by the fact that edges can only be ‘created’ via fusion of sources (after having taken the disjoint union of two graphs). Because there are only a fixed number of source names available in a finite HR-equational system there is a bound on the size of complete bipartite subgraphs  $K_{n,n}$  that can be created [12], cf. Theorem 1.2.12. The infinite bipartite graph  $K_{\omega,\omega}$  is thus an example of a prefix-recognisable graph which is not HR-equational.

It is a key observation that in case of HR-terms the evaluation mapping

$t \mapsto \llbracket t \rrbracket$  is expressible as an MSO-interpretation. In fact, since edges cannot be created by any of the HR operations, the vertex-edge-adjacency graph of  $\llbracket t \rrbracket$  is MSO-interpretable in the tree representation of  $t$ , whether  $t$  is finite or infinite.

**Theorem 1.2.9** *For a countable graph  $G$  the following are equivalent.*

- (1)  *$G$  is generated by a deterministic HR grammar;*
- (2)  *$G$  is HR-equational, i.e. the evaluation of a regular HR-term, i.e. the least solution of a finite system of HR-equations;*
- (3) *The two-sorted incidence graph  $\hat{G}$  of  $G$  is monadic second-order interpretable in the full binary tree, i.e.  $\hat{G} \leq_{\text{MSO}} \mathfrak{T}_2$ .*

For a detailed presentation of these and other algebraic frameworks and their connections to the generative approach based on graph grammars we advise consulting [55, 12, 21]. In [54] Courcelle considered an extension of monadic second-order logic, denoted CMSO<sub>2</sub>, in which one can quantify over sets of edges as well as over sets of vertices and, additionally, make use of modulo counting quantifiers. Notice that the last item of the previous theorem implies that the CMSO<sub>2</sub>-theory of equational graphs is interpretable in S2S and is thus decidable. Further, Courcelle proved that CMSO<sub>2</sub> is able to axiomatise each and every equational graph up to isomorphism.

**Theorem 1.2.10** *Each HR-equational graph is axiomatisable in CMSO<sub>2</sub>. Consequently the isomorphism problem of equational graphs is decidable.*

Sénizergues considered HR-equational graphs of finite out-degree and proved that they are, up to isomorphism, identical with the  $\varepsilon$ -closures of configuration graphs of normalised<sup>7</sup> pushdown automata restricted to the set of reachable configurations. Further, he proved that bisimulation equivalence of HR-equational graphs of finite out-degree is decidable [128]. This last result is an improvement on the decidability of bisimulation equivalence for deterministic context-free processes, which is a consequence of the celebrated result of Sénizergues establishing decidability of the DPDA language equivalence problem.

<sup>7</sup> Here a PDA is said to be normalised, if in addition to being in a familiar normal-form its  $\varepsilon$ -transitions may not push anything on the stack. Hence the finiteness bound on the out-degree of configurations. For precise definitions see [128].

### Vertex-replacement grammars

Vertex replacement systems are a finite collection of graph rewriting rules that allow one to substitute given finite graphs in place of single vertices while keeping all the connections. This form of graph rewriting emerged as the most robust and manageable from among a host of different notions within a very general framework [55, 69, 59, 58]. The corresponding VR-algebra of graphs is built on the following operations: constant graphs of a single  $c$ -coloured vertex  $\bullet^a$ , disjoint union  $\oplus$ , recolouring of vertices  $\text{recol}_{c \rightarrow c'}$  and introduction of  $a$ -coloured edges  $\text{edge}_{c \xrightarrow{a} d}$  from every  $c$ -coloured vertex to every  $d$ -coloured vertex.

The evaluation of VR-terms, whether finite or infinite, is realisable as a monadic second-order interpretation. More precisely, as VR-equational graphs are interpretations of regular terms obtained by unfolding a finite system of VR equations, they can be MSO-interpreted in a regular tree, hence also in the full binary tree  $\mathfrak{T}_2$ , and thus are prefix-recognisable. These and other characterisations, together with our previous discussion of prefix-recognisable structures are summarised in the next theorem.

**Theorem 1.2.11** *For a countable graph  $G$  the following are equivalent.*

- (1)  $G$  is isomorphic to a prefix-recognisable structure;
- (2)  $G$  is generated by a deterministic VR grammar;
- (3)  $G$  is VR-equational, i.e. the evaluation of a regular VR-term, i.e. the least solution of a finite system of equations of the form  $X_i = t_i(\overline{X})$  with finite VR-terms  $t_i(\overline{X})$ ;
- (4)  $G \leq_{\text{MSO}} \mathfrak{T}_2$ ;
- (5)  $G = h^{-1}(\mathfrak{T}_2)|_C$ , i.e. the vertices of  $G$  are obtained by restricting the nodes of  $\mathfrak{T}_2$  to a regular set  $C$ , and its edges are obtained by taking the inverse of a rational substitution  $h$  to  $\mathfrak{T}_2$ ;
- (6)  $G$  is isomorphic to the  $\epsilon$ -closure of the configuration graph of a push-down automaton.

Further, the HR-equational graphs can be characterised as the class of VR-equational graphs of finite tree width [11].

**Theorem 1.2.12** *VR-equational graphs of finite tree width are HR-equational.*

**Example 1.2.13** The complete bipartite graph  $K_{\omega,\omega}$  is a prominent example of a VR-equational graph that is not HR-equational. A VR grammar and the corresponding system of VR equations generating  $K_{\omega,\omega}$  are

given below.

$$\begin{array}{ll} \bullet^X \Rightarrow \overset{A}{\bullet \leftarrow \rightarrow \bullet} & X = \mathbf{edge}_{a \leftrightarrow b}(A \oplus \mathbf{recol}_{a \mapsto b}(A)) \\ \overset{A}{\bullet} \Rightarrow \bullet^A & A = \overset{a}{\bullet \oplus A} \end{array}$$

The expressive power of this formalism (for describing families of finite graphs) is not increased by extending the VR operations by graph transformations that are definable using quantifier-free formulas (of which  $\mathbf{recol}_{c \leftrightarrow c'}$  and  $\mathbf{edge}_{c \xrightarrow{a} d}$  are particular examples), nor by the *fusion* operations  $\mathbf{fuse}_c$  identifying all nodes bearing a certain colour  $c$  [60]. Care has to be taken when defining countable graphs as evaluations of infinite terms, for it is unclear how to deal with infinite terms built with non-monotonic operations. Nonetheless, infinite terms built with operations definable by *positive* quantifier-free formulas can be evaluated unambiguously [11].

In this setting Theorem 1.2.11 can be generalised to infinite systems of equations (whose unfoldings are typically non-regular terms) using infinite deterministic automata [33], leading us to the following families of transition graphs.

### 1.2.3 Higher-order data structures

#### Tree-constructible graphs and Caucal's pushdown hierarchy

Courcelle introduced MSO-*compatible transductions* in the investigation of structures with decidable monadic theories. Let  $\mathcal{C}$  and  $\mathcal{C}'$  be classes of structures on signatures  $\sigma$  and  $\sigma'$ , respectively. Following [57] we say that a functional transduction  $T : \mathcal{C} \rightarrow \mathcal{C}'$  is MSO-compatible if there is an algorithm mapping each monadic formula  $\varphi$  of signature  $\sigma'$  to a monadic formula  $\varphi^T$  in the signature  $\sigma$  such that

$$\mathfrak{A} \models \varphi^T \iff T(\mathfrak{A}) \models \varphi .$$

MSO-interpretations are the most natural examples of MSO-compatible transductions. Slightly more generally, the MSO-*definable transductions* of Courcelle are MSO-compatible. Recall that these are given by a  $k$ -copying operation (for some  $k$ ) followed by an MSO-interpretation and in particular the resulting structure may have  $k$  times the cardinality of the original one.

The more difficult result that the *unfolding* operation, mapping graphs  $(\mathfrak{G}, v)$  to trees  $\mathfrak{T}_{(\mathfrak{G}, v)}$ , is also MSO-compatible appeared in [61] (see also [57] for an exposition and a treatment of the simpler case of deterministic

graphs). We note that this result also follows from Muchnik's Theorem [126, 138, 17] and that it generalises Rabin's theorem.

A rich class of graphs, each with decidable monadic theory, can now be constructed. Caucal [43] proposed the hierarchies of graphs and trees obtained by alternately applying unfoldings and MSO-interpretations starting with finite graphs:

**Definition 1.2.14**

$$\begin{aligned}\text{Graphs}_0 &= \{\text{finite edge- and vertex-labelled graphs}\} \\ \text{Trees}_{n+1} &= \{\mathfrak{T}_{\mathfrak{G}, v} \mid (\mathfrak{G}, v) \in \text{Graphs}_n\} \\ \text{Graphs}_{n+1} &= \{\mathcal{I}(\mathfrak{T}) \mid \mathfrak{T} \in \text{Trees}_{n+1}, \mathcal{I} \text{ is an MSO interpretation}\}\end{aligned}$$

By the results above, we have

**Theorem 1.2.15** *For every  $n \in \mathbb{N}$  every graph  $G$  from  $\text{Graphs}_n$  has a decidable MSO-theory.*

Fratani [72, 73] provided an alternative proof of the above theorem, among a host of other results on higher-order pushdown graphs, using a different kind of MSO-compatible operation. Indeed, she established that if a homomorphism of words maps the branches of a tree  $T$  to those of  $T'$  surjectively while also preserving the node-labeling then definability and decidability results for MSO over  $T'$  can be transferred to  $T$ .

The Caucal hierarchy is very robust. Various weakenings and strengthenings of the definition yield exactly the same classes [37]. In fact, in place of MSO-interpretations, Caucal originally used inverse rational mappings in the style of item (5) of Theorem 1.2.11. Recently Colcombet [51] proved that every graph of  $\text{Graphs}_{n+1}$  can in fact be obtained via a first-order interpretation in some tree belonging to  $\text{Trees}_{n+1}$ . The next theorem provides internal presentations of graphs of each level as a generalisation of Theorem 1.2.11 item (6) thereby justifying the name pushdown hierarchy.

**Theorem 1.2.16 ([37])** *For every  $n$  a graph  $G$  is in  $\text{Graphs}_n$  if, and only if, it is isomorphic to the  $\epsilon$ -closure of the configuration graph of a higher-order pushdown automaton at level  $n$ .*

The strictness of the hierarchy was also shown in [37]. The level-zero graphs are the finite graphs, trees at level one are the regular trees, and as we have seen in Theorem 1.2.11 the level-one graphs are the prefix-recognisable ones. The deterministic level-two trees are known as

algebraic trees. From the second level onwards we have no clear structural understanding of the kind of graphs that inhabit the individual levels. We recommend [134] for an exposition.

### Term-trees defined by recursion schemes

Caucal also gave a kind of algebraic characterisation of term-trees at level  $n$  as fixed points of *safe* higher-order recursion schemes.

**Theorem 1.2.17** ([43]) *For every  $n$ , the class of term-trees  $\text{Trees}_n$  coincides with that of term-trees generated by safe higher-order recursion schemes of level at most  $n$ .*

The notion of higher-order schemes is a classical one [62, 56]. Safety is a technical restriction (implicit in [62]) ensuring that no renaming of variables ( $\alpha$ -conversion) is needed during the generative substitutive reduction ( $\beta$ -reduction) process constructing the solution-term [1, 117]. Safe schemes are intimately related to the pushdown hierarchy. This connection is well explained in [1] showing that while on the one hand order- $n$  schemes can define the behaviour and hence (the unfolding of) the configuration graphs of level- $n$  deterministic pushdown automata, on the other hand, deterministic pushdown automata of level  $n$  can evaluate safe order- $n$  schemes. Safety is hereto essential.

In order to evaluate arbitrary schemes [81] introduced *higher-order collapsible pushdown automata* (CPDA), a kind of generalisation of panic automata [92], and gave in essence the following characterisation in the spirit of Theorem 1.2.16.

**Theorem 1.2.18** *The term-trees defined by order- $n$  recursion schemes are up to isomorphism identical with the unfoldings of  $\epsilon$ -closures of configuration graphs of level- $n$  collapsible higher-order pushdown automata.*

As shown in [117, 81], it is not necessary to assume safety for establishing decidability of the MSO-theories of term-trees that are solutions of higher-order schemes.

**Theorem 1.2.19** *The MSO-theory of a term-tree defined by an arbitrary higher-order recursion scheme is decidable.*

Consequently, configuration graphs of higher-order collapsible pushdown automata can be model-checked against modal  $\mu$ -calculus formulas. However, there is a second-order CPDA whose configuration graph interprets the infinite grid and whose MSO-theory is thus undecidable

[81]. This shows that higher-order CPDA configuration graphs constitute a proper extension of Caucal's pushdown hierarchy.

### 1.2.4 Introducing products

There is a connection between the internal presentations of graphs seen so far and the graph operations used in the corresponding equational framework. Pushdown stacks are naturally represented as strings. The set of strings over some alphabet can in turn be modelled as an algebra of terms built with unary functions, one for each letter of the alphabet. Strings thus correspond to terms and letters to unary functions. In functional programming terminology the abstract data type of, say, binary strings has the recursive type definition

$$\mathcal{T} = \perp \oplus 0(\mathcal{T}) \oplus 1(\mathcal{T}) \quad (1.2)$$

Here the letters 0 and 1 are seen as type constructors and the empty string  $\perp$  is a constant type constructor. The set of finite strings is the least fixed-point solution of this equation.

Automata operating on terms of type  $\mathcal{T}$  can be viewed as functions mapping terms to states. Moreover these functions are defined according to structural recursion. Analogously, recursion schemes (fix-point equations) in an algebra of graph operations transform automata-based internal presentations of a graph into equational specifications. We can use the recursion scheme associated to the type definition (1.2) to define any PR-graph by a VR equation extending the type definition. For instance, the graph of the lexicographic order from Example 1.2.3 satisfies the following equation

$$L = \text{edge}_{0 \rightarrow 1, \varepsilon \rightarrow 0, \varepsilon \rightarrow 1}(\bullet^\varepsilon \oplus \text{recol}_{0,1,\varepsilon \mapsto 0}(L) \oplus \text{recol}_{0,1,\varepsilon \mapsto 1}(L)).$$

We briefly explain how to go from automata presenting a PR-graph to a VR-equation. For a language  $V \subset \{0,1\}^*$  recognised by an automaton with transition table  $\Delta \subset Q \times \Sigma \times Q$  and final states  $F$  the following VR-equation colours each word  $w \in \{0,1\}^*$  by those states  $q$  such that the automaton starting from  $q$  accepts  $w$ . (N.B. in accordance with (1.2) the simulation proceeds right-to-left.)

$$X = \bullet^F \oplus \text{recol}_{\{q' \mapsto q : \Delta(q,0,q')\}}(X) \oplus \text{recol}_{\{q' \mapsto q : \Delta(q,1,q')\}}(X)$$

In general, every PR-graph  $\bigcup_i U_i \cdot (V_i \times W_i)$  is the recolouring of a graph satisfying a VR-equation of the form

$$X = \vartheta(\vartheta_\varepsilon(\bullet) \oplus \vartheta_0(X) \oplus \vartheta_1(X)) . \quad (1.3)$$

Here, the states of the automata recognising  $V_i$  or  $W_i$  are encoded as vertex colours (just as above) and  $\vartheta_\varepsilon$  colours  $\bullet$  by the final states of the  $V_i$ 's and  $W_i$ 's. Edge colours are used to represent states of automata for each  $U_i$ . For every  $v \in V_i$  and  $w \in W_i$ , and  $z$  accepted by the automaton for  $U_i$  from state  $q$  there is a  $q$ -coloured edge  $(zv, zw)$ . To this end,  $\vartheta_0$  and  $\vartheta_1$  recolour the vertices and edges, and  $\vartheta$  adds an edge between all  $x \in V_i$  and  $y \in W_i$  coloured by the final states of  $U_i$ .

In passing we mention that higher-order stacks can also be represented as strings: either as well-bracketed sequences of stack symbols, or as strings of stack operations yielding the particular stack configuration. The former comes at the cost of losing regularity of the domain and has no apparent algebraic counterpart. The latter gives rise to a unary algebra of higher-order stacks that is not, except for level 1 pushdown stacks, freely generated by the stack operations. Thus there is no unique term representing a general stack. The work of Fratani, Carayol and others [72, 73, 33, 32] has shown that both of these deficiencies can be turned into features.

We now turn to graphs internally presented by finite trees. A type definition for  $\{0, 1\}$ -labelled binary branching trees is

$$\mathcal{T} = \perp \oplus 0(\mathcal{T} \otimes \mathcal{T}) \oplus 1(\mathcal{T} \otimes \mathcal{T}) \quad (1.4)$$

where  $\otimes$  denotes direct product. Later we will compare this with another type definition (1.6). Colcombet observed that this schema can be used to define graphs with internal presentations involving tree automata operating on finite trees. He proposed extensions of the VR-algebraic framework by the *asynchronous product*  $\otimes_A$  [48] and by the *synchronous product*  $\otimes_S$  [50, 49] which we shall denote here by VRA and VRS, respectively.

**Definition 1.2.20** (Synchronous and asynchronous product) The products are defined for vertex and edge-coloured graphs  $\mathcal{G}$  and  $\mathcal{H}$  as follows. In the synchronous product there is a  $d$ -coloured edge from  $(g, h)$  to  $(g', h')$  if, and only if, both  $(g, g')$  and  $(h, h')$  are connected by a  $d$ -edge in  $\mathcal{G}$  and  $\mathcal{H}$ , respectively. The edge relation  $E_d$  of the asynchronous product  $\mathcal{G} \otimes_A \mathcal{H}$  is defined as the union of  $\{((g, h), (g', h')) \mid E_d^{\mathcal{G}}(g, g'), h \in H\}$  and  $\{((g, h), (g, h')) \mid E_d^{\mathcal{H}}(h, h'), g \in G\}$ . The definition of vertex colours requires a little care. In both cases a vertex  $(g, h)$  of the product has colour  $\delta(c, c')$  whenever  $g$  has colour  $c$  and  $h$  has colour  $c'$ . Here the function  $\delta : C^2 \rightarrow C$  is a parameter of the product operation. However,

it is really only relevant that  $\delta$  acts as a pairing function on some sufficiently large subsets of the colours. For instance, Colcombet identifies  $C$  with  $\{0, 1, \dots, N - 1\}$  and defines  $\delta$  as addition modulo  $N$  [48].

As before, VRA-equational and VRS-equational graphs are defined as least fixed-point solutions of a finite system of equations in the respective algebra. Both product operations are cocontinuous with respect to graph embeddings. Therefore the evaluation mapping of both VRA and VRS terms uniquely extends from finite terms to infinite terms. Hence, just as for HR- and VR-equational graphs, the solution of a system of VRA or VRS equations is the evaluation of the regular term obtained by unraveling the system of equations.

**Example 1.2.21** The infinite two-dimensional grid  $(\mathbb{N} \times \mathbb{N}, \text{Up}, \text{Right})$  is easily constructed as the asynchronous product of the VR-equational, even context-free, graphs  $(\mathbb{N}, \text{Up})$  and  $(\mathbb{N}, \text{Right})$ :

$$\begin{aligned} G &= \otimes_A(N_u, N_r) \\ N_u &= \text{edge}_{a \xrightarrow{\text{Up}} b} \left( \bullet \oplus \text{recol}_{a \mapsto b, b \mapsto c}(N_u) \right) \\ N_r &= \text{edge}_{a \xrightarrow{\text{Right}} b} \left( \bullet \oplus \text{recol}_{a \mapsto b, b \mapsto c}(N_r) \right) \end{aligned}$$

The unfolding of this system of equations is, schematically, an infinite term consisting of two periodic branches joined at the root. Elements of the grid  $G$ , by definition of asynchronous product, are represented as pairs of nodes of this term-tree with one node on either branch, corresponding to the respective co-ordinates. The example of the grid, whose MSO theory is undecidable, shows that the evaluation mapping of VRA terms (also of VRS terms) can not be realised by an MSO-interpretation.

For any VRA or VRS-term  $t$ , vertices of  $\llbracket t \rrbracket$  can be identified with maximal *subsets* of nodes of  $t$  belonging to sub-terms joined by a product operator. It is thus easily expressible in MSO whether a set  $X$  of nodes (finite or infinite<sup>8</sup>) is actually well-formed in this sense, i.e. whether it represents an element of  $\llbracket t \rrbracket$ .

#### VR with asynchronous product and ground term rewriting

*Ground term rewrite systems* (GTRSs) are a natural generalisation of prefix-rewriting to trees. They are term rewrite systems given by rewrit-

<sup>8</sup> In least fixed-point semantics only finite sets are considered, whereas in greatest fixed-point semantics both finite and infinite sets can represent elements of the solution, provided that there is an infinite nesting of product operators in  $t$ .

ing rules in which no variables occur. Tree automata are a special case of GTRSs (see [52]).

**Example 1.2.22** The rewrite rule  $a \rightarrow f(a)$  confined to terms of the form  $d(f^n(a), f^m(a))$  is a GTRS whose configuration graph is isomorphic to the infinite square grid.

We have noted that prefix-recognisable graphs are identical to  $\varepsilon$ -closures of pushdown graphs. This correspondence is achieved by generalising the simple prefix-rewriting rules of pushdown systems of the form  $v \rightarrow w$  where  $v$  and  $w$  are strings to replacement rules  $V \rightarrow W$  for given regular languages  $V, W$ . The latter rule allows one to rewrite any prefix  $v \in V$  of a given string by any word from  $W$ . Regular Ground Term Rewrite Systems (RGTRS) generalise GTRS in the exact same manner: simple ground rewrite rules  $s \rightarrow t$  with ground terms  $s, t$  are replaced by ‘rule schemes’  $S \rightarrow T$  with regular sets of terms on both left and right-hand side.

Löding [99, 100] and Colcombet [48] studied transition graphs of GTRSs and RGTRSs from a model-checking point of view. In Löding’s work vertices of the transition graph are those terms reachable from an initial term, whereas Colcombet considers all terms of a given type as vertices.

The VR-equations defining PR graphs (1.3) easily generalise to VRA-equations defining graphs of RGTRSs using the recursion scheme (1.4):

$$X = \vartheta(\vartheta_\varepsilon(\bullet) \oplus \vartheta_0(X \otimes_A X) \oplus \vartheta_1(X \otimes_A X)) \quad (1.5)$$

For each rule  $S_i \rightarrow T_i$  of the RGTRS we simulate (frontier to root) tree automata recognising  $S_i$  and  $T_i$ . Vertices of  $X$  represent terms, so we call these vertex-terms. A vertex-term is coloured by those states  $q$  occurring at the root of the term after being processed by the automata. The simulation is initialised as follows:  $\vartheta_\varepsilon$  labels  $\bullet$  by initial states, and  $\vartheta$  adds edges between all vertex-terms coloured by accepting states of automata for  $S_i$  and  $T_i$ . Updates occur in  $\vartheta_j$ s according to the transition rules, similarly to (1.3). To this end assume that two vertex-terms  $v', v''$  are coloured by states  $q'$  and  $q''$  respectively. After taking the product the paired vertex-term  $j(v', v'')$  is initialised with colour  $(q', q'')$  (cf. Def. 1.2.20). This pair is then recoloured to  $q$  by  $\vartheta_j$  whenever  $(q, j, q', q'')$  is a transition.

Notice how naturally the asynchronous product captures closure of RGTR rewriting under contexts: if there was an edge between  $v$  and  $v'$  then there is an edge between  $j(v, v'')$  and  $j(v', v'')$ , and, symmetrically,

between  $j(v'', v)$  and  $j(v'', v')$ . One obtains along these lines the following generalisations of Theorem 1.2.11 (cf. examples 1.2.22 and 1.2.21).

**Theorem 1.2.23** (Colcombet [48])

- (i) *A countable graph is VRA-equational if, and only if, it is (after removal of certain colours) isomorphic to an RGTRS graph<sup>9</sup>.*
- (ii) *Each VRA-equational graph is finite-subset interpretable in a regular term-tree, hence also in the full binary tree.*

Theorem 1.2.12 also extends to VRA-equational graphs [48, 100].

**Theorem 1.2.24** *VRA-equational graphs of finite tree-width are HR-equational.*

An immediate consequence of Theorem 1.2.23 is that the FO-theory of every VRA-equational structure is decidable via interpretation in S2S. In fact, for any VRA-equational graph  $G = (V, \{E_a\}_a)$  the subset interpretation, hence also first-order decidability, extends to  $G$  with additional reachability predicates  $R_C = \{(v, w) \mid w \text{ can be reached from } v \text{ using edges of colours from } C\}$  for arbitrary subsets  $C$  of edge colours [48].

**Theorem 1.2.25** *VRA-equational graphs have a decidable first-order theory with reachability.*

This result cannot be improved much further. Examples of [139] show that ‘regular reachability’, i.e. the problem whether there exists a path in a given VRA-equational graph between two given nodes and such that the labeling of the path belongs to a given regular language over the set of colours, is undecidable. In [100] Löding identified a maximal fragment of CTL that is decidable on every GTRS graph (with vertices restricted to terms reachable from an initial one) that can express, besides reachability, recurring reachability.

#### VR with synchronous product and tree-automatic structures

We have remarked that in the subset interpretation of VRA terms the subsets are used in a special form. Indeed, in the evaluating interpretation they merely serve the purpose of outlining the shape of a finite term. General finite-subset interpretations are more powerful and are capable of expressing the evaluation of VRS terms. In fact, these two formalism are equally expressive.

<sup>9</sup> Here RGTRS graphs are taken in the sense of [48] as being restricted to the set of terms of a given type.

This is best explained by *tree-automatic presentations*. These are internal presentations of VRS-structures which will be formally introduced in the next section. For now it suffices to use the characterisation (Theorem 1.3.18) that tree-automatic graphs are those that are wMSO-interpretable in a regular tree (reflected in the equivalence of (1) and (2) below).

**Theorem 1.2.26** (Colcombet [50])

For every countable graph  $G$  the following are equivalent

- (1)  $G$  is isomorphic to a tree-automatic graph.
- (2)  $G$  is interpretable in a regular tree (wlog. the full binary tree) via a finite-subset interpretation.
- (3)  $G$  is the restriction of a VRS-equational vertex-labelled graph  $G'$  to its set of vertices of a given colour;

We have noted that the evaluation mapping of VRS-terms can be naturally defined as a finite subset interpretation - this justifies (3)  $\rightarrow$  (2). Continuing our discussion of translations from automata-based internal presentations into equational specifications using graph products we illustrate the remaining translation (2)  $\rightarrow$  (3) from finite-tree automatic to VRS-equational presentations on graphs as we did for PR and RGTRS. That is, we build the terms of the presentation from the bottom up while also simulating the automata constituting the tree-automatic presentation by VRS-operations.

Start with a graph  $(V, E)$  that is definable via finite-subset interpretation in the full binary tree. By the fundamental correspondence that wMSO-definable relations in a regular tree are exactly those that are recognised by tree automata operating on finite trees, we see that  $V$  may be taken to be a regular set of finite  $\Sigma$ -labelled binary trees, and  $E$  is recognised by an automaton  $\mathcal{A}$  accepting pairs of such trees.

The tree automaton  $\mathcal{A}$  has transition rules (here we read them from left-to-right, i.e. in top-down fashion, but that is a matter of choice and the simulation will actually proceed from bottom up) of the form

$$r : (q, \langle a, b \rangle, q_0, q_1) \quad \text{with } a, b \in \{0, 1, \square\}$$

where the symbol  $\square$  is necessary for padding either components of a pair of trees so that they have the same shape. It indicates the fact that no node is defined in the current position, i.e. that the automaton finds itself below a leaf of the respective tree (while still reading the other).

We may assume that the transition rules enforce a proper usage of the padding symbols.

We introduce edge relations  $E_q$  and  $E_r$  for each state  $q$  and each rule  $r$  of the automaton. The simulation of transitions of the synchronous automaton on *pairs of labelled trees* necessitates a more sophisticated recursion scheme associated to the following type definition of  $\{0, 1\}$ -labelled binary branching trees.

$$\mathcal{T} = \perp \oplus (\{0, 1\} \otimes \mathcal{T} \otimes \mathcal{T}) \quad (1.6)$$

There is a natural identification of terms of this type and of those of the more natural type definition (1.4). As far as unary predicates are concerned the current type definition does not provide any advantage. However, compared with (1.4) the current type definition has a more powerful associated recursion scheme allowing for defining non-trivial *binary* relations between terms with different root labels. This will allow us to specify tree-automatic graphs via VRS-equations of the following form analogous to (1.6)

$$X = \vartheta(\bullet^\perp \oplus (\vartheta_0 \otimes_S \vartheta_1(X) \otimes_S \vartheta_2(X))) \quad (1.7)$$

Here too, as in (1.3) and in (1.5) the  $\vartheta$ 's are VR-expressions facilitating the simulation of the automaton. The expression  $\vartheta_0$  specifies the graph with vertex set  $\{0, 1\}$  and having an  $r$ -labelled edge from  $a$  to  $b$  for each rule  $r$  such that  $r = (\cdot, \langle a, b \rangle, \cdot, \cdot)$  and with VR operations (here equivalently expressed as positive quantifier-free definable operations) responsible for updating the edge relations to simulate the transitions of  $\mathcal{A}$ . This is done in two phases.

- First, in preparation, state-labelled edges are used to ‘enable’ compatible rule-labelled edges in either copy of the graph: for each rule  $r = (\cdot, \langle \cdot, \cdot \rangle, q_1, q_2)$  and  $i \in \{1, 2\}$  the expression  $\vartheta_i$  adds an  $E_r$ -edge from  $x$  to  $y$  for every  $E_{q_i}$ -edge from  $x$  to  $y$  in the graph.
- Then, after the synchronous product of rule-labelled edges has been taken, edges labelled by rules are renamed to their resulting states:  $\vartheta$  adds for each state  $q$  an  $E_q$ -edge from  $x$  to  $y$  for every  $E_r$ -edge from  $x$  to  $y$  such that  $r = (q, \langle \cdot, \cdot \rangle, \cdot, \cdot)$ . In addition,  $\vartheta$  deals with the case when either  $x$  or  $y$  is the singleton tree  $\perp$ . For this we may assume that all necessary information is coded in vertex labels implemented as reflexive edges and maintained along with the rest of the edge labels as explained here.

Finally, to obtain the graph  $G'$  as required in item (3) of Theorem 1.2.26

we also use vertex colours to keep track of the states of the tree automaton recognising  $V$ . The generalisation of this construction to arbitrary relational structures is straightforward.

## 1.3 Automatic Structures

### 1.3.1 Fundamentals

This section concerns structures with internal presentations consisting of automata operating synchronously on their inputs. The starting point of this investigation is the robust nature of finite automata. In particular, synchronous automata are effectively closed under certain operations that can be viewed in logical terms, i.e. Boolean operations, projection, cylindrification and permutation of arguments. Thus a structure whose domain and atomic operations are computable by such automata has decidable first-order theory (Definition 1.3.2 and Theorem 1.3.4).

**Example 1.3.1** (i) The domain and relations of the following structure are regular.

$$\mathcal{S}_\Sigma = (\Sigma^*, \{\mathbf{suc}_a\}_{a \in \Sigma}, \prec_{\text{prefix}}, \mathbf{e1})$$

where  $\Sigma^*$  is the set of finite words over alphabet  $\Sigma$ , the binary relation  $\mathbf{suc}_a$  is the successor relation  $(x, xa)$  for  $x \in \Sigma^*$ , the binary relation  $\prec_{\text{prefix}}$  is the prefix relation and the binary relation  $\mathbf{e1}$  is the equal-length relation.

(ii) The following structure can be coded (eg. in base  $k$  least significant digit first) so that the domain and atomic operations are regular.

$$\mathcal{N}_k = (\mathbb{N}, +, |_k)$$

where  $+$  is the usual addition on natural numbers and  $x |_k y$  holds precisely when  $x$  is a power of  $k$  and  $x$  divides  $y$ .

Actually the link between synchronous automata and logic goes both ways. It was first expressed in terms of weak monadic second-order logic: a set of tuples  $(A_1, \dots, A_n)$  of finite sets of natural numbers is weak monadic second-order definable in  $(\mathbb{N}, S)$  if and only if the corresponding  $n$ -ary relation of characteristic strings (a subset of  $(\{0, 1\}^*)^n$ ) is synchronous rational. This was proved by [27] and [68], and is implicit in [135].

A first-order characterisation was provided by [65]: a relation  $R \subset (\Sigma^*)^n$  is synchronous rational if and only if  $R$  is first-order definable

in  $\mathcal{S}_\Sigma$  for  $|\Sigma| \geq 2$ . Similarly, the Büchi-Bruyère Theorem states that a relation  $R \subset \mathbb{N}^n$  (coded in base  $k \geq 2$  least significant digit first) is synchronous rational if and only if it is first-order definable in  $\mathcal{N}_k$  (proofs of which can be found in [104] and [137]).

These results were generalised to full MSO on the line  $(\mathbb{N}, S)$  and weak MSO and full MSO on the tree  $(\{0, 1\}^*, \text{suc}_0, \text{suc}_1)$  and form the basis of the logical characterisation of automatic structures (Section 1.3.4). However, we start with the more common internal definition.

Recall that the four basic types of automata operate on finite or infinite words or trees. So, let  $\square$  be one of **word**,  **$\omega$ -word**, **tree**,  **$\omega$ -tree**.

We consider a structure  $\mathfrak{B} = (B, \{R_i\})$  comprising relations  $R_i$  over the domain  $\text{dom}(\mathfrak{B}) = B$ . Thus constants and operations are implicitly replaced by their graphs.

**Definition 1.3.2** (Automatic presentation) A  $\square$ -automatic presentation of  $\mathfrak{B}$  consists of a tuple  $\mathfrak{d} = (\mathcal{A}, \mathcal{A}_\approx, \{\mathcal{A}_i\})$  of finite synchronous  $\square$ -automata and a *naming function*  $f : \mathcal{L}(\mathcal{A}) \rightarrow B$  such that

- Each  $\mathcal{L}(\mathcal{A}_i)$  is a relation on the set  $\mathcal{L}(\mathcal{A})$ .
- $\mathcal{L}(\mathcal{A}_\approx)$  is a congruence relation on the structure  $(\mathcal{L}(\mathcal{A}), \{\mathcal{L}(\mathcal{A}_i)\}_i)$ .
- The quotient structure is isomorphic to  $\mathfrak{B}$  via  $f$ .

Moreover, the quotient structure is called an *automatic copy* of  $\mathfrak{B}$ . We say that the presentation is *injective* whenever  $f$  is, in which case  $\mathcal{A}_\approx$  can be omitted.

**Definition 1.3.3** (Automatic structure<sup>10</sup>) A structure  $\mathfrak{B}$  is  $\square$ -*automatic* if it has an  $\square$ -automatic presentation. If  $\mathfrak{B}$  is  $\square$ -automatic for some  $\square$  then  $\mathfrak{B}$  is simply called *automatic*. The classes of automatic structures are respectively denoted by **S-AutStr**,  **$\omega$ S-AutStr**, **T-AutStr** and  **$\omega$ T-AutStr**.

The following theorem motivates the study of automatic structures and so may be called the *Fundamental Theorem* of automatic structures/presentations.

**Theorem 1.3.4** (Definability) *There is an algorithm that given a  $\square$ -automatic presentation  $(\mathfrak{d}, f)$  of a structure  $\mathfrak{A}$  and a FO-formula  $\varphi(\bar{x})$  in the signature of  $\mathfrak{A}$  defining a  $k$ -ary relation  $R$  over  $\mathfrak{A}$ , effectively constructs a synchronous  $\square$ -automaton recognising  $f^{-1}(R)$ .*

Immediate corollaries are

<sup>10</sup> Some authors write *automatically presentable*.

- (i) *Decidability:* The FO-theory of every automatic structure is decidable.
- (ii) *Interpretations:* The class of  $\square$ -automatic structures is closed under FO-interpretations.

We point out that the Fundamental Theorem implies that every relation first-order definable from  $\square$ -regular relations is itself  $\square$ -regular.

*Remark 1.3.5* One may allow finitely many parameters  $\varphi(\bar{a}, \bar{x})$  under the following conditions. For finite-word and finite-tree presentations any parameters can be used. However, for  $\omega$ -tree (and  $\omega$ -word) presentations a parameter  $a$  can be used if  $f^{-1}(a)$  contains a regular  $\omega$ -tree (ultimately periodic  $\omega$ -word).

Consequently  $\square$ -automatic structures (on a given signature) are closed with respect to operations such as disjoint union, ordered sum and direct product – each a special case of generalised products treated in [20, 23]. However **AutStr** and  $\omega$ **S-AutStr** are not closed under weak direct-power. For instance,  $(\mathbb{N}, +)$  is in **S-AutStr** but its weak direct-power is isomorphic to  $(\mathbb{N}, \times)$ , which is not in **S-AutStr** (see [20]). On the other hand, it is straightforward to see that **T-AutStr** and  $\omega$ **T-AutStr** are closed under weak direct-power.

### 1.3.2 Examples

Obviously every finite structure is automatic. Here are some examples of structures with automatic presentations.

- Example 1.3.6** (Ordinals)
- (i)  $(\omega, <) \in \mathbf{S-AutStr}$ : The simplest automatic copy is the unary one:  $(0^*, \{(0^k, 0^l) \mid k < l\})$ .
  - (ii) Every ordinal below  $\omega^\omega$  is in **S-AutStr**: An automatic copy of  $\omega^k$  is  $((0^*1)^k, <_{\text{lex}})$  where  $<_{\text{lex}}$  denotes the lexicographic order<sup>11</sup> which is clearly regular. In this presentation the naming function is

$$0^{n_{k-1}}1\dots0^{n_0}1 \mapsto n_{k-1}\omega^{k-1} + \dots + n_1\omega^1 + n_0 .$$

- (iii) Every ordinal below  $\omega^{\omega^\omega}$  is in **T-AutStr**: recall that the ordinal  $\omega^\alpha$  has a representation as the set of functions  $f : \alpha \rightarrow \omega$  with  $f$  equal to 0 in all but finitely many places. These functions are ordered as follows:  $f < g$  if the largest  $\beta$  with  $f(\beta) \neq g(\beta)$  has that  $f(\beta) < g(\beta)$ . Then for fixed  $k$ , a function  $f : \omega^k \rightarrow \omega$  is coded by the tree  $T_f$  with

<sup>11</sup> Given an ordering on the symbols of the alphabet a word  $u$  is lexicographically smaller than  $w$  if either  $u$  is a proper prefix of  $w$  or if in the first position where  $u$  and  $w$  differ there is a smaller symbol in  $u$  than in  $w$ .

domain a finite subset of  $0^*1^*2^*\cdots k^*$  so that for every  $\beta$ , expressed in Cantor-normal-form as  $\omega^{k-1}c_0 + \omega^{k-2}c_1 + \cdots + \omega^0c_{k-1}$ ,  $0 \leq c_i < \omega$ , we have  $T_f(0^{c_0}1^{c_1}\cdots(k-1)^{c_{k-1}}k^{f(\beta)}) = 1$ .

- Example 1.3.7** (Orderings) (i)  $(\mathbb{Q}, <) \in \mathbf{S}\text{-AutStr}$ : The countable linear order  $(\{0, 1\}^*1, <_{\text{lex}})$  is dense without endpoints.  
(ii)  $(\mathbb{R}, <) \in \omega\mathbf{S}\text{-AutStr}$ .

**Example 1.3.8** (Groups) (i) Every finitely-generated group with an Abelian group of finite index is in  $\mathbf{S}\text{-AutStr}$ . And these are the only finitely generated word-automatic groups [116].

- (ii) The direct sum of countably many copies of  $\mathbb{Z}/m\mathbb{Z}$  is in  $\mathbf{S}\text{-AutStr}$ .
- (iii) The subgroup  $\mathbb{Z}[1/k]$  of rationals of the form  $\{zk^{-i} \mid z \in \mathbb{Z}, i \in \mathbb{N}\}$  for fixed  $k \in \mathbb{N}$  is in  $\mathbf{S}\text{-AutStr}$ .
- (iv) The Prüfer  $p$ -group  $\mathbb{Z}(p^\infty) = \mathbb{Z}[1/p]/\mathbb{Z}$  (prime  $p$ ) is in  $\mathbf{S}\text{-AutStr}$  [114].
- (v) Real addition  $(\mathbb{R}, +)$  is in  $\omega\mathbf{S}\text{-AutStr}$ .

However, the additive group of the rationals  $(\mathbb{Q}, +)$  is not automatic [136]. In fact, Tsankov shows that no torsion free Abelian group that is  $p$ -divisible for infinitely many primes  $p$  is automatic.

- Example 1.3.9** (Arithmetics) (i)  $(\mathbb{N}, +)$  is in  $\mathbf{S}\text{-AutStr}$ : For every natural  $k > 1$ , the base  $k$  least-significant-digit-first presentation of naturals (with or without leading zeros) constitutes a naming function of an automatic presentation. A finite automaton can perform the school-book addition method while keeping track of the carry in its state. Such a presentation is injective when leading zeros are suppressed.  
(ii)  $(\mathbb{N}, \cdot)$  is in  $\mathbf{T}\text{-AutStr}$ : The presentation is based on the unique factorisation of every natural number  $n$  into prime powers  $2^{n_2}3^{n_3}\cdots p^{n_p}$ . Each  $n_k$  is written, say in binary notation, on a single branch of a tree with domain  $0^*1^*$ . Multiplication is reduced to the addition of corresponding exponents. This construction can naturally be generalised to give tree-automatic presentations of weak direct powers of word-automatic structures [20, 25].

**Example 1.3.10** (Equivalence relations) The following have finite-word automatic presentations.

- (i) There is one class of size  $n$  for every  $n \in \mathbb{N}$ .
- (ii) There are  $d(n)$  classes of size  $n \in \mathbb{N}$  where  $d(n)$  is the number of divisors of  $n$ . (This is the direct product of the previous equivalence relation with itself).

**Example 1.3.11** (Free algebras) (i) The free algebra with  $n$  unary operations and at most  $\omega$  many constants is in **S-AutStr**.

- (ii) The free monoid generated by a single constant is in **S-AutStr**. However, no non-unary free or even free-associative algebra on two or more constants is in **S-AutStr**.
- (iii) The free algebra generated by countably many constants and any finite number of operations is in **T-AutStr**.<sup>12</sup> For instance suppose there is one binary operation  $F$ . The domain of the presentation consists of all  $\{F, c, \perp\}$ -labelled binary trees. The operation (representing  $F$ ) takes trees  $S$  and  $T$  as input and returns the tree with domain the prefix-closure of  $(\text{dom}(S) \cup \text{dom}(T))\{0, 1\}$  and taking the following values: the root position is labelled  $F$ ; position  $\alpha 0$  is labelled by the label of  $S$  at position  $\alpha$ ; position  $\alpha 1$  by the label of  $T$  at position  $\alpha$  (if either of these latter positions does not exist, the label is  $\perp$ ). It is not known whether finitely generated (non-unary) term algebras are in **T-AutStr**.

**Example 1.3.12** (Boolean Algebras) The signature we work in consists of the symbols for boolean operations  $\cap, \cup, \cdot^c$  and constants  $\perp, \top$ .

- (i) Every finite power of the algebra of finite and co-finite subsets of  $\mathbb{N}$  is in **S-AutStr**.
- (ii) The countable atomless Boolean algebra is in **T-AutStr**: It is isomorphic to the algebra of sets consisting of the clopen sets in Cantor space. Each clopen set has a natural representation as a finite tree.
- (iii) The algebra of all subsets of  $\mathbb{N}$  is in  **$\omega$ S-AutStr**.
- (iv) The algebra of all subsets of  $\mathbb{N}$  factored by the congruence of having finite symmetric difference is in  **$\omega$ S-AutStr**. It is unknown whether this structure can be injectively presented in  **$\omega$ S-AutStr**.
- (v) The interval algebra of the real interval  $[0, 1]$  is in  **$\omega$ T-AutStr**.
- (vi) The algebra of all subsets of  $\{0, 1\}^*$  with a distinguished set  $\mathcal{F}$  consisting of those  $X \subset \{0, 1\}^*$  such that for every path  $\pi \in \{0, 1\}^\omega$  only finitely many prefixes of  $\pi$  are in  $X$ .

**Example 1.3.13** (Graphs) (i) The infinite upright grid is in **S-AutStr**:

Here the structure is  $(\mathbb{N} \times \mathbb{N}, \text{Up}, \text{Right})$  with the functions  $\text{Right} : (n, m) \mapsto (n + 1, m)$  and  $\text{Up} : (n, m) \mapsto (n, m + 1)$ . It can be automatically presented on the domain  $a^*b^*$  with relations

$$R = \binom{a}{a}^* \binom{b}{a} \binom{b}{b}^* \binom{\square}{b}$$

<sup>12</sup> Communicated by Damian Niwinski.

- and  $U$  defined by a similar regular expression.
- (ii) The transition graphs of pushdown automata are in  $\mathbf{S}\text{-AutStr}$ .<sup>13</sup> Given a pushdown automaton  $\mathcal{A}$  with states  $Q$ , stack alphabet  $\Gamma$ , input alphabet  $\Sigma$  and transition relation  $\Delta$  we can construct an automatic presentation of the transition graph of its configurations as follows. We take  $Q\Gamma^*$  to be the domain of the presentation in which  $q\gamma$  represents the configuration of state  $q$  and stack  $\gamma \in \Gamma^*$ . For each  $a \in \Sigma$  there is an  $a$ -transition from  $q\gamma$  to  $q'\gamma'$  if, and only if,  $\gamma = z\alpha$ ,  $\gamma' = w\alpha$  and  $(q, z, q', w) \in \Delta$  for some  $z \in \Gamma$  and  $w \in \Gamma^*$ . Since  $\Delta$  is finite, this relation is obviously regular for each  $a$ . Notice that in these presentations the transition relations are not only regular but in fact defined by prefix-rewriting rules (cf. Section 1.2.1 on context-free graphs).
  - (iii) The transition graphs of Turing machines are in  $\mathbf{S}\text{-AutStr}$  [87]. We can give an automatic presentation of each TM  $\mathcal{M}$  similar to those of pushdown automata. Configurations are encoded as strings  $\alpha q \beta \in \Gamma^* Q \Gamma^*$  where  $\alpha$  and  $\beta$  are the tape contents to the left, respectively, to the right of the head of  $\mathcal{M}$ , and  $q$  is the current state. Observe that, as opposed to presentations of pushdown graphs, the state is now positioned not at the left of the string but at the location of the head. Consequently, rewriting is not confined to prefixes, but rather occurs around the state symbol: transitions are of the form  $\alpha u q w \beta \mapsto \alpha u' q' w' \beta$  for adequate  $u, w, u', w'$  and  $q, q'$  as determined by the transition function of  $\mathcal{M}$ . The fact that TM graphs are presentable using *infix rewriting* has the profound consequence that reachability questions in infix-rewriting systems are generally undecidable, as opposed to graphs of *prefix-rewriting* systems, whose monadic second-order theory is decidable (cf. Theorem 1.2.4).

**Example 1.3.14** (Automata-theoretic structures) The following structures turn out to be universal for their respective classes (see Theorem 1.3.17).

- (i) Let

$$\mathcal{S}_\Sigma = (\Sigma^*, \{\mathbf{suc}_a\}_{a \in \Sigma}, \prec_{\text{prefix}}, \mathbf{el})$$

and

$$\mathcal{S}_\Sigma^\omega = (\Sigma^{\leq\omega}, \{\mathbf{suc}_a\}_{a \in \Sigma}, \prec_{\text{prefix}}, \mathbf{el})$$

<sup>13</sup> For *visibly pushdown automata* the same representation of configurations also allows for the trace equivalence relation to be recognised by a finite automaton. In [10] this presentation was utilised to obtain a decidability result.

be the structures defined on finite, respectively on finite and  $\omega$ -words, comprising the successor relations  $\mathbf{suc}_a = \{(w, wa) \mid w \in \Sigma^*\}$ ; the prefix relation  $u \prec_{\text{prefix}} w$  (where  $u$  is finite and  $w$  is finite or infinite); and the equal-length relation:  $u \in w$  if, and only if,  $|u| = |w|$ . Clearly  $\mathcal{S}_\Sigma \in \mathbf{S}\text{-AutStr}$  and  $\mathcal{S}_\Sigma^\omega \in \omega\mathbf{S}\text{-AutStr}$ . Note that if  $\Sigma$  is unary, then  $\mathcal{S}_\Sigma$  reduces to  $(\mathbb{N}, +1, <, =)$ .

- (ii) The structure  $\mathcal{T}_\Sigma \in \mathbf{T}\text{-AutStr}$  has domain consisting of all finite binary  $\Sigma$ -labelled trees and has operations

$$(\preceq_{\text{ext}}, \equiv_{\text{dom}}, (\mathbf{suc}_a^d)_{d \in \{l, r\}, a \in \Sigma}, (\epsilon_a)_{a \in \Sigma})$$

where  $T \preceq_{\text{ext}} S$  if  $\text{dom}(T) \subset \text{dom}(S)$  and  $S(\alpha) = T(\alpha)$  for  $\alpha \in \text{dom}(T)$ ;  $T \equiv_{\text{dom}} S$  if  $\text{dom}(T) = \text{dom}(S)$ ;  $\mathbf{suc}_a^d(T) = S$  if  $S$  is formed from  $T$  by extending its leaves in direction  $d$  and labeling each new such node by  $a$ ; and  $\epsilon_a$  is the tree with a single node labelled  $a$ .

Similarly the structure  $\mathcal{T}_\Sigma^\omega \in \omega\mathbf{T}\text{-AutStr}$  has domain consisting of all finite and infinite trees and operations

$$(\preceq_{\text{ext}}, \equiv_{\text{dom}}, (\mathbf{suc}_a^d)_{d \in \{l, r\}, a \in \Sigma}, (\epsilon_a)_{a \in \Sigma}).$$

that are restricted to finite trees, except that  $T \preceq_{\text{ext}} S$  is defined as above but allows  $S$  to be an infinite tree.

### 1.3.3 Injectivity

Recall that an automatic presentation is injective if the naming function is injective. The problem of injectivity is this:

Does every  $\square$ -automatic structure have an injective  $\square$ -automatic presentation?

An injective presentation has the advantage that it is easier to express certain cardinality-properties of sets of elements (Theorem 1.4.6). We consider the four cases.

#### Finite words

From a finite-word automatic presentation of  $\mathfrak{A}$  one defines an injective presentation of  $\mathfrak{A}$  by restricting to a regular set  $D$  of unique representatives. These can be chosen using a regular well-ordering of the set of all finite words. For instance, define  $D \subset \mathcal{L}(\mathfrak{A})$  to be the length-lexicographically least words from each  $\mathcal{L}(\mathfrak{A}_\approx)$  equivalence class.

### Finite trees

Except in the finite word case, there is no regular well ordering of the set of all finite trees [39]. However one can still convert a finite-tree automatic presentation into an injective one [47]. The idea is to associate with each tree  $t$  a new tree  $\hat{t}$  of the following form: the domain is the intersection of the prefix-closures of the domains of all trees that are  $\mathcal{L}(\mathcal{A}_\approx)$ -equivalent to  $t$ ; a node is labelled  $\sigma$  if  $t$  had label  $\sigma$  in that position; a leaf  $x$  is additionally labelled by those states  $q$  from which the automaton  $\mathcal{A}_\approx$  accepts the pair consisting of the subtree of  $t$  rooted at  $x$  and the tree with empty domain.<sup>14</sup> Using transitivity and symmetry of  $\mathcal{L}(\mathcal{A}_\approx)$ , if  $\hat{t} = \hat{s}$  then  $t$  is  $\mathcal{L}(\mathcal{A}_\approx)$ -equivalent to  $s$ . Moreover each equivalence class is associated with finitely many new trees, and so a representative may be chosen using any fixed regular linear ordering of the set of all finite trees.

### $\omega$ -words

There is a structure in  $\omega\mathbf{S}\text{-AutStr}$  that does not have an injective  $\omega$ -word automatic presentation [82]. The proof actually shows that the structure has no injective presentation in which the domain and atomic relations are Borel.

However, every *countable* structure in  $\omega\mathbf{S}\text{-AutStr}$  does have an injective  $\omega$ -word automatic presentation [85] (and consequently is also in  $\mathbf{S}\text{-AutStr}$ ). This follows from the more general result that every  $\omega$ -word regular equivalence relation with countable index has a regular set of representatives [85].

### $\omega$ -trees

It has not yet been settled whether injective presentations suffice, even for the countable structures.

#### 1.3.4 Alternative characterisations

Automatic structures were defined internally. We now present equivalent characterisations: logical (FO and MSO) and equational.

##### First-order characterisations

In order to capture regularity in the binary representation of  $\mathbb{N}$  using first-order logic Büchi suggested the expansion  $(\mathbb{N}, +, \{2^n \mid n \in \mathbb{N}\})$  of

<sup>14</sup> The construction given in [47] is slightly more general and allows one to effectively factor finite-subset interpretations in any tree.

Presburger arithmetic, which is, however, insufficient (see [26]). Boffa and Bruy  re considered expressively complete expansions of  $(\mathbb{N}, +)$  by relations of the form  $x|_k y$  (defined to hold precisely when  $x$  is a power of  $k$  and  $x$  divides  $y$ ).

**Theorem 1.3.15** (B  chi-Bruy  re, cf. [26]) *A relation  $R \subseteq \mathbb{N}^r$  is regular in the least-significant-digit-first base  $k$  presentation of  $\mathbb{N}$  if, and only if,  $R$  is first-order definable in the structure  $\mathcal{N}_k = (\mathbb{N}, +, |_k)$ .*

Closer to automata, the structures  $\mathcal{S}_\Sigma$  on words (see example 1.3.14) allow one to define every regular relation on alphabet  $\Sigma$ .

**Theorem 1.3.16** ([65]) *Let  $\Sigma$  be a finite, non-unary alphabet. A relation over  $\Sigma^*$  is regular if, and only if, it is first-order definable in  $\mathcal{S}_\Sigma$ .*

The proofs of these theorems are by now standard. From left to write one writes a formula  $\phi_{\mathcal{A}}(x)$  that expresses the existence of a successful run in automaton  $\mathcal{A}$  on input  $x$ . For the other direction the atomic operations of the structures are regular forms the base case for structural induction on the formula. Both theorems transfer to automatic structures by replacing definability with interpretability [24, 25].

**Theorem 1.3.17** (First-order characterisation of **S-AutStr**) *The following conditions are equivalent.*

- $\mathfrak{A} \in \mathbf{S}\text{-AutStr}$ .
- $\mathfrak{A}$  is first-order interpretable in  $\mathcal{S}_\Sigma$  (for some/all  $\Sigma$  with  $|\Sigma| \geq 2$ ).
- $\mathfrak{A}$  is first-order interpretable in  $\mathcal{N}_k$  (for some/all  $k \geq 2$ ).

These structures have been called *universal* or *complete* (with respect to FO-interpretations) for the class of finite-word automatic structures. There are similar universal structures for the other classes of automatic structures. These are the structures  $\mathcal{S}_\Sigma^\omega$ ,  $\mathcal{T}_\Sigma$  and  $\mathcal{T}_\Sigma^\omega$  from Example 1.3.14 [20, 14].

#### Finite set interpretations

The four notions of automatic presentation have straightforward reformulations in terms of subset interpretations either in the line  $\Delta_1 = (\mathbb{N}, \mathbf{suc})$  or in the tree  $\Delta_2 = (\{0, 1\}^*, \mathbf{suc}_0, \mathbf{suc}_1)$ .

**Theorem 1.3.18** (Automatic presentations as subset interpretations) *There are effective transformations establishing the following equivalences.*

- (i)  $\mathfrak{A} \in \mathbf{S}\text{-AutStr}$  if, and only if,  $\mathfrak{A} \leq_{\text{fset}} \Delta_1$
- (ii)  $\mathfrak{A} \in \omega\mathbf{S}\text{-AutStr}$  if, and only if,  $\mathfrak{A} \leq_{\text{set}} \Delta_1$
- (iii)  $\mathfrak{A} \in \mathbf{T}\text{-AutStr}$  if, and only if,  $\mathfrak{A} \leq_{\text{fset}} \Delta_2$
- (iv)  $\mathfrak{A} \in \omega\mathbf{T}\text{-AutStr}$  if, and only if,  $\mathfrak{A} \leq_{\text{set}} \Delta_2$

Equivalently, one may formulate universality with respect to FO interpretations. Following [47] we define the (*finite*) *subset envelope*  $\mathcal{P}_{(f)}(\mathfrak{A})$  of a structure  $\mathfrak{A}$  by adjoining to  $\mathfrak{A}$  its (finite) subsets as new elements ordered by set inclusion.

**Definition 1.3.19** Given  $\mathfrak{A} = (A, \{R_i\})$  write  $P(A)$  for the set of all subsets of  $A$ . The *subset envelope*  $\mathcal{P}(\mathfrak{A})$  is the structure with domain  $P(A)$  and relations  $R'_i := \{\{\{a_1\}, \dots, \{a_n\}\} \mid (a_1, \dots, a_n) \in R_i\}$  and the subset relation  $\subseteq$  defined on  $P(A)$ . The *finite-subset envelope*  $\mathcal{P}_f(\mathfrak{A})$  is the substructure of  $\mathcal{P}(\mathfrak{A})$  whose domain is the set of finite subsets of  $A$ .

It is immediate that

$$\mathfrak{B} \leq_{(f)\text{set}} \mathfrak{A} \iff \mathfrak{B} \leq_{\text{FO}} \mathcal{P}_{(f)}(\mathfrak{A})$$

In particular, this yields natural universal structures, with respect to FO-interpretations, for each of the four classes of automatic structures.

- Corollary 1.3.20**
- (i)  $\mathcal{P}_f(\Delta_1)$  is universal for  $\mathbf{S}\text{-AutStr}$ .
  - (ii)  $\mathcal{P}(\Delta_1)$  is universal for  $\omega\mathbf{S}\text{-AutStr}$ .
  - (iii)  $\mathcal{P}_f(\Delta_2)$  is universal for  $\mathbf{T}\text{-AutStr}$ .
  - (iv)  $\mathcal{P}(\Delta_2)$  is universal for  $\omega\mathbf{T}\text{-AutStr}$ .

### VRS-Equational structures

Recall that the VRS-algebra of graphs extends the VR-algebra with the synchronous product operation and that VRS-equational systems define exactly the finite-tree automatic graphs (see Section 1.2.4 and Theorem 1.2.26).

A finite VRS-equational system whose unfolding is a linear VRS-term specifies a structure in  $\mathbf{S}\text{-AutStr}$ . This happens if in the defining equations one of the arguments of each occurrence of  $\oplus$  and of  $\otimes_S$  is a finite graph (and so these act like unary operations). Conversely, for word-automatic presentations Equation (1.7) reduces to the following form:

$$X = \vartheta (\bullet^\perp \oplus (\vartheta_0 \otimes \vartheta_1(X))) \quad (1.8)$$

This scheme matches the following type definition obtained by restricting (1.6) to words:

$$\mathcal{T} = \perp \oplus (\{0, 1\} \otimes \mathcal{T}) \quad (1.9)$$

This recursive definition of the set of words has the same advantage over (1.2) as (1.6) has over (1.4) when it comes to defining binary relations over words via structural induction, e.g. via finite automata. Over words we have the following special case of Theorem 1.2.26.

**Theorem 1.3.21** (Colcombet [50])

For every countable structure  $\mathfrak{A}$  the following are equivalent

- (1)  $\mathfrak{A}$  is isomorphic to a word-automatic graph.
- (2)  $\mathfrak{A}$  is the restriction of some  $\mathfrak{B}$  to its elements of a certain colour, where  $\mathfrak{B}$  can be specified by a VR-equation  $Z = \pi(X)$ , where  $\pi$  simply forgets some of the structure of  $X$ , together with a VRS-equation for  $X$  of the form (1.8);
- (3)  $\mathfrak{A}$  is finite-subset interpretable in  $(\mathbb{N}, \text{suc})$ .

The equivalence of the first and the third item is a direct consequence of the classical correspondence of automata on words and monadic second-order logic of one successor and was already stated in Theorem 1.3.18. Nonetheless, this can also be inferred from the fact that the solution term obtained by unfolding (1.8) is (essentially) a periodic linear VRS-term that evaluates, via a finite-subset interpretation, to the word-automatic structure specified by equation (1.8).

More generally, let  $\text{VRS}^-$  denote the extension of VR with unary operations  $X \mapsto G_0 \otimes_S X$  where  $G_0$  is any finite graph. Moreover let us call a *chain interpretation* a subset interpretation in a tree where each of the subsets representing an element is linearly ordered by the ancestor relation of the tree. It is not hard to see that solutions of finite systems of  $\text{VRS}^-$ -equations are finite-chain interpretable in a regular tree and that these in turn are word automatic [50].

### 1.3.5 Rational graphs

If we allow the more general *asynchronous automata* in the definition of an automatic presentation of a graph we get the notion of a rational graph. Thus vertices are labelled with finite words of a rational language over some finite alphabet  $\Sigma$ , and the edge relations are required to be rational subsets of  $\Sigma^* \times \Sigma^*$ .

With no aim for completeness we list below some results on rational graphs (asynchronous) in comparison with automatic graphs (synchronous). For a comprehensive treatment the reader is referred to [105].

The class of rational graphs strictly includes that of finite-word automatic graphs. In their seminal paper [87] Khoussainov and Nerode also introduced asynchronous automatic structures. As an example they gave an asynchronous automatic presentation of  $\omega^\omega$ , which is not in **S-AutStr** (see Theorem 1.4.12). Asynchronous automatic presentations of Cayley-graphs of finitely generated groups have also been considered as generalisations of ‘automatic groups’ [31].

The price of increasing expressiveness is a loss of tractability: in general, rational graphs do not have a decidable first-order theory. This renders rational graphs useless for representing data, let alone programs. However, in the context of formal language theory rational graphs seem to fill a gap. Considering rational graphs as infinite automata, i.e. as acceptors of languages, Morvan and Stirling have shown that they trace exactly the context-sensitive languages [108, 107] (see also [34] for a simplified approach). Rispal and others [123, 107, 34] have subsequently observed that this holds true for automatic graphs as well.

Although first-order queries on rational graphs are in general intractable there are some interesting decidable subclasses.

Morvan observed that by a result of Eilenberg and Schützenberger, graphs defined by rational relations over a *commutative* monoid have a decidable first-order theory. In particular, over the unary alphabet the monoid structure is isomorphic to  $(\mathbb{N}, +)$  whence the unary rational graphs are those first-order definable in  $(\mathbb{N}, +)$  [105]. Similarly, rational graphs over  $(\mathbb{N}, +)^d$  are those having a  $d$ -dimensional first-order interpretation in  $(\mathbb{N}, +)$ .

Carayol and Morvan showed that on rational graphs that also happen to be trees (this is an undecidable property) first-order logic is decidable [36, 106]. The decision method is based on locality of FO as formulated by Gaifman and uses a compositional technique. The authors also exhibit a rational graph that is a finitely branching tree but is not finite-word automatic.

### 1.3.6 Generalisations

#### Automata with oracles

Consider an expansion  $\Delta_i^O$  of  $\Delta_i := ([i]^*, \mathbf{suc}_0, \dots, \mathbf{suc}_{i-1})$  by a unary predicate  $O \subset [i]^*$ . Every MSO formula (with free MSO variables) of the expanded structure corresponds to a tree automaton with *oracle*  $O$ . An automaton with oracle is one that, while in position  $u \in [i]^*$ , can decide on its next state using the additional information of whether or

not  $u \in O$ . Thus for automata working on infinite words/trees the oracle  $O$  is simply read as part of the input. In the case of automata working on finite words/trees, the entire oracle is scanned, and so the acceptance condition should be taken appropriately (eg. Muller/Rabin).

Call a set  $O$  *decidable* if  $\text{MSO}(\Delta_i^O)$  is decidable, and *weakly decidable* if  $\text{wMSO}(\Delta_i^O)$  is decidable. Early work on decidable oracles used the contraction method to show that certain oracles on the line, such as  $\{n! \mid n \in \mathbb{N}\}$ , are decidable [67]. This was extended to the profinitely ultimately periodic words [38], which it turns out capture all the decidable unary predicates on the line [119, 120]. Nonetheless, it is still of interest to produce explicit examples of decidable oracles, see for instance [38, 74, 75, 7].

**Definition 1.3.22** If in the definition of automatic presentation (1.3.2) we replace  $\square$ -automata with  $\square$ -automata with oracle  $O$ , we get a notion of  $\square$ -automatic presentation with oracle  $O$ . A structure is called *automatic with oracle* if it has a  $\square$ -automatic presentation with some oracle.

**Example 1.3.23** The group of rationals  $(\mathbb{Q}, +)$  has recently been shown to have no word-automatic presentation [136]. However it is finite-word automatic with oracle  $\#2\#3\#4\#\dots$ . This is based on the idea, independently found by Frank Stephan and Joe Miller and reported in [114], that there is a presentation of  $([0, 1] \cap \mathbb{Q}, +)$  by finite words in which  $+$  is regular, but the domain is not: every rational in  $[0, 1]$  can be expressed as  $\sum_{i=2}^n \frac{a_i}{i!}$  for a unique sequence of natural numbers  $a_i$  satisfying  $0 \leq a_i < i$ . The presentation codes this rational as  $\#a_2\#a_3\#a_4\#\dots$  where  $a_i$  is written in decimal notation (and hence has length less than the length of  $i$  written in decimal notation). Addition is performed with the least significant digit first, based on the fact that

$$\frac{a_i + b_i + c}{i!} = \frac{1}{(i-1)!} + \frac{a_i + b_i + c - i}{i!}$$

where  $c \in \{0, 1\}$  is the carry in.

We immediately have that a structure is (finite-)word/tree automatic with oracle  $O$  if and only if it is (finite) set interpretable in  $\Delta_1^O/\Delta_2^O$ . Hence we have the following generalisation of the Fundamental Theorem and its corollaries (1.3.4).

**Theorem 1.3.24** (i) Definability: Say  $(\mathfrak{d}, f)$  is a  $\square$ -automatic presentation with oracle  $O$  of a structure  $\mathfrak{A}$  and  $\varphi(\bar{x})$  is a FO-formula in the

signature of  $\mathfrak{A}$  defining a  $k$ -ary relation  $R$  over  $\mathfrak{A}$ . Then the relation  $f^{-1}(R)$  is recognised by an  $\square$ -automaton with oracle  $O$ .

- (ii) Interpretations: The class of  $\square$ -automatic structures with oracle  $O$  is closed under FO-interpretations.
- (iii) Decidability: The previous statements can be made effective under the following conditions.

1 For  $\square \in \{\text{word, tree}\}$  we require that  $\text{wMSO}(\Delta_i^O)$  be decidable.

2 For  $\square \in \{\omega\text{-word, } \omega\text{-tree}\}$  we require that  $\text{MSO}(\Delta_i^O)$  be decidable.

In particular, under these conditions, every  $\mathfrak{A}$  that is  $\square$ -automatic with oracle  $O$  has decidable FO-theory.

Of course  $\Delta_i^O$  can be viewed as a coloured tree. As in Corollary 1.3.20 we have universal structures with respect to FO-definability. For instance  $\mathcal{P}(\Delta_2^O)$  is universal for  $\omega\text{T-AutStr}$  with oracle  $O$ . The following result concerns finite-set interpretations in arbitrary trees.

**Theorem 1.3.25** ([47]) To every finite set interpretation  $\mathcal{I}$  one can effectively associate a wMSO interpretation  $\mathcal{J}$  such that for every tree  $t$  and structure  $\mathfrak{A}$  if  $\mathcal{P}_f(\mathfrak{A}) \cong \mathcal{I}(t)$  then  $\mathfrak{A} \cong \mathcal{J}(t)$ .

This can be used to show that certain structures, such as the random graph, are not finite-tree automatic in the presence of any oracle [47].

### 1.3.7 Subclasses

In this section we restrict the complexity of the regular domains in automatic presentations to yield some of the more robust subclasses of S-AutStr and T-AutStr.

#### Polynomial domain

The most natural restriction is to consider presentations where the words and trees take labels from a unary alphabet  $|\Sigma| = 1$ . Word-automatic presentations over a unary alphabet were introduced and studied by Blumensath [20] and Rubin [89, 124].

The *density* of a language  $L \subset \Sigma^*$  is the function  $n \mapsto |L \cap \Sigma^n|$ .

**Definition 1.3.26** A structure is *unary automatic* if it has an injective word-automatic presentation in which the domain consists of words from a unary alphabet. A structure is *p-automatic* if it has an injective word-automatic presentation in which the domain has polynomial density. Let 1-AutStr and P-AutStr denote these respective classes of structures.

Regular sets of polynomial density were characterised by Szilard et al. [131] as being a finite union of the form

$$D = \bigcup_{i < N} u_{i,1} v_{i,1}^* u_{i,2} \dots u_{i,n_i} v_{i,n_i}^* u_{i,n_i+1} \quad (1.10)$$

where the degree of the polynomial of the density function is equal to the maximum of the  $n_i$ 's. In [6] it was demonstrated that every finite-word-automatic presentation over a domain as in (1.10) can be transformed into an equivalent one (cf. Section 1.4.4) over a domain that is a regular subset of

$$a_1^* a_2^* \dots a_n^*$$

where  $n$  is equal to the maximum of the  $n_i$ 's. In particular, word-automatic presentations over a domain of linear density are unary automatic. This transformation yields a kind of normal-form of word-automatic presentations over a polynomially growing domain.

**Theorem 1.3.27** ([6]) *A structure  $\mathfrak{A}$  has an automatic presentation over a domain of density  $\mathcal{O}(n^d)$  if, and only if, it has a  $d$ -dimensional interpretation in  $\mathfrak{M} := (\mathbb{N}, <, \{\equiv_{(mod m)}\}_{m > 1})$  if, and only if, it is finite-subset interpretable in  $\Delta_1 := (\mathbb{N}, \text{suc})$  with subsets of size at most  $d$ .*

**Corollary 1.3.28** ([113],[20]) *A structure  $\mathfrak{A}$  is unary automatic if, and only if, it is first-order definable in  $\mathfrak{M}$  if, and only if, it is MSO-interpretable in  $\Delta_1$ .*

Unary automatic structures form a very restricted subclass of VR-equational structures and have a decidable MSO-theory. Using pumping arguments one can show that Presburger arithmetic  $(\mathbb{N}, +)$  has no p-automatic presentation [20, 121]. On the other hand, the infinite grid is p-automatic but not unary automatic. Thus we have

$$\text{1-AutStr} \subsetneq \text{P-AutStr} \subsetneq \text{S-AutStr} .$$

The expansion of  $\mathfrak{M}$  with the successor function **suc** and a constant for 0 admits quantifier elimination. Hence, every p-automatic structure can be interpreted in  $(\mathbb{N}, 0, \text{suc}, <, \{\equiv_{(mod m)}\}_{m > 1})$  using quantifier-free formulas.

Every p-automatic structure inherits the PSPACE upper-bound on the complexity of its first-order theory from  $\mathfrak{M}$ . This is as low as possible since FO model-checking is PSPACE-hard for any structure with at least two elements. Adding even the simplest form of iteration to

FO leads to undecidability. For every  $k$ -counter machine it is straightforward to construct a  $p$ -automatic presentation of its configuration graph where each configuration  $(q, n_1, \dots, n_k)$  is represented by the word  $qc_1^{n_1} \cdots c_k^{n_k}$ . It follows that the first-order theory with reachability  $\text{FO}[\mathbf{R}]$  of a  $p$ -automatic structure is undecidable in general. In comparison, while unary automatic structures have a decidable MSO-theory, the  $\text{FO}(\text{DTC})$  theory of  $(\mathbb{N}, \text{succ})$  interprets full first-order arithmetic and is therefore highly undecidable [20].

Observe, that graphs having rational presentation over a finitely generated commutative monoid (cf. Section 1.3.5) can be seen as analogues of  $p$ -automatic graphs. Indeed, every monoid element is represented by some word  $g_1^{r_1} g_2^{r_2} \cdots g_n^{r_n}$  over the generators.

#### Finite-rank tree-automatic presentations

The analogue of  $p$ -automatic to tree-automatic structures is restricting to presentations involving trees of bounded rank. Intuitively the rank of a tree corresponds to its branching degree (which can be measured in terms of the Cantor-Bendixson rank).

Recall a  $\Sigma$ -labelled  $n$ -ary tree  $T$  is a function from a prefix-closed subset of  $[n]^*$  to  $\Sigma$ . We say that  $T$  has *rank*  $k$  if its domain has polynomial density of degree at most  $k$ .

A finite-tree automatic presentation is called of *rank*  $k$  if for some regular language  $D$  of polynomial density of degree at most  $k$  the domain of every tree in the presentation is a subset of  $D$ . Collectively we speak of *bounded-rank tree-automatic presentations*. The class of structures with rank  $k$  presentations is denoted  $\mathbf{k}\text{-T-AutStr}$ .

**Example 1.3.29** The ordinal  $\omega^{\omega^k}$  has a rank  $k + 1$  tree-automatic presentation.

Let  $\mathcal{T}_k$  denote the structure corresponding to the unlabelled  $k$ -ary tree with domain  $0^*1^* \cdots (k-1)^*$ . Note that  $\mathcal{T}_k$  is wMSO-interpretable in the ordinal  $\omega^k$  (in the signature of order), and vice-versa.

**Proposition 1.3.30** *The following are equivalent.*

- $\mathfrak{A}$  is in  $\mathbf{k}\text{-T-AutStr}$ ,
- $\mathfrak{A}$  is finite-set interpretable in  $\mathcal{T}_k$  (or equivalently in the ordinal  $\omega^k$ ),
- $\mathfrak{A}$  is the solution of a finite system of VRS-equations whose unfolding is a term-tree of rank  $k$ .

The hierarchy is strict:

$$\mathbf{S}\text{-AutStr} = \mathbf{1}\text{-T-AutStr} \subsetneq \mathbf{2}\text{-T-AutStr} \subsetneq \cdots \subsetneq \mathbf{T}\text{-AutStr}.$$

Indeed, if  $\mathbf{k+1}\text{-T-AutStr} = \mathbf{k}\text{-T-AutStr}$  for some  $k$  then the finite-subset envelope  $\mathcal{P}_f(\omega^{k+1})$  would be finite-set interpretable in  $\omega^k$ . But by Theorem 1.3.25 then  $\omega^{k+1}$  is wMSO interpretable in  $\omega^k$ , which is known not to be possible [98, Lemma 4.5].<sup>15</sup>

### 1.3.8 Comparison of classes

Since words are special cases of trees, and finite ones special cases of infinite ones, one immediately sees the inclusions indicated by the arrows in the figure. All the arrows except for the dotted one are known to be strict inclusions. We now discuss the separating examples as well as the double lines indicating equality of the classes when restricted to countable structures. Since  $\omega\mathbf{S}\text{-AutStr}$  and  $\omega\mathbf{T}\text{-AutStr}$  contain uncountable structures while  $\mathbf{S}\text{-AutStr}$  and  $\mathbf{T}\text{-AutStr}$  do not, we split our discussion along these lines.

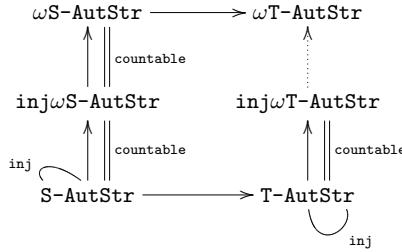


Figure 1.2 Relationship of classes of automatic structures

#### Countable structures

The structure  $(\mathbb{N}, \times)$  separates  $\mathbf{T}\text{-AutStr}$  from  $\mathbf{S}\text{-AutStr}$  (see [20], or [88] for an alternative proof).

Every injective  $\omega\mathbf{S}\text{-AutStr}$  presentation of a countable structure can be effectively transformed into a  $\mathbf{S}\text{-AutStr}$  presentation. This is because a countable  $\omega$ -regular set  $X \subseteq \{0, 1\}^\omega$  only contains ultimately periodic words, and moreover there is a bound on the size of the periods

<sup>15</sup> We thank Lukasz Kaiser for discussions on the notions of this section and Alex Rabinovich for providing the latter reference.

(which can be computed from an automaton for  $X$ ). Similar facts hold for countable regular sets of infinite trees [115].

The next theorem generalises this in the word case:

**Theorem 1.3.31** ([85]) (i) *The countable structures in  $\omega\text{S-AutStr}$  are precisely those in  $\text{S-AutStr}$ .*

(ii) *Given a (not necessarily injective) automatic presentation of some  $\mathfrak{A} \in \omega\text{S-AutStr}$  it is decidable whether  $\mathfrak{A}$  is countable or not, and if it is, an automatic presentation of  $\mathfrak{A}$  over finite words can be constructed.*

On the other hand, we do not know whether every countable structure in  $\omega\text{T-AutStr}$  is in  $\text{T-AutStr}$ .

### Uncountable structures

The only known non-trivial methods dealing with uncountable structures appear in [82]:

- (i) The algebra  $(\mathcal{P}(\{0, 1\}^*), \cap, \cup, \cdot^c, \mathcal{F})$  from example 1.3.12(6) is an uncountable structure separating  $\omega\text{T-AutStr}$  from  $\omega\text{S-AutStr}$ .
- (ii) Recall Example 1.3.12(4) consisting of the algebra of subsets of  $\mathbb{N}$  (call it  $\mathcal{A}$ ) quotiented by having finite symmetric difference (call it  $\approx$ ). Construct a variant structure as the disjoint union of  $\mathcal{A}$  and  $\mathcal{A}/\approx$ , with a unary predicate  $U$  identifying the elements of  $\mathcal{A}$  and a binary relation  $R$  relating  $a \in A$  to its representative in  $\mathcal{A}/\approx$ . This uncountable structure separates  $\omega\text{S-AutStr}$  from  $\text{inj}\omega\text{S-AutStr}$ .

## 1.4 More on word-automatic presentations

### 1.4.1 Beyond first-order logic

The Fundamental Theorem can be strengthened to include order-invariant definable formulas as well as certain additional quantifiers.

#### Generalised quantifiers

We briefly recall the definition of generalised quantifiers as introduced by Lindström.

**Definition 1.4.1** Fix a finite signature  $\tau = (R_i)_{i \leq k}$ , where  $R_i$  has associated arity  $r_i$ . A *quantifier*  $Q$  is a class of  $\tau$ -structures closed under isomorphism. Let  $\sigma$  be another signature. Given  $\sigma$ -formulas  $\Psi_i(\bar{x}_i, \bar{z})$

with  $|\bar{x}_i| = r_i$  ( $i \leq k$ ), the syntax  $Q\bar{x}_1, \dots, \bar{x}_n(\Psi_1, \dots, \Psi_k)$  has the following meaning on a  $\sigma$ -structure  $\mathcal{A}$ :

$$(\mathcal{A}, \bar{a}) \models Q\bar{x}_1, \dots, \bar{x}_k(\Psi_1, \dots, \Psi_k) \text{ iff } (A; \Psi_1^{\mathcal{A}}(\cdot, \bar{a}), \dots, \Psi_k^{\mathcal{A}}(\cdot, \bar{a})) \in Q,$$

where  $\Psi^{\mathcal{A}}(\cdot, \bar{a})$  is the relation defined in  $\mathcal{A}$  by  $\Psi$  with parameters  $\bar{a}$ . The *arity* of a quantifier is the maximum of the  $r_i$ s. A quantifier is *n-ary* if its arity is at most  $n$ .

The extension of first-order logic by a collection  $\mathbf{Q}$  of generalised quantifiers will be denoted  $\text{FO}[\mathbf{Q}]$ .

**Examples 1.4.2** (i) The unary quantifier  $\{(A; X) \mid \emptyset \neq X \subset A\}$  is ‘there exists’.

(ii) The unary quantifier ‘there exists infinitely many’, written  $\exists^\infty$ , is the class of structures  $(A; X)$  where  $X$  is an infinite subset of  $A$ .

(iii) The unary *modulo quantifier* ‘there are  $k$  modulo  $m$  many’ (here  $0 \leq k < m$ ), written  $\exists^{(k,m)}$ , is the class of structures  $(A; X)$  where  $X$  contains  $k$  modulo  $m$  many elements. Write  $\exists^{\text{mod}}$  for the collection of modulo quantifiers.

(iv) The unary *Härtig quantifier* is the class of structures  $(A; P, Q)$  where  $P, Q \subset A$  and  $|P| = |Q|$ .

(v) Every set  $C \subset (\mathbb{N} \cup \{\infty\})^n$  induces the unary *cardinality quantifier*  $Q_C = \{(A; P_1, \dots, P_n) \mid (|P_1|, \dots, |P_n|) \in C\}$ . In fact, a given unary quantifier over signature  $(R_i)_{i \leq k}$  is identical to some cardinality quantifier with  $n = 2^k$ .

(vi) The binary *reachability quantifier* is the class of structures of the form  $(A; E, \{c_s\}, \{c_f\})$  where  $E \subset A^2$ ,  $c_s, c_f \in A$ , and there is a path in the directed graph  $(A; E)$  from  $c_s$  to  $c_f$ .

(vii) The  $k$ -ary *Ramsey quantifier*  $\exists^{k\text{-ram}}$  is the class of structures  $(A; E)$ ,  $E \subset A^k$ , for which there is an infinite  $X \subset A$  such that for all pairwise distinct  $x_1, \dots, x_k \in X$ ,  $E(x_1, \dots, x_k)$ .

The following general definition will allow us to compare the expressive strength of quantifiers.

**Definition 1.4.3** Let  $Q$  be a quantifier,  $\mathbf{Q}$  a collection of quantifiers, and  $\tau$  the signature of  $Q$ . Say that  $Q$  is *definable in*  $\mathbf{Q}$  if there is a sentence  $\theta$  over the signature  $\tau$  in the logic  $\text{FO}[\mathbf{Q}]$  with  $Q = \{\mathcal{A} \mid \mathcal{A} \models \theta\}$ .

For instance, a structure  $(A; X)$  satisfies  $\exists^{(0,2)} z X(z) \vee \exists^{(1,2)} z X(z)$  if and only if  $X$  is finite. Hence  $\exists^\infty$  is definable in  $\{\exists^{(0,2)}, \exists^{(1,2)}\}$ .

Of course the generalised quantifiers that interest us most are the ones, like  $\forall$  and  $\exists$ , that preserve regularity.

**Definition 1.4.4** Fix class  $\mathcal{C}$  as one of  $\text{S-AutStr}$ ,  $\text{T-AutStr}$ ,  $\omega\text{S-AutStr}$ , or  $\text{T-AutStr}$ . Let  $Q$  be a quantifier with signature  $\tau = (R_i)_{i \leq k}$ , where  $R_i$  has associated arity  $r_i$ . Say that quantifier  $Q$  *preserves regularity for the class  $\mathcal{C}$*  if for every  $n \in \mathbb{N}$ , and every automatic presentation  $\mu$  of a structure  $\mathcal{A} \in \mathcal{C}$ , every formula

$$Q\bar{x}_1, \dots, \bar{x}_k(\Psi_1^{\mathcal{A}}(\bar{x}_1, \bar{z}), \dots, \Psi_k^{\mathcal{A}}(\bar{x}_k, \bar{z}))$$

defines a relation  $R$  in  $\mathcal{A}$  with  $\mu^{-1}(R)$  regular (here  $\bar{z} = (z_1, \dots, z_n)$  and the  $\Psi_i$  are first-order  $\mathcal{A}$ -formulas).

Say that  $Q$  *preserves regularity effectively* if an automaton for  $\mu^{-1}(R)$  can effectively be constructed from the automata of the presentation and the formulas  $\Psi_i$ .

Since not every structure is injectively presentable, we may restrict this definition to the class  $\mathcal{C}$  of injectively presentable structures from  $\omega\text{S-AutStr}$  (or  $\omega\text{T-AutStr}$ ). For this, replace ‘automatic presentation’ with ‘injective automatic presentation’ in the above definition.

**Example 1.4.5** The reachability quantifier is not regularity preserving (for any of the classes). For otherwise, by Example 1.3.13, the set of starting configurations that drive a given Turing Machine to a halting state would be regular, and hence computable.

The first steps have been taken in exploring those quantifiers that preserve regularity.

**Theorem 1.4.6** Let  $\mathcal{C}$  be any of the following classes of structures  $\text{inj-}\omega\text{T-AutStr}$ ,  $\omega\text{S-AutStr}$ ,  $\text{T-AutStr}$ ,  $\text{S-AutStr}$ .

- (i) The following unary quantifiers preserve regularity effectively for  $\mathcal{C}$ :  $\exists^\infty, \exists^{\text{mod}}, \exists^{\leq\aleph_0}, \exists^{>\aleph_0}$  [20, 90, 94, 85, 9].
- (ii) Every unary quantifier that preserves regularity for the class  $\text{S-AutStr}$  is already definable from  $\exists^{\text{mod}}, \exists^\infty$  [125].

The second item also implies that every unary quantifier that preserves regularity for the class  $\text{inj-}\omega\text{S-AutStr}$  is already definable from  $\exists^{\text{mod}}, \exists^\infty, \exists^{\leq\aleph_0}, \exists^{>\aleph_0}$ . This is because for an  $\omega$ -regular relation  $R(\bar{x}, \bar{z})$  the cardinality of the set  $R(-, \bar{c})$  (for any fixed parameter  $\bar{c}$ ) is finite, countable or has size continuum [94].

**Theorem 1.4.7** (see [125]) Each  $k$ -ary Ramsey quantifier preserves regularity effectively for the class  $\text{S-AutStr}$ .

Kuske and Lohrey observed that the proof of this theorem can be generalised to quantifiers of the form ‘there exists an infinite set  $X$  satisfying  $\theta$ ’, where  $\theta$  is a property of sets closed under taking subsets. They use this to show that certain problems, while  $\Sigma_1^1$ -complete for recursive graphs, are decidable on automatic graphs [96].

### Order-invariance

**Definition 1.4.8** Fix a signature  $\tau$  and a new symbol  $\leq$ . A formula  $\phi(\bar{x})$  in the signature  $\tau \cup \{\leq\}$  is called *order invariant* on a  $\tau$ -structure  $\mathcal{A}$  if for all tuples  $\bar{a}$  from  $A$  and all linear orders  $\leq_1$  and  $\leq_2$  on  $A$ , we have that  $(\mathcal{A}, \leq_1) \models \phi(\bar{a})$  if and only if  $(\mathcal{A}, \leq_2) \models \phi(\bar{a})$ . The *relation defined by the order invariant  $\phi$  in  $\mathcal{A}$*  is the set of tuples  $\bar{a}$  from  $A$  such that  $(\mathcal{A}, \leq) \models \phi(\bar{a})$  for some (and hence all) linear orders  $\leq$  on  $A$ .

The Fundamental Theorem can be extended on injective presentations to include order-invariant formulas in those cases where there is a regular linear ordering of the set  $f^{-1}(A)$ . On finite-words, finite-trees and  $\omega$ -words there are regular linear orderings. However, we do not know if there is a regular linear ordering on the set of all  $\omega$ -trees. On the other hand, certain separating examples from finite model theory are adaptable to the automatic world.

**Proposition 1.4.9** ([5]) *There exists a structure  $\mathfrak{B} \in \mathbf{S}\text{-AutStr}$  and an order-invariant definable relation  $S^*$  in  $\mathfrak{B}$  that is not definable in  $\mathfrak{B}$  using any extension of FO with only unary quantifiers.*

### 1.4.2 Complexity of some problems

#### First-order theories

By Theorem 1.3.4 query-evaluation and model-checking for first-order formulas are effective on automatic structures. However, the complexity of these problems is in general non-elementary, i.e. it exceeds any fixed number of iterations of the exponential function. For instance the first-order theories of the universal structures  $\mathcal{N}_k$  and  $\mathcal{S}_{[k]}$  ( $k \geq 2$ ) have non-elementary complexity [77] (cf. also the remark after Example 1.4.39).

There are various sensible ways of measuring model-checking complexity. First, one may fix a formula and ask how the complexity depends on the input structure. This measure is called *structure complexity*. On the other hand, *expression complexity* is defined relative to a fixed structure in terms of the length of the formula. Finally, one can look at the combined complexity where both parts may vary.

	Structure-Complexity <sup>a</sup>	Expression-Complexity
Model-Checking		
$\Sigma_0$	LOGSPACE-complete	ALOGTIME-complete <sup>16</sup>
$\Sigma_0 + \text{func}$	NLOGSPACE	in quadratic time <sup>16</sup> and PTIME-complete
$\Sigma_1$	PTIME <sup>17</sup>	PSPACE-complete (EXPTIME-c. for T-AutStr)
$\Sigma_2$	PSPACE-complete <sup>17</sup>	EXPSPACE-complete (2EXPTIME-c. for T-AutStr)
Query-Evaluation		
$\Sigma_0$	LOGSPACE	PSPACE
$\Sigma_1$	PSPACE	EXPSPACE

Figure 1.3 Complexity of fragments of FO on automatic structures

<sup>a</sup> Structure complexity is measured in terms of the size of the largest deterministic automaton in the input presentation.

In [25] Blumensath and Grädel studied the expression and structure complexity of model-checking and query evaluation for quantifier-free and existential first-order formulas both in a relational signature and allowing terms in quantifier-free formulas. Their results are complemented by those of Kuske and Lohrey [95] on the expression complexity of  $\Sigma_1$  (existential) and  $\Sigma_2$  formulas of a relational signature over arbitrary word- and tree-automatic structures. Figure 1.3 provides a summary.

On certain subclasses of automatic structures there is better complexity. In section 1.3.7 above we have mentioned that the first-order theory of each structure allowing a word-automatic presentation of polynomial density is decidable in PSPACE. Kuske and Lohrey [101, 95] studied automatic structures whose Gaifman graphs are of *bounded degree*. Relying on locality of first-order logic they have identified the expression complexity of FO model checking on word-automatic and tree-automatic structures of bounded degree to be 2EXPSPACE-complete and 3EXPTIME-complete, respectively. The combined complexity remains 2EXPSPACE for word-automatic presentations and is in 4EXPTIME for tree-automatic presentations. For finer results we refer to [95].

<sup>16</sup> This is a generalisation of the quadratic solution of the word problem in automatic groups [31] (see Section 1.4.5).

<sup>17</sup> Model checking with a fixed  $\Sigma_1$  formula reduces to a membership or non-emptiness test for an NFA. For fixed  $\Pi_2$  formulas the problem is polynomially equivalent to the universality problem of NFAs, and thus PSPACE-complete. (We thank Anthony To for pointing out the error in [25].)

### Beyond first-order

A fundamental problem in verification is deciding *reachability*: whether there is a path between specified source and target nodes. Since the configuration space of an arbitrary Turing machine is finite-word automatic, the halting problem can be reduced to the reachability problem on the configuration graph of a universal Turing-machine. Similar reductions show the undecidability, over (finite-word) automatic structures, of connectivity, isomorphism, bisimulation and hamiltonicity [25, 96].

On the other hand there are natural classes of automatic structures for which these problems become decidable (see Figure 1.1). For instance, VRA-equational graphs have a decidable FO-theory with reachability and are finite-tree automatic. Reachability and connectivity in locally-finite unary-automatic graphs are in fact decidable in PTIME. Bisimulation equivalence of HR-equational graphs of finite out-degree is decidable [128] (see section 1.2.2).

Finally we mention some cases where full MSO is decidable. Prefix recognisable structures (which include the unary automatic structures) are finite-word automatic. A structure of the form  $(\mathbb{N}, <, C_1, \dots, C_k)$  is called a colouring of the line. Every known finite-word automatic colouring of the line, and this includes every morphic sequence, has decidable MSO-theory (cf. Theorem 1.4.38 and see [7]). Furthermore, every word-automatic equivalence relation has a decidable MSO-theory. This follows from the above and the observation (Proposition 1.4.40) that if there are only finitely many infinite classes then the equivalence relation is FO-definable in some word-automatic colouring of the line [7].

### Isomorphism problem

A measure of the complexity of a class of structures is the *isomorphism problem*, namely the problem of deciding, given two  $\square$ -automatic presentations  $\mathfrak{d}$  and  $\mathfrak{d}'$ , whether or not the structures they present are isomorphic.

The characterisations of the finite-word automatic Boolean algebras and ordinals [88, 63] imply that the isomorphism problem for each of these classes is decidable. Also, as noted, the isomorphism problem for equational graphs is decidable 1.2.10.

Configuration spaces of Turing machines are locally finite and the complexity of the isomorphism problem for locally-finite directed graphs in **S-AutStr** is  $\Pi_3^0$ -complete [124]. However, by massaging the configuration spaces we get that the isomorphism problem for automatic graphs is as hard as possible:  $\Sigma_1^1$ -complete. This is done by reducing the isomorphism

problem for computable structures, known to be  $\Sigma_1^1$ -complete, to that of automatic structures.

**Theorem 1.4.10** ([124]) *The complexity of the isomorphism problem for each of the following classes of  $\mathbf{S}\text{-AutStr}$  structures is  $\Sigma_1^1$ -complete: (i) undirected graphs, (ii) directed graphs, (iii) successor trees, and (iv) lattices of height 4.*

*Problem 1.4.11* What is the exact complexity of the isomorphism problem for the following classes: <sup>18</sup>

- (i) Automatic equivalence structures (easily seen to be  $\Pi_1^0$ ).
- (ii) Automatic linear orders.

### Traces

Infinite edge-labelled graphs, when viewed as infinite automata, can accept non-regular languages. Naturally, context-free graphs accept precisely the context-free languages. Though prefix-recognisable graphs form a structurally much richer class they have the same language accepting power as context-free graphs (cf. Theorem 1.2.11 items (1) and (6)). Graphs in the Caucal hierarchy have the same accepting power as higher-order pushdown automata (see Theorem 1.2.16) tracing languages on the corresponding levels of the OI-hierarchy of [62]. The traces of GTRS-graphs form a language class in between the context-free and context-sensitive classes of the Chomsky hierarchy [99]. Rational graphs accept precisely the context-sensitive languages [108]. All context-sensitive languages can in fact be accepted by word-automatic graphs [123], cf. also [35] for a more accessible proof and finer analysis. Meyer proved that the traces of tree-automatic graphs are those languages recognisable in ETIME, i.e. in  $2^{\mathcal{O}(n)}$  time [103].

#### 1.4.3 Non-automaticity via pumping and counting

It is usually quite simple to show that a structure has an automatic presentation (if indeed it does have one!). On the other hand, there are only a handful of elementary techniques for showing that a structure has no automatic presentation. Most rely on the pumping lemma of automata theory.

<sup>18</sup> While this work has been in print, Kuske, Liu and Lohrey have greatly contributed to settling these and related questions. We refer to their forthcoming paper.

Sometimes we can provide a full characterisation of classes of automatic structures. The first non-trivial characterisation was for the word-automatic ordinals (in the signature of order).

**Theorem 1.4.12** (Delhommé [63])

- (i) An ordinal  $\alpha$  is in **S-AutStr** if, and only if,  $\alpha < \omega^\omega$ .
- (ii) An ordinal  $\alpha$  is in **T-AutStr** if, and only if,  $\alpha < \omega^{\omega^\omega}$ .

A relation  $R$  is *( $n+m$ ) locally finite* if for every  $(x_1, \dots, x_n)$  there are only finitely many  $(y_1, \dots, y_m)$  such that  $R(\bar{x}, \bar{y})$  holds. Obviously, every functional relation  $f(\bar{x}) = y$  is locally finite. Other examples of locally finite relations are equal-length **e1**, length comparison  $|y| < |x|$ , and the prefix relation  $y \prec_{\text{prefix}} x$ . Note that local finiteness depends on the partitioning of the variables, e.g.  $x \prec_{\text{prefix}} y$  is not locally finite.

A simple pumping argument gives the following important tool.

**Proposition 1.4.13** (Elgot and Mezei [66]) *Let  $R \subseteq (\Sigma^*)^{n+m}$  be a regular and locally finite relation. Then there is a constant  $k$  such that for all  $\bar{x}, \bar{y}$  satisfying  $R$ ,  $\max_j |y_j| \leq \max_i |x_i| + k$ . In particular, if  $f$  is a regular function then there is a constant  $k$  such that for every  $\bar{x}$  in its domain we have  $|f(\bar{x})| \leq \max_i |x_i| + k$ .*

### Growth of generations

Consider a structure  $\mathfrak{A}$  with functions  $\mathcal{F} = \{f_1, \dots, f_s\}$  and a sequence  $E = \{e_0, e_1, e_2, \dots\}$  of elements of  $\mathfrak{A}$ . The generations of  $E$  with respect to  $\mathcal{F}$  are defined recursively as follows.

$$\begin{aligned} G_{\mathcal{F}}^0(E) &= \{e_0\} \\ G_{\mathcal{F}}^{n+1}(E) &= G_{\mathcal{F}}^n(E) \cup \{e_{n+1}\} \\ &\quad \cup \{f(\bar{a}) \mid f \in \mathcal{F}, a_i \in G_{\mathcal{F}}^n(E) \text{ for each } i \leq |\bar{a}|\} \end{aligned}$$

We are interested in how fast  $|G_{\mathcal{F}}^n(E)|$  grows as a function of  $n$ .

**Example 1.4.14** (i) Free semigroup on  $m$  generators: here  $\mathcal{F} = \{\cdot\}$  and  $E = \{e_1, \dots, e_m\}$ . For  $m \geq 2$ , since  $G_{\mathcal{F}}^m(E) \supset E$ , the set  $G_{\mathcal{F}}^{m+n}(E)$  includes all strings over  $E$  of length at most  $2^n$ ; thus the cardinality of  $G_{\mathcal{F}}^{m+n}(E)$  is at least a double exponential in  $n$ .  
(ii) If  $p : D \times D \rightarrow D$  is injective then for  $\mathcal{F} = \{p\}$  and  $E = \{e_1, e_2\}$  (distinct elements of  $D$ )  $|G_{\mathcal{F}}^n(E)|$  is at least a double exponential.

We now iterate Proposition 1.4.13.

**Proposition 1.4.15** ([87],[20, 25]) *Let  $\mathfrak{A} \in \mathbf{S}\text{-AutStr}$  and consider an injective presentation  $\mathfrak{d}$  with naming function  $f$ . Let  $\mathcal{F}$  be a finite set of functions FO-definable in  $\mathfrak{A}$  and  $E = \{e_0, e_1, \dots\}$  a definable set of elements ordered according to length in  $\mathfrak{d}$ , i.e.  $|f^{-1}(e_0)| \leq |f^{-1}(e_1)| \leq \dots$ . Then there is a constant  $k$  such that for every  $n$  and for every  $a \in G_{\mathcal{F}}^n$   $|f^{-1}(a)| \leq kn$ . In particular,  $|G_{\mathcal{F}}^n| = 2^{\mathcal{O}(n)}$ .*

In other words, the number of elements that can be generated using functions is at most a single exponential in the number of iterations. Continuing the previous examples, neither the free semigroup nor any bijection  $f : D \times D \rightarrow D$  (also called a pairing function) is word-automatic. It is trickier to apply the proposition to show that Skolem arithmetic  $(\mathbb{N}, \times)$  is not word-automatic (see [20, 25]). It is nevertheless tree-automatic, cf. Example 1.3.9.

The application of propositions 1.4.13 and 1.4.15 has been pushed to their limits:

- Proposition 1.4.16** (i) *If a group  $(G, \cdot)$  is word-automatic then every finitely generated subgroup is virtually Abelian (has an Abelian subgroup of finite index). In particular, a finitely generated group is in  $\mathbf{S}\text{-AutStr}$  if, and only if, it is virtually Abelian [116, 114].*
- (ii) *A Boolean Algebra (in the signature  $(\cup, \cap, \cdot^c, \perp, \top)$ ) is in  $\mathbf{S}\text{-AutStr}$  if, and only if, it is a finite power of the Boolean Algebra of finite or co-finite subsets of  $\mathbb{N}$  [88]. In particular, the countable atomless Boolean Algebra is not in  $\mathbf{S}\text{-AutStr}$ .*
- (iii) *There is no infinite integral domain in  $\mathbf{S}\text{-AutStr}$  [88].*
- (iv) *No word-automatic structure  $(D, R)$  has a subset  $N \subset D$  such that  $(N, R)$  is isomorphic to  $(\mathbb{N}, \cdot)$ , cf. [114].*

The proof of the first item starts with the observation that every finitely-generated group  $G \in \mathbf{S}\text{-AutStr}$  has polynomial density - that is, for every finite set  $A = \{a_1, \dots, a_k\}$  the function

$$\gamma(n) = |\{\prod_{i < n} c_i^{\sigma_i} \mid \forall i < n : c_i \in A, \sigma_i \in \{1, -1\}\}|$$

is bounded by a polynomial (this exploits associativity of the group operation). The rest of the proof uses powerful theorems of Gromov and Ershov (see [114] for a survey of word-automatic groups).

#### Number of definable subsets

Various countable random structures, such as the random graph, do not have word- or tree-automatic presentations [88, 63]. The approach

to proving these facts has a model-theoretic flavour: for a purported automatic presentation, it involves counting the number of definable subsets of elements represented by words of bounded length.

Consider the usual definition of a set defined by  $\varphi$  with parameter  $b$  that remains fixed:

$$\varphi(-, b)^{\mathfrak{A}} = \{a \in \mathfrak{A} \mid \mathfrak{A} \models \varphi(a, b)\}.$$

A finite set  $X \subset A$  is *fully shattered by  $\varphi$*  if the cardinality of the family

$$\{\varphi(-, b)^{\mathfrak{A}} \cap X \mid b \in A\}$$

is as large as possible, namely  $2^{|X|}$ . For instance, Benedikt et al. [16] observe that in  $\mathcal{S}_{[2]}$  each of the sets  $\{0, 00, \dots, 0^n\}$  can be fully shattered by the formula  $\varphi(x, b) = \exists z (\mathbf{suc}_1 z \prec_{\text{prefix}} b \wedge \mathbf{el}(z, x))$ .

By contrast, in every automatic presentation with naming function  $f$  and domain  $D \subseteq \Sigma^*$ , the image under  $f$  of each  $D_{\leq n} := D \cap \Sigma^{\leq n}$  can only be linearly shattered by definable families.

**Proposition 1.4.17** ([88, 63]) *In every automatic presentation of a structure  $\mathfrak{A}$  with naming function  $f$  and for every formula  $\varphi$ :*

$$|\{\varphi(-, b)^{\mathfrak{A}} \cap f(D_{\leq n}) \mid b \in A\}| = \mathcal{O}(|f(D_{\leq n})|).$$

As an application recall that the random graph is characterised by the property that for every partition of a finite set  $X$  of vertices into sets  $U$  and  $V$ , there is a vertex  $b$  connected to all elements of  $U$  and to no element of  $V$ . In other words, every finite set  $X$  of vertices is fully shattered by the edge relation as the parameter  $b$  is varied. So by Proposition 1.4.17 the random graph has no word-automatic presentation. Similar reasoning yields the following.

**Proposition 1.4.18** ([88, 63]) *The following are not in  $\mathbf{S}\text{-AutStr}$ : the random graph, the random partial order, the random  $K_n$ -free graph.*

Using Theorem 1.3.25 one can establish non-automaticity of the random graph in a far more general sense.

**Theorem 1.4.19** ([47]) *Neither the random graph nor the free monoid on two generators is finite-tree automatic with any oracle.*

In fact neither is  $\omega$ -word automatic with any oracle, as witnessed by the following theorem which follows from the proof of Theorem 1.3.31.

**Theorem 1.4.20** *If a countable structure is  $\omega$ -word automatic with oracle, then it is also finite-word automatic with (the same) oracle.*

#### 1.4.4 Comparing presentations

When we think of an automatic structure we frequently have a particular automatic presentation in mind. Some structures have *canonical* presentations. For instance,  $(a^*, \leq_{\text{len}})$  is arguably the canonical presentation of  $(\mathbb{N}, <)$  and  $(\{0, 1\}^*, \mathbf{suc}_0, \mathbf{suc}_1, \prec_{\text{prefix}}, \mathbf{e1})$  is the canonical presentation of itself. Some well-known structures have *natural* presentations, none of which can be indisputably called canonical. The base  $k \in \mathbb{N}$  ( $k > 1$ ) presentations of  $(\mathbb{N}, +)$  can be considered equally natural; but then what about the Fibonacci numeration system? The field of *regular numeration systems*, though using a somewhat different terminology, investigates automatic presentations of  $(\mathbb{N}, +)$  and  $\omega$ -word automatic presentations of  $(\mathbb{R}, +)$ . Finally, there are *pathological* presentations that are used to pin down the relationship between definability in a structure and regularity in its presentations [90].

How are we to compare different automatic presentations of the same structure? What are the crucial aspects of a presentation that distinguish it from others?

Canonical representations of context-free graphs were investigated by Séniergues. In [127] a p-structure for a graph  $G$  is a PDA  $\mathcal{A}$  (having no  $\epsilon$ -transitions) together with an isomorphism between the configuration graph of  $\mathcal{A}$  and  $G$ . Furthermore, a p-structure for  $G$  is *P-canonical* if the distance in  $G$  between a vertex  $v$  and the root is equal to the stack height of the configuration representing  $v$  (cf. [112]'s notion of a canonical automaton for a context-free graph; and [41, 44]). For a fixed graph  $G$  Séniergues considers two p-structures equivalent if there is a rational isomorphism between them, and shows that every equivalence class of p-structures contains a P-canonical one [127].

An example from the theory of numeration systems is provided by the celebrated result of Cobham and Semenov. Recall that naturals  $p$  and  $q$  are called *multiplicatively independent* if they have no common power (ie.  $p^k \neq q^l$  for all  $k, l \geq 1$ ) and *multiplicatively dependent* otherwise.

**Theorem 1.4.21** (Cobham-Semenov <sup>19</sup>, cf. [26, 19, 109])

The following dichotomy holds for  $p, q \geq 2$ .

- (i) If  $p$  and  $q$  are multiplicatively dependent then a relation  $R \subseteq \mathbb{N}^r$  is regular when coded in base  $p$  iff it is regular when coded in base  $q$ .
- (ii) If  $p$  and  $q$  are multiplicatively independent then a relation  $R \subseteq \mathbb{N}^r$  is regular in both base  $p$  and base  $q$  iff  $R$  is FO-definable in  $(\mathbb{N}, +)$ .

<sup>19</sup> Cobham proved it for sets; Semenov later extended it to arbitrary relations.

The meaning of (i) is that, for instance, bases  $2^l$  and  $2^k$  are expressively equivalent. There is a very simple coding translating numerals between these bases, which bijectively maps blocks of  $k$  digits in the first system to blocks of  $l$  digits in the second system. Every pair of multiplicatively dependent numeration systems are linked by similar translations.

According to (ii) the base  $2^k$  presentation is as different as it can be from, say, the base 3 presentation. This point is further stressed by the following result of Bés based on the work of Michaux and Villemaire.

**Theorem 1.4.22** ([18]) *Let  $p$  and  $q$  be multiplicatively independent, and  $R \subseteq \mathbb{N}^r$  regular when coded in base  $q$ , but not first-order definable in  $(\mathbb{N}, +)$ . Then the first-order theory of  $(\mathbb{N}, +, |_p, R)$  is undecidable.*

On a similar note we introduce the following general notions.

**Definition 1.4.23** (Subsumption and equivalence)

Consider two  $\square$ -automatic presentations of some structure  $\mathfrak{A}$  with naming functions  $f$  and  $g$ , respectively. We say that  $f$  *subsumes*  $g$  ( $g \preccurlyeq f$ ) if for every relation  $R$  over the domain of  $\mathfrak{A}$ , if  $g^{-1}(R)$  is  $\square$ -regular then  $f^{-1}(R)$  is  $\square$ -regular. If both  $f \preccurlyeq g$  and  $g \preccurlyeq f$  then we say that the two presentations are *equivalent* and write  $f \sim g$ . Moreover, we say that a  $\square$ -automatic presentation of  $\mathfrak{A}$  is *prime* if it is subsumed by all other  $\square$ -automatic presentations of  $\mathfrak{A}$ .

### word-automatic presentations

The definition of equivalence of automatic presentations is modelled on case (i) of Theorem 1.4.21. In [5] it has been shown that two finite-word automatic presentations are equivalent if and only if the transduction translating names of elements from one presentation to the other is computable by a *semi-synchronous transducer*: a two-tape finite automaton processing its first tape in blocks of  $k$  letters and its second tape in blocks of  $l$  letters for some fixed positive  $k$  and  $l$ . (Note that, except in trivial cases,  $k/l$  is uniquely determined [5].)

**Theorem 1.4.24** ([5]) *Two finite-word automatic presentations of some  $\mathfrak{A} \in \mathbf{S}\text{-AutStr}$  with naming functions  $f_i : D_i \rightarrow A$ ,  $i \in \{1, 2\}$ , are equivalent if, and only if, the transduction  $T = \{(x, y) \in D_1 \times D_2 \mid f_1(x) = f_2(y)\}$  translating names of elements from one presentation to the other is semi-synchronous rational.*

**Corollary 1.4.25** *Let  $f_1$  and  $f_2$  be naming functions of equivalent automatic presentations of  $\mathfrak{A}$ . Then there is a constant  $C$  such that*

for every  $n$ -ary relation  $R$  over  $\text{dom}(\mathfrak{A})$  and for every automaton  $\mathcal{A}_1$  recognising  $f_1^{-1}(R)$  there is an automaton  $\mathcal{A}_2$  of size  $|\mathcal{A}_2| \leq C^n \cdot |\mathcal{A}_1|$  recognising  $f_2^{-1}(R)$ , and vice versa.

Let  $\mathfrak{U}$  be one of the universal finite-word automatic structures  $\mathcal{S}_\Sigma$  (for  $|\Sigma| > 1$ ),  $\mathcal{P}_f(\Delta_1)$ , or  $(\mathbb{N}, +, |_k)$  (for  $k > 1$ ). Using semi-synchronous translations one can establish the following.

**Theorem 1.4.26** ([5, 6]) *The universal structure  $\mathfrak{U}$  has only a single word-automatic presentation up to equivalence.*

The assertion of the theorem can be reformulated as follows.

**Corollary 1.4.27** *For a relation  $R$ , the expansion  $(\mathfrak{U}, R)$  is in  $\mathbf{S}\text{-AutStr}$  if, and only if,  $R$  is FO-definable in  $\mathfrak{U}$ .*

The prime presentation of a structure, if one exists, is unique up to equivalence, hence may as well be called canonical. The unary presentation of  $(\mathbb{N}, <)$  is a prime word-automatic presentation. It is, however, not a prime presentation of  $(\mathbb{N}, \text{suc})$ , which allows, for every  $m > 1$  a word-automatic presentation in which divisibility by  $m$  is not regular [90]. It can be inferred that  $(\mathbb{N}, \text{suc})$  has no prime presentation.

Recall Theorem 1.3.27 stating that each word-automatic presentation, of structure  $\mathfrak{A}$ , over a domain of polynomial density of degree  $d$  directly corresponds to a  $d$ -dimensional interpretation of  $\mathfrak{A}$  in the structure  $\mathfrak{M} = (\mathbb{N}, <, \{\equiv_{(\text{mod } m)}\}_{m>1})$ , and hence also in  $(\mathbb{N}, +)$ . So every p-automatic structure has infinitely many pairwise incomparable word-automatic presentations ‘inherited’ from  $(\mathbb{N}, +)$ , namely, based on different numeration systems.

In fact,  $\mathfrak{M}$  allows a non-trivial 2-dimensional interpretation in itself. Simply consider the lexicographic ordering of all pairs  $(n_1, n_2)$  such that  $n_1 \geq n_2$  as an interpretation of  $(\mathbb{N}, <)$  and observe that moduli of positions within the lexicographic ordering of tuples can be expressed in terms of moduli of their components. Thus, by composing interpretations, every p-automatic presentation of  $\mathfrak{M}$  is properly subsumed by other p-automatic presentations with domains of asymptotically greater polynomial densities. This carries over to all p-automatic structures.

In contrast, from results of [5, 8] it follows that  $g \preccurlyeq f$  implies  $g \sim f$  for any two word-automatic presentations of a given structure, provided that either both  $f$  and  $g$  have domains of exponential density, or both have a domain of polynomial density of the same degree.

Therefore, the height of the partial order of word-automatic presen-

tations of  $\mathfrak{A}$  under subsumption and modulo equivalence is  $\omega$  if  $\mathfrak{A}$  is p-automatic and 1 if  $\mathfrak{A}$  is not p-automatic. It is not known whether the width of the subsumption order modulo equivalence is always one or infinite for word-automatic structures that are not p-automatic.

#### tree-automatic presentations

Colcombet and Löding [47] investigated the power of finite-subset interpretations applied to arbitrary trees. In our terminology these are tree-automatic presentations with arbitrary oracles.

In the tree-automatic model the analogue of Theorem 1.4.26 does not hold. A tree-automatic presentation of  $\mathcal{P}_f(\Delta_2)$  incomparable with the natural one can be forged simply by ‘folding each tree in half about the vertical axis’, i.e. taking the mirror image of the subtree below the right child of the root and smoothly combing it together with the untouched left half, e.g. as in Example 1.3.11(3). Despite this, the fact concerning primality of the natural presentation of the universal structure holds in an even stronger sense.

**Proposition 1.4.28** ([47, Lemma 5.6]) *The natural tree-automatic presentation with oracle  $O$  and with the identity naming function of the finite-subset envelope  $\mathcal{P}_f(\mathfrak{T}_O)$  of the oracle tree  $\mathfrak{T}_O$  is a prime presentation with respect to tree-automatic presentations with arbitrary oracle.*

In particular, ‘the’ word-automatic presentation of  $\mathcal{P}_f(\Delta_1)$  and the natural tree-automatic presentation of  $\mathcal{P}_f(\Delta_2)$  are both prime even among tree-automatic presentations with arbitrary oracles. This is complemented by the following result of [47].

**Theorem 1.4.29** *All tree-automatic presentations of  $\mathcal{P}_f(\Delta_1)$  are equivalent.*

Therefore, the same holds true for all of the universal structures from Theorem 1.4.26.

#### 1.4.5 Other notions of automaticity

Specific automatic presentations have been employed in other mathematical fields: computational group theory [31], symbolic dynamics [13], numeration systems (of integers or reals) [76], and infinite sequences represented in natural numeration systems [2, 26, 4]. In this section we survey natural presentations of certain structures that have mostly been considered independently of the general theory of automatic structures.

### Automatic groups

Thurston (1986) motivated by work of Cannon on hyperbolic groups introduced the notion of automatic groups. A finitely generated group  $G$  is *automatic* in this sense if for some set of semigroup generators  $S$  and associated canonical homomorphism  $f : S^* \rightarrow G$

- (i) there is a regular language  $W \subset S^*$  so that  $f$  restricted to  $W$  is surjective,
- (ii) for every  $s$  a generator from  $S$  or the group identity, the following binary relation over  $W$  is regular:

$$\{(u, v) \mid f(u) = f(v)s\}.$$

This is in fact an algebraic notion: it does not depend on the particular choice of generators. From the automata presenting the group one can extract a finite presentation of the group, and a quadratic-time algorithm deciding the word problem.

**Proposition 1.4.30** ( $k$ -fellow traveler property) *A group  $G$  with semigroup generators  $S = \{s_1, \dots, s_r\}$  is automatic if, and only if, there exists a regular set  $W \subseteq S^*$  and  $k \in \mathbb{N}$  such that  $f|_W$  is surjective and  $W$  satisfies the  $k$ -fellow traveler property:*

$$\forall u, v \in W \text{ with } d(u, v) \leq 1 \quad \forall i \leq \max\{|u|, |v|\} : d(u_1 \dots u_i, v_1 \dots v_i) \leq k$$

where  $d(u, v)$  denotes the length of the shortest path between  $u$  and  $v$  in the Cayley graph of  $G$  with generators  $S$ .

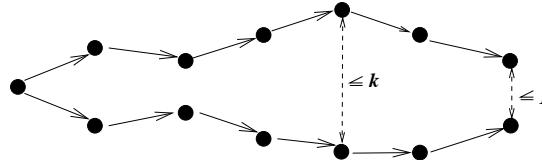


Figure 1.4  $k$ -fellow traveler property.

Virtually Abelian groups and Gromov's word hyperbolic groups constitute important examples of automatic groups in this sense. Major results of this programme are presented in [31] (see also the introductions by Farb [71] and by Choffrut [46]).

More recently, this notion has been extended to semigroups [29, 30, 84, 28] and monoids [83, 129, 102].

Let us compare the following three notions: (i) groups whose multiplication function admits a word-automatic presentation, (ii) finitely generated automatic groups, and (iii) finitely generated groups with a Cayley graph admitting a word-automatic presentation. It is known [116] that a finitely generated group allows a word-automatic presentation of type (i) iff it is virtually Abelian. All virtually Abelian finitely generated groups are automatic in the sense of this subsection. Hence (i) implies (ii) for finitely generated groups. Furthermore, by definition, the Cayley graph of every automatic group has a word-automatic presentation. Hence (ii) implies (iii), but the converse fails. As Séniérgues has pointed out the Heisenberg group is not automatic even though its Cayley graph has an automatic presentation. For further reading we recommend the survey by Nies [114].

### Generalised numeration systems

The theory of *generalised numeration systems* [76] is concerned with representations of  $\mathbb{N}$  and  $\mathbb{R}$  in various bases and using different (possibly negative) digits. In general, the basis  $U_0 < U_1 < U_2 < \dots$  of the system does not have to be the sequence of powers of a natural. One considers bases satisfying appropriate linear recursions, or alternatively powers of a base  $\beta$  which is the greatest root of a polynomial of a certain type. The study of generalised numeration systems goes back to Rényi who in 1957 introduced  $\beta$ -expansions.

Without going into the particulars of this very rich field we point out that a number may have more than one representation in a given numeration system. Thus from a practical perspective one is interested in *normalised* numerals obtained via the *greedy* algorithm. Normalised numerals are ordered according to  $<_{\text{lex}}$  (length and then lexicographically, most significant digit first). A regular set of (normalised) numerals  $N \subseteq [d]^*$  over the set of digits  $0, \dots, d - 1$  is simply an automatic copy of  $(\mathbb{N}, <)$  of the form  $(N, <_{\text{lex}})$ .

A fundamental question in this context asks under which circumstances addition can be computed by a synchronous finite automaton. When this is the case one speaks of a *regular numeration system*. On this matter we refer to [76] and the references therein.

**Example 1.4.31** The *Fibonacci numeration system* is a prominent example of a regular numeration system. It has the Fibonacci numbers  $1, 2, 3, 5, 8, \dots$  as its basis, and the binary digit set. The normalised numerals delivered by the greedy algorithm are  $\varepsilon, 1, 10, 100, 101, 1000,$

1001, 1010, 10000, 10001, ... in the length-lexicographic ordering. They are the binary strings avoiding 11 as a factor since greedy normalisation prefers 100 to 11. Naturally,  $10^n$  represents the  $n$ th Fibonacci number.

More generally we ask how can one classify the word-automatic presentations of  $(\mathbb{N}, +)$ ? Or those of  $(\mathbb{N}, <)$ ? Below we survey known classes of automatic presentations of expansions of  $(\mathbb{N}, <)$  by unary predicates, i.e. infinite sequences.

### Automatic sequences

The theory of *automatic sequences* [2] studies  $\omega$ -words representable in more-or-less standard numeration systems. Presentations of primary concern are those of base  $k \in \mathbb{N}$ , or of base  $-k$ , and possibly involving negative digits.

**Definition 1.4.32** A sequence  $s : \mathbb{N} \rightarrow \Sigma$  is *k-automatic* if for every  $a \in \Sigma$  the set  $N_a$  of numerals in the standard base  $k$  numeration system representing all positions  $n$  such that  $s(n) = a$  constitutes a regular language.

These *k-automatic sequences* have been characterised in both algebraic and logical terms. In order to formulate another characterisation some notions are required. A morphism  $\varphi : \Gamma^* \rightarrow \Sigma^*$  is said to be *k-uniform* if  $|\varphi(a)| = k$  for each  $a \in \Gamma$ . *Codings* are 1-uniform morphisms. A morphism  $\varphi : \Gamma^* \rightarrow \Gamma^*$  is *prolongable* on some  $a \in \Gamma$  if  $a$  is the first symbol of  $\varphi(a)$ . In this case the sequence  $(\varphi^n(a))_{n \in \mathbb{N}}$  converges to either a finite or infinite word, which is a fixed point of  $\varphi$ , denoted  $\varphi^\omega(a)$ .

**Theorem 1.4.33** ([26, 2]) *For any sequence  $s : \mathbb{N} \rightarrow \Sigma$  the following are equivalent:*

- (1) *s is k-automatic;*
- (2) *the k-kernel of s:  $\{(s_{nk^m+r})_n \mid r, m \in \mathbb{N}, r < k^m\}$  is finite;*
- (3) *the sets  $s^{-1}(a)$  are FO-definable in  $(\mathbb{N}, +, |_k)$  for each  $a \in \Sigma$ ;*
- (4)  *$s = \sigma(\tau^\omega(a))$  for some k-uniform morphism  $\tau$  on some  $\Gamma^*$  and a coding  $\sigma : \Gamma \rightarrow \Sigma$ ;*
- (5) *(assuming k is a prime and  $\Sigma \subseteq \{0, \dots, k-1\}\}): the formal power series  $S(x) = \sum_n s_n x^n \in \mathbb{F}_k[[x]]$  is algebraic over  $\mathbb{F}_k[x]$ .$*

For example, consider the morphism  $\tau : 0 \mapsto 01, 1 \mapsto 10$ . Its fixed point  $\tau^\omega(0)$  is the *Thue-Morse sequence*  $t = 01101001100101101001\dots$ . This is a truly remarkable sequence bearing a number of characterisations and combinatorial properties [3]. For instance, its  $n$ th digit is 1 if, and only

if, the binary numeral of  $n$  contains an odd number of 1's. The 2-kernel of  $t$  is  $\{t, \bar{t}\}$ , where  $\bar{t}$  is obtained from  $t$  by flipping every bit.

### Morphic words

One obtains a definition of *morphic words* by relaxing characterisation (4) of the above theorem. Morphic words thus constitute a generalisation of automatic sequences. They and their relatives have been extensively studied in the context of formal language theory, Lindenmayer systems and combinatorics on words.

**Definition 1.4.34** *Morphic words* are those of the form  $\sigma(\tau^\omega(a))$  for arbitrary homomorphism  $\tau$  prolongable on  $a$  and arbitrary homomorphism  $\sigma : \Gamma^* \rightarrow \Sigma^*$  extended to  $\omega$ -words in the obvious way.

**Example 1.4.35** Consider  $\tau : a \mapsto ab, b \mapsto ccb, c \mapsto c$  and  $\sigma : a, b \mapsto 1, c \mapsto 0$  both homomorphically extended to  $\{a, b, c\}^*$ . The fixed point of  $\tau$  starting with  $a$  is the word  $abccbccccbc^6b\dots$ , and its image under  $\sigma$ ,  $110010^410^610^81\dots$ , is the characteristic sequence of the set of squares. In general, for every strictly positive N-rational sequence  $(s_k)$  the characteristic sequence of the set  $\{\sum_{k=0}^n s_k \mid n \in \mathbb{N}\}$  is morphic [38]. This result also follows from Proposition 1.4.37.

While  $k$ -automatic sequences allow automatic presentations over the set of standard base  $k$  numerals, the above example suggests that morphic words may need generalised numeration systems. Indeed, every morphic word is automatically presentable in the following sense.

Consider a finite ordered alphabet  $\Gamma = \{a_1 < a_2 < \dots < a_r\}$ . In the induced *length-lexicographic order*, denoted  $<_{\text{llex}}$ , words over  $\Gamma$  are ordered according to their length first, while words of the same length are ordered lexicographically. Thus  $(D, <_{\text{llex}})$  provides an automatic presentation of  $(\mathbb{N}, <)$  for every infinite regular language  $D$  over  $\Gamma$ . Base  $k$  as well as so called generalised numeration systems are special cases of this scheme. The following notion thus generalises Definition 1.4.32.

**Definition 1.4.36** We say that an  $\omega$ -word  $w : \mathbb{N} \rightarrow \Sigma$  is *length-lexicographically presentable* if there is an automatic presentation  $(D, <_{\text{llex}})$  of  $(\mathbb{N}, <)$  with naming function  $f : D \rightarrow \mathbb{N}$  such that the sets  $f^{-1}(w^{-1}(a))$  are regular for each  $a \in \Sigma$ .

It is not hard to see that an  $\omega$ -word is length-lexicographically presentable if and only if it is morphic. There is a perfectly natural correspondence between the morphisms generating a word and the automaton

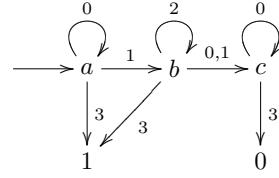
recognising the set of ‘numerals’, which, when length-lexicographically ordered, give an automatic presentation of the morphic word.

**Proposition 1.4.37** ([122]) *An  $\omega$ -word  $w$  is length-lexicographically presentable if, and only if,  $w$  is morphic.*

We illustrate the transformation from one formalism to the other on the characteristic sequence of squares from Example 1.4.35. Recall that it is generated by the following morphism  $\tau$  and final substitution  $\sigma$

$$\begin{aligned}\tau : \quad a &\mapsto ab & b &\mapsto ccb & c &\mapsto c \\ \sigma : \quad a &\mapsto 1 & b &\mapsto 1 & c &\mapsto 0\end{aligned}$$

The idea is to interpret symbols  $\{a, b, c, 0, 1\}$  as states. Without loss of generality, the alphabets of the ranges of  $\sigma$  and  $\tau$  are disjoint. The alphabet  $\Gamma$  of the automatic presentation consists of digits ranging from 0 to  $|\tau| + |\sigma| - 1$ , where  $|\tau|$  is the maximum of  $|\tau(x)|$  with  $x \in \{a, b, c\}$  and  $|\sigma|$  is defined similarly. Letters of the alphabet, ordered as usual, are used to index positions within the right-hand side of a  $\tau$ -rule, or, when larger, positions inside the right-hand side of a substitution via  $\sigma$ .



The domain  $D$  of the presentation is recognised by the above automaton with both 1 and 0 as final states. With only 1 as a terminal state, the automaton recognises the numerals representing a square relative to the length-lexicographic enumeration of  $D$ . Starting with a deterministic automaton this transformation can be reversed producing a morphism  $\tau$  representing the transition function linearised according to the ordering on the alphabet and with  $\sigma$  identified by the terminal states.

The MSO-theory of the structure  $(\mathbb{N}, <, (w^{-1}(a))_a)$  for morphic  $w$  is decidable [38]. Moreover, the class of morphic words is closed under MSO-definable recolourings, i.e. under *deterministic generalised sequential mappings* [118]. These results are generalised by the following one, which can be seen as an extension of the Fundamental Theorem 1.3.4.

**Theorem 1.4.38** ([7]) *Let  $\mathfrak{d} = (D, <_{\text{llex}}, \overline{P})$  be a length-lexicographic presentation of a morphic word  $w$  and let  $\varphi(\overline{x})$  be an  $\text{MSO}[<, \overline{P}]$ -formula*

having only first-order variables free. Then there is an automaton  $\mathcal{A}$ , computable from  $\mathfrak{d}$  and  $\varphi$  and such that  $(\mathfrak{d}, \mathcal{A})$  is a word-automatic presentation of  $w$  expanded by the relation defined by  $\varphi$ .

Caucal has shown that morphic sequences can be constructed as graphs on the second level of the pushdown hierarchy (cf. Definition 1.2.14) [43]. However, there are automatically presentable  $\omega$ -words on higher levels as well.

### Higher-order morphic words

*Higher-order morphic words* were introduced in [4, 7]. Morphic words of order  $k$  can be defined either in the style of Definition 1.4.34 based on a notion of ‘morphisms of order- $k$  stacks’ or similar rules, or as in Definition 1.4.36 as those having an automatic presentation using the ‘ $k$ -fold nested length-lexicographic order’ induced by an ordered alphabet. Theorem 1.4.38 extends to these automatic presentations of higher-order morphic words. The classes of order  $k$  morphic words form an infinite hierarchy, and are constructible on the  $2k$ -th level of the pushdown hierarchy [7].

**Example 1.4.39** As an example we mention the Champernowne word (cf. Example 1.3.23) obtained by concatenating decimal numerals in their usual order:

$$C = 1234567891011121314\dots$$

It is on the second level of this hierarchy (and on the fourth level of the pushdown hierarchy). Consider the level 2 morphism  $\Delta$  given by the following intuitive production rules

$$\begin{aligned} S_x &\rightarrow S_x A_{\tau_1(x)} \dots A_{\tau_9(x)} \\ A_x &\rightarrow A_{\tau_0(x)} A_{\tau_1(x)} \dots A_{\tau_9(x)} \end{aligned}$$

where each  $\tau_i$  is a (level 1) morphism of words in the usual sense mapping each digit  $d \in \{0, \dots, 9\}$  to  $d$  and  $\#$  to  $i\#$ . Applying  $\Delta$  repeatedly to the initial level 2 stack  $S_\#$  yields the following converging sequence

$$\begin{aligned} S_\# &\rightarrow S_\# A_{1\#} A_{2\#} \dots A_{9\#} \\ &\rightarrow S_\# A_{1\#} A_{2\#} \dots A_{9\#} A_{10\#} \dots A_{19\#} \dots \dots A_{90\#} \dots A_{99\#} \\ &\rightarrow \dots \end{aligned}$$

Hence  $C$  can be specified as  $C = \sigma(\Delta^\omega(S_\#))$  with the morphism  $\sigma$  erasing all  $\#$ 's while preserving the other (level 1) symbols.

To give a word-automatic presentation we take the domain  $D$  to be

comprised of all words of the form  $d_1m_1d_2m_2\dots d_sm_s$  with  $d_1d_2\dots d_s$  a conventional decimal numeral and  $m_1m_2\dots m_s = o^i x o^{s-i-1}$  a marker indexing the  $i$ th digit of this numeral. Elements of the domain are ordered using the length-lexicographic ordering in a nested fashion: comparing numerals (i.e. odd positions) first, and then according to the position of the marker  $x$ .

The Champernowne word contains every finite word over  $\{0, 1, \dots, 9\}$  as a factor. The satisfiability problem of first-order logic on finite words, known to be non-elementary [79], is thus expressible in the FO theory of the Champernowne word, which is therefore also non-elementary. For the same reason the Champernowne word is not morphic. Every morphic word is MSO-definable in the Champernowne word, and every word-automatic equivalence structure having only finitely many infinite equivalence classes is interpretable in a second-order morphic word [7].

**Proposition 1.4.40** *Consider  $\mathfrak{A} = (A, E)$  with  $E$  an equivalence relation having, for each  $n > 0$ ,  $f(n) \in \mathbb{N}$  many equivalence classes of size  $n$ , and no infinite classes. Then  $\mathfrak{A} \in \text{S-AutStr}$  if, and only if, there is a second-order morphic word  $w = 0^{m_0}10^{m_1}10^{m_2}1\dots$  such that  $f(n) = |\{i \mid m_i = n\}|$ .*

It remains open whether the decidability and definability results for MSO hold for all word-automatic infinite sequences. We are intrigued whether the isomorphism problem of automatic  $\omega$ -words, or more broadly for automatic scattered linear orders, is decidable. Already for morphic words this is a notorious long-standing open problem.

## 1.5 Automatic Model Theory

We may reformulate the original problem — we seek a class of finitely-presentable structures  $\mathcal{C}$  that has an interesting model theory and lies somewhere between the finite structures (finite model theory) and all structures (classical model theory).

The richest and oldest class consists of the computable structures — these are structures whose domain and atomic relations are computable by Turing machines [70]. In computable model theory, a common theme is to take classical results from mathematics and model theory and to see to what extent they can be made effective. Here are two illustrative observations:

- (i) A computable (consistent) first-order theory has a computable model. Indeed, Henkin's construction can be seen as an algorithm computing the domain and atomic relations.
- (ii) Every two computable presentations of the rational ordering  $(\mathbb{Q}, <)$  are computably isomorphic. Again, the standard back-and-forth argument can be seen as an algorithm building the isomorphism.

The program of feasible mathematics in the 1980's included the development of polynomial-time model theory [45]. However, every relational computable structure is isomorphic (in fact computably isomorphic) to a polynomial-time structure. Automatic structures can be seen as a further restriction of this class, and in fact this is the motivation in [87]. In this section we discuss some aspects of the model theory of automatic structures, a subject still in its infancy.

We split our discussion along two lines: model theory of the class  $\mathbf{S}\text{-AutStr}$ , and model theory of the particular universal structure  $\mathcal{S}_{[2]}$  (cf. Theorem 1.3.17).

### 1.5.1 Model theory restricted to the class of word-automatic structures

Blumensath shows that, as expected, certain notions of model theory fail when restricted to the class of automatic structures.

- Proposition 1.5.1**
- (i) *It is undecidable whether an FO-formula has a word-automatic model.*
  - (ii) *The following properties fail on the class of word automatic structures: compactness, Beth, Interpolation, and Los-Tarski.*

The proofs are based on the observation that there is a FO formula which has automatic models of every finite cardinality but no infinite automatic models.

#### Löwenheim-Skolem

An automatic version of the Downward Löwenheim-Skolem Theorem would say that every uncountable  $\omega$ -automatic structure has a countable elementary substructure that is also  $\omega$ -automatic. Unfortunately this is false since there is a first-order theory with an  $\omega$ -automatic model but no countable  $\omega$ -automatic model. Indeed, consider the first-order theory of atomless Boolean Algebras. Kuske and Lohrey [94] have observed that it

has an uncountable  $\omega$ -automatic model (namely the algebra from Example 1.3.12.4). However, Khoussainov et al. [88] show that the countable atomless Boolean algebra is not automatic and so, by Theorem 1.4.20, not  $\omega$ -automatic either.

Here is the closest we can get to an automatic Downward Löwenheim-Skolem Theorem for  $\omega$ -automatic structures.

**Proposition 1.5.2** ([85]) *Let  $(D, \approx, \{R_i\}_{i \leq \omega})$  be an omega-automatic presentation of  $\mathfrak{A}$  and let  $\mathfrak{A}_{up}$  be its restriction to the ultimately periodic words of  $D$ . Then  $\mathfrak{A}_{up}$  is a countable elementary substructure of  $\mathfrak{A}$ .*

*Proof* Relying on the Tarski-Vaught criterion for elementary substructures we only need to show that for all first-order formulas  $\varphi(\bar{x}, y)$  and elements  $\bar{b}$  of  $\mathfrak{A}_{up}$

$$\mathfrak{A} \models \exists y \varphi(\bar{b}, y) \Rightarrow \mathfrak{A}_{up} \models \exists y \varphi(\bar{b}, y).$$

By Theorem 1.3.4  $\varphi(\bar{x}, y)$  defines an omega-regular relation and, similarly, since the parameters  $\bar{b}$  are all ultimately periodic the set defined by  $\varphi(\bar{b}, y)$  is omega-regular. Therefore, if it is non-empty, then it also contains an ultimately periodic word, which is precisely what we needed.  $\triangleleft$

An identical proposition, also independently noted by Khoussainov and Nies, holds for  $\mathfrak{A} \in \omega\text{-AutStr}$  with regular trees in place of ultimately periodic words.

Consider the natural, say, binary  $\omega$ -automatic presentation of  $(\mathbb{R}, +)$ . Its restriction to the set of elements represented by ultimately periodic  $\omega$ -words is isomorphic to the additive group of the rationals  $(\mathbb{Q}, +)$ . Tsankov [136] has shown that there is no automatic divisible torsion-free Abelian group (DTAG). Hence the theory of DTAGs is another example of a first-order theory having an uncountable  $\omega$ -automatic model but no countable ( $\omega$ -)automatic models.

### Automatic theorems König's Lemma

König's Lemma says that an infinite finitely-branching tree has an infinite path. We split our discussion of automatic analogues along two lines, depending on whether the signature is that of partial order  $(T, \preceq)$  or successor  $(T, S)$ .

**Theorem 1.5.3** ([91]) *If  $\mathcal{T} = (T, \preceq)$  is an automatic copy of an infinite finitely-branching tree, then  $\mathcal{T}$  has a regular infinite path. That is, there exists a regular set  $P \subseteq T$  where  $P$  is an infinite path of  $\mathcal{T}$ .*

*Proof* Define a set  $P$  as those elements  $x$  such that  $\exists^\infty w[x \prec w]$  and for which every  $y \prec x$  satisfies that

$$\forall z, z' \in S(y)[z \preceq x \Rightarrow z \leq_{lex} z'].$$

Then  $P$  is the length-lexicographically least infinite path of  $\mathcal{T}$  (in the ordering induced by the finite strings presenting the tree).  $\triangleleft$

However, using the 2-Ramsey quantifier we can do more.

**Theorem 1.5.4 ([91])** *If  $\mathcal{T} = (T, \preceq)$  is an automatic copy of a tree with countably many infinite paths, then every infinite path is regular.*

*Proof* Denote by  $E(\mathcal{T}) \subseteq T$  the set of elements of a tree  $\mathcal{T}$  that are on infinite paths. It is definable in  $\mathcal{T}$  using the 2-Ramsey quantifier, so Theorem 1.4.7 gives that  $E(\mathcal{T})$  is regular. Then every isolated path of  $\mathcal{T}$  is regular, since it is definable as  $\{x \in E(\mathcal{T}) \mid p \preceq x\} \cup \{x \in E(\mathcal{T}) \mid x \prec p\}$ , for suitable  $p \in E(\mathcal{T})$ . Replace  $\mathcal{T}$  by its derivative  $d(\mathcal{T})$ , which is also automatically presentable. Since the CB-rank of  $\mathcal{T}$  is finite [91] and  $d^{\text{CB}}(\mathcal{T})(\mathcal{T})$  is the empty tree, every infinite path is defined in this way.  $\triangleleft$

However, automatic successor trees behave more like computable trees:

**Theorem 1.5.5 ([96])** *The problem of deciding, given automata presenting a successor tree  $(T, S)$ , whether or not it has an infinite path, is  $\Sigma_1^1$ -complete.*

The proof consists of a reduction from the problem of whether a non-deterministic Turing machine visits a designated state infinitely often.

We compare with the computable case.<sup>20</sup> Fix the computable presentation of the full binary tree as consisting of the finite binary sequences with the immediate successor relation (so in fact the prefix relation is also computable). To stress this presentation, we refer to the tree as  $2^\omega$ . Similarly fix a natural computable presentation  $\omega^\omega$  of the  $\omega$ -branching tree. A *computable subtree* of either of these trees is a computable prefix-closed subset.

- (i) There is an infinite computable subtree of  $2^\omega$  with no computable infinite path.
- (ii) There is a computable subtree of  $\omega^\omega$  with exactly one infinite path, and this path is not computable.
- (iii) The set of indices of computable subtrees of the binary tree  $2^\omega$  with at least one infinite path is  $\Pi_2^0$ -complete.

<sup>20</sup> Thanks to Frank Stephan for discussions concerning this case.

- (iv) The set of indices of computable subtrees of  $\omega^\omega$  with at least one infinite path  $\Sigma_1^1$ -complete.

### Cantor's Theorems

One of Cantor's theorems says that every countable linear ordering embeds in the rational ordering  $\mathbb{Q}$ . The standard proof is easily seen to be effective given a computable presentation of  $(\mathbb{Q}, <)$ .

There are potentially a variety of automatic versions. The following proposition is the best known.

**Proposition 1.5.6** [93] *Every automatic copy  $\mathcal{M}$  of a linear order can be embedded into some automatic copy of  $\mathbb{Q}$  by a function  $f : \mathcal{M} \rightarrow \mathbb{Q}$  with the following properties:*

- (i) *The function  $f$  is continuous with respect to the order topology.*
- (ii) *The graph of  $f$  is regular.*

It is not known whether there is a single automatic copy of  $\mathbb{Q}$  that embeds, in the sense above, all automatic copies of all automatically presentable linear orders  $\mathcal{M}$ .

Cantor also proved that  $\mathbb{Q}$  is homogeneous: For every two tuples  $x_1 < \dots < x_m$  and  $y_1 < \dots < y_m$  there is an automorphism  $f : \mathbb{Q} \rightarrow \mathbb{Q}$  with  $f(x_i) = y_i$  for  $i \leq m$ . Again there might be a number of automatic variations. Call an automatic copy of  $\mathbb{Q}$  *automatically homogeneous* if for every two tuples there is an automorphism as above that is also regular.

**Proposition 1.5.7** [93] *There is an automatic copy of  $\mathbb{Q}$  that is automatically homogeneous. There is an automatic copy of  $\mathbb{Q}$  that is not automatically homogeneous.*

### Scott ranks

Every countable structure  $\mathcal{A}$  has a sentence of the infinitary logic  $L_{\omega_1, \omega}$  (it allows, in addition to FO, countable disjuncts but still only finitely many free variables) that characterises  $\mathcal{A}$  up to isomorphism. The *Scott rank* of  $\mathcal{A}$  is the minimal quantifier rank amongst all such sentences.

**Theorem 1.5.8** ([86]) *For every computable ordinal there is an automatic structure of Scott Rank at least  $\alpha$ .*

The idea is to massage the configuration space of Turing machines presenting a computable structure (having Scott Rank  $\alpha$ ) to get an automatic structure of similar rank.

### 1.5.2 On the universal word-automatic structure

We conclude by highlighting some model-theoretic properties of the universal structure  $\mathcal{S}_{[2]}$ .

- (i)  $\mathcal{S}_{[2]}$  has infinite VC-dimension [15]. That is, there is a formula  $\phi(x, z)$  that defines a family of sets of the form  $\phi(-, z)^{\mathcal{S}_{[2]}}$  as one varies the parameter  $z$ , and this family fully shatters arbitrarily large finite sets.
- (ii)  $\mathcal{S}_{[2]}$  admits quantifier elimination (QE) in the expansion of all definable unary predicates and binary functions. In fact, no expansion with definable unary functions (and arbitrary predicates) admits QE [15].

Blumensath [20, p. 67] raised the question of whether there are non-standard models of the theory of the universal structure  $\mathcal{S}_{[2]}$  in  $\mathbf{S}\text{-AutStr}$ . Here we sketch an argument resting on Theorem 1.4.26 that shows that there are no word-automatic non-standard models. This result was obtained in discussions with Bakhadyr Khoussainov.

**Theorem 1.5.9**  *$\mathcal{S}_{[2]}$  is the only word-automatic model of its theory.*

*Proof* Assume, for a contradiction, an automatic presentation of a non-standard elementary extension of  $\mathcal{S}_{[2]}$ . By ‘component’ we mean a maximal set of elements connected by successor relations. Every elementary extension of  $\mathcal{S}_{[2]}$  consists of the standard component isomorphic to  $\mathcal{S}_{[2]}$  (containing the root), and any number of non-standard components, that are, as unlabelled graphs, all isomorphic to one-another. The non-standard components are distinguished by the infinite sequences of 0-1 successors ascending towards the root.

(0) *The set of representatives of elements of each component is regular.*

Indeed, the equivalence relation of belonging to the same component is  $\text{FO} + \exists^\infty$ -definable in the model (by saying that there is a common ancestor having finite distance from both elements), hence regular in the representation.

(1) *There is a non-standard element below every standard node.*

This follows from the fact that the formula

$$\forall x, x', y : \text{el}(x, x') \wedge x \prec y \rightarrow \exists y' : \text{el}(y, y') \wedge x' \prec y'$$

being true in  $\mathcal{S}_{[2]}$  must also hold in every non-standard model.

Combining observation (0) and Theorem 1.4.26 we may assume that the presentation restricted to the standard component is the natural one having the identity as naming function. The binary  $\omega$ -sequence naturally associated with an infinite branch of the standard component provides a representation of the set of nodes along that branch consistent with the assumed presentation of the model. Denote by  $\Pi$  the set of paths with a non-standard element below them.

(2) *The set  $\Pi$  is  $\omega$ -regular.*

Indeed, a Büchi-automaton is built to guess a finite word representing a non-standard element and to check, using the automata of the assumed presentation, that it is a descendant of all finite prefixes of the input path. Given that our model is countable, hence so is  $\Pi$ , we have the following consequence of claim (2).

(3) *Every path in  $\Pi$  is ultimately periodic with a period of bounded length.*

To close the circle, consider for each  $n \in \mathbb{N}$  the sentence

$$\forall x \exists y |y| > |x| \wedge 0^n 1 \preceq_{\text{prefix}} y \wedge (\forall z \prec_{\text{prefix}} y)[\text{end}_1(z) \rightarrow z 0^n 1 \preceq_{\text{prefix}} y]$$

where  $\text{end}_1(z)$  is shorthand for saying that the last letter of  $z$  is 1. This sentence expresses that for every length  $|x|$  there is a longer word  $y$  with as many initial prefixes in  $(0^n 1)^*$  as possible. In particular this sentence holds for non-standard elements  $x$ . Consequently,

(4) *for every  $n \in \mathbb{N}$  there is an infinite branch of the standard component with label  $(0^n 1)^\omega$  and having non-standard elements below it.*

This contradicts observation (3).  $\square$

Therefore, by Theorem 1.4.20, there are no countable  $\omega$ -word automatic non-standard models either. Furthermore, using Theorem 1.4.29 in place of Theorem 1.4.26 in the argument shows there are no non-standard finite-tree automatic models of  $S_{[2]}$ . To prove that there are no uncountable  $\omega$ -word automatic non-standard models of  $S_{[2]}$  one tightens (4) and exploits that all automatic presentations of non-standard components are equivalent.

## References

- [1] K. Aehlig, J. G. de Miranda, and C.-H. L. Ong. Safety is not a restriction at level 2 for string languages. In *FoSSaCS*, pages 490–504, 2005.
- [2] J.-P. Allouche and J. Shallit. *Automatic Sequences, Theory, Applications, Generalizations*. Cambridge University Press, 2003.
- [3] J.-P. Allouche and J. O. Shallit. The Ubiquitous Prouhet-Thue-Morse Sequence. In C. Ding, T. Helleseth, and H. Niederreiter, editors, *Sequences and Their Applications: Proceedings of SETA '98*, pages 1–16. Springer-Verlag, 1999.
- [4] V. Bárány. A hierarchy of automatic  $\omega$ -words having a decidable MSO theory. *Journées Montoises '06*, Rennes, 2006.
- [5] V. Bárány. Invariants of automatic presentations and semi-synchronous transductions. In *STACS '06*, volume 3884 of *LNCS*, pages 289–300, 2006.
- [6] V. Bárány. *Automatic Presentations of Infinite Structures*. Phd thesis, RWTH Aachen University, 2007.
- [7] V. Bárány. A hierarchy of automatic  $\omega$ -words having a decidable MSO theory. *R.A.I.R.O. Theoretical Informatics and Applications*, 42:417–450, 2008.
- [8] V. Bárány. Semi-synchronous transductions. *Acta Informatica*, 46(1):29–42, 2009.
- [9] V. Bárány, L. Kaiser, and A. Rabinovich. Eliminating cardinality quantifiers from MLO. Manuscript, 2007.
- [10] V. Bárány, Ch. Löding, and O. Serre. Regularity problems for visibly pushdown languages. In *STACS '06*, volume 3884 of *LNCS*, pages 420–431, 2006.
- [11] K. Barthelmann. On equational simple graphs. Tech. Rep. 9, Universität Mainz, Institute für Informatik, 1997.
- [12] K. Barthelmann. When can an equational simple graph be generated by hyperedge replacement? In *MFCS*, pages 543–552, 1998.
- [13] M.-P. Béal and D. Perrin. Symbolic Dynamics and Finite Automata. In A. Salomaa and G. Rosenberg, editors, *Handbook of Formal Languages, Vol. 2*, pages 463–503. Springer Verlag, 1997.

- [14] M. Benedikt and L. Libkin. Tree extension algebras: logics, automata, and query languages. In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 203–212, 2002.
- [15] M. Benedikt, L. Libkin, Th. Schwentick, and L. Segoufin. A model-theoretic approach to regular string relations. In Joseph Halpern, editor, *LICS 2001*, pages 431–440. IEEE Computer Society, June 2001.
- [16] M. Benedikt, L. Libkin, Th. Schwentick, and L. Segoufin. Definable relations and first-order query languages over strings. *J. ACM*, 50(5):694–751, 2003.
- [17] D. Berwanger and A. Blumensath. The monadic theory of tree-like structures. In E. Grädel, W. Thomas, and T. Wilke, editors, *Automata, Logics, and Infinite Games*, number 2500 in LNCS, chapter 16, pages 285–301. Springer Verlag, 2002.
- [18] A. Bès. Undecidable extensions of Büchi arithmetic and Cobham-Seménov theorem. *Journal of Symbolic Logic*, 62(4):1280–1296, 1997.
- [19] A. Bès. An Extension of the Cobham-Seménov Theorem. *J. of Symb. Logic*, 65(1):201–211, 2000.
- [20] A. Blumensath. Automatic Structures. Diploma thesis, RWTH-Aachen, 1999.
- [21] A. Blumensath. Prefix-Recognisable Graphs and Monadic Second-Order Logic. Technical report AIB-2001-06, RWTH Aachen, 2001.
- [22] A. Blumensath. Axiomatising Tree-interpretable Structures. In *STACS*, volume 2285 of *LNCS*, pages 596–607. Springer-Verlag, 2002.
- [23] A. Blumensath, Th. Colcombet, and Ch. Löding. Logical theories and compatible operations. In J. Flum, E. Grädel, and T. Wilke, editors, *Logic and Automata: History and Perspectives*, Texts in Logic and Games, pages 73–106. Amsterdam University Press, 2007.
- [24] A. Blumensath and E. Grädel. Automatic structures. In *LICS 2000*, pages 51–62. IEEE Computer Society, 2000.
- [25] A. Blumensath and E. Grädel. Finite presentations of infinite structures: Automata and interpretations. *Theory of Comp. Sys.*, 37:641 – 674, 2004.
- [26] V. Bruyère, G. Hansel, Ch. Michaux, and R. Villemaire. Logic and p-recognizable sets of integers. *Bull. Belg. Math. Soc.*, 1:191 – 238, 1994.
- [27] J. R. Büchi. Weak second-order arithmetic and finite automata. *Zeit. Math. Logik Grund. Math.*, 6:66–92, 1960.
- [28] A. J. Cain, E. F. Robertson, and N. Ruskuc. Subsemigroups of groups: presentations, malcev presentations, and automatic structures. *Journal of Group Theory*, 9(3):397–426, 2006.
- [29] C. M. Campbell, E. F. Robertson, N. Ruskuc, and R. M. Thomas. Automatic semigroups. *Theor. Comput. Sci.*, 250(1-2):365–391, (2001).
- [30] C. M. Campbell, E. F. Robertson, N. Ruskuc, and R. M. Thomas. Automatic completely-simple semigroups. *Acta Math. Hungar.*, 96:201–215, 2002.
- [31] J.W. Cannon, D.B.A. Epstein, D.F. Holt, S.V.F. Levy, M.S. Paterson, and W.P. Thurston. *Word processing in groups*. Jones and Barlett Publ., Boston, MA, 1992.

- [32] A. Carayol. Regular sets of higher-order pushdown stacks. In *Proceedings of Mathematical Foundations of Computer Science (MFCS 2005)*, volume 3618 of *LNCS*, pages 168–179, 2005.
- [33] A. Carayol and Th. Colcombet. On equivalent representations of infinite structures. In *ICALP*, volume 2719 of *LNCS*, pages 599–610. Springer, 2003.
- [34] A. Carayol and A. Meyer. Linearly bounded infinite graphs. In *MFCS*, volume 3618 of *Lecture Notes in Computer Science*, pages 180–191. Springer, 2005.
- [35] A. Carayol and A. Meyer. Context-Sensitive Languages, Rational Graphs and Determinism. *Logical Methods in Computer Science*, 2(2), 2006.
- [36] A. Carayol and C. Morvan. On rational trees. In Z. Ésik, editor, *CSL 06*, volume 4207 of *LNCS*, pages 225–239, 2006.
- [37] A. Carayol and S. Wöhrle. The Causal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *FSTTCS*, volume 2914 of *LNCS*, pages 112–123. Springer, 2003.
- [38] O. Carton and W. Thomas. The monadic theory of morphic infinite words and generalizations. *Information and Computation*, 176(1):51–65, 2002.
- [39] A. Caryl and Ch. Löding. MSO on the Infinite Binary Tree: Choice and Order. In *CSL*, volume 4646 of *LNCS*, pages 161–176, 2007.
- [40] D. Caucal. Monadic theory of term rewritings. In *LICS*, pages 266–273. IEEE Computer Society, 1992.
- [41] D. Caucal. On the regular structure of prefix rewriting. *Theor. Comput. Sci.*, 106(1):61–86, 1992.
- [42] D. Caucal. On infinite transition graphs having a decidable monadic theory. In *ICALP'96*, volume 1099 of *LNCS*, pages 194–205, 1996.
- [43] D. Caucal. On infinite terms having a decidable monadic theory. In *MFCS*, pages 165–176, 2002.
- [44] D. Caucal. Deterministic graph grammars. In J. Flum, E. Grädel, and T. Wilke, editors, *Logic and Automata: History and Perspectives*, Texts in Logic and Games, pages 169–250. Amsterdam University Press, 2007.
- [45] D. Cenzer and J.B. Remmel. Complexity-theoretic model theory and algebra. In *Handbook of Recursive Mathematics, Vol. 1*, volume 138 of *Studies in Logic and the Foundations of Mathematics*, pages 381–513. North-Holland, Amsterdam, 1998.
- [46] Ch. Choffrut. A short introduction to automatic group theory, 2002.
- [47] T. Colcombet and C. Löding. Transforming structures by set interpretations. *Logical Methods in Computer Science*, 3(2), 2007.
- [48] Th. Colcombet. On families of graphs having a decidable first order theory with reachability. In *ICALP*, volume 2380 of *LNCS*, pages 98–109. Springer, 2002.
- [49] Th. Colcombet. Equational presentations of tree-automatic structures. In Workshop on Automata, Structures and Logic, Auckland, NZ, 2004.
- [50] Th. Colcombet. *Propriétés et représentation de structures infinies*. Thèse de doctorat, Université Rennes I, 2004.

- [51] Th. Colcombet. A combinatorial theorem for trees. In *ICALP*, volume 4596 of *LNCS*, pages 901–912. Springer, 2007.
- [52] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree Automata Techniques and Applications. In preparation, draft available online at <http://www.grappa.univ-lille3.fr/tata/>.
- [53] B. Courcelle. *Graph algebras and monadic second-order logic*. Cambridge University Press, in writing...
- [54] B. Courcelle. The definability of equational graphs in monadic second-order logic. In *ICALP*, volume 372 of *LNCS*, pages 207–221. Springer, 1989.
- [55] B. Courcelle. Graph rewriting: An algebraic and logic approach. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 193–242. Elsevier and MIT Press, 1990.
- [56] B. Courcelle. Recursive applicative program schemes. In J. v.d. Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B*, pages 459–492. Elsevier and MIT Press, 1990.
- [57] B. Courcelle. The monadic second-order logic of graphs ix: Machines and their behaviours. *Theoretical Computer Science*, 151(1):125–162, 1995.
- [58] B. Courcelle. Finite model theory, universal algebra and graph grammars. In *LFCS '97, Proceedings of the 4th International Symposium on Logical Foundations of Computer Science*, pages 53–55, London, UK, 1997. Springer-Verlag.
- [59] B. Courcelle. The Expression of Graph Properties and Graph Transformations in Monadic Second-Order Logic. In G. Rozenberg, editor, *Handbook of graph grammars and computing by graph transformations, vol. 1: Foundations*, pages 313–400. World Scientific, New-Jersey, London, 1997.
- [60] B. Courcelle and J. A. Makowsky. Fusion in Relational Structures and the Verification of Monadic Second-Order Properties. *Mathematical Structures in Computer Science*, 12(2):203–235, 2002.
- [61] B. Courcelle and I. Walukiewicz. Monadic second-order logic, graph coverings and unfoldings of transition systems. *Annals of Pure and Applied Logic*, 92:35–62, 1998.
- [62] W. Damm. The IO- and OI hierarchies. *Theoretical Computer Science*, 20(2):95–208, 1982.
- [63] C. Delhommé. Automaticité des ordinaux et des graphes homogènes. *Comptes Rendus Mathematique*, 339(1):5–10, 2004.
- [64] M.J. Dunwoody. The accessibility of finitely presented groups. *Inventiones Mathematicae*, 81(3):449–457, 1985.
- [65] S. Eilenberg, C.C. Elgot, and J.C. Shepherdson. Sets recognised by  $n$ -tape automata. *Journal of Algebra*, 13(4):447–464, 1969.
- [66] C. C. Elgot and J. E. Mezei. On relations defined by generalized finite automata. *IBM J. Research and Development*, 9:47 – 68, 1965.

- [67] C. C. Elgot and M. O. Rabin. Decidability and undecidability of extensions of second (first) order theory of (generalized) successor. *Journal of Symbolic Logic*, 31(2):169–181, 1966.
- [68] C.C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–51, 1961.
- [69] J. Engelfriet. Context-free graph grammars. In *Handbook of formal languages, vol. III*, pages 125–213. Springer-Verlag New York, Inc., New York, NY, USA, 1997.
- [70] Y.L. Ershov, S.S. Goncharov, A. Nerode, and J.B. Remmel, editors. *Handbook of Recursive Mathematics, Vol. 1*, volume 138 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, 1998.
- [71] B. Farb. Automatic Groups: A Guided Tour. *L'Enseignement Math.*, 38:291–313, 1992.
- [72] S. Fratani. *Automates à Piles de Piles ... de Piles*. Thèse de doctorat, Université Bordeaux 1, 2005.
- [73] S. Fratani. Regular sets over tree structures. Rapport Interne 1358-05, LaBRI, Université Paris 7, 2005.
- [74] S. Fratani. The theory of successor extended by severals predicates. *Journées Montoises '06*, Rennes, 2006.
- [75] S. Fratani and G. Sénizergues. Iterated pushdown automata and sequences of rational numbers. *Ann. Pure Appl. Logic*, 141(3):363–411, 2006.
- [76] Ch. Frougny. Numeration systems. In M. Lothaire, editor, *Algebraic Combinatorics on Words*. Cambridge University Press, 2002.
- [77] E. Grädel. Simple interpretations among complicated theories. *Information Processing Letters*, 35:235–238, 1990.
- [78] E. Grädel, P. G. Kolaitis, L. Libkin, M. Marx, J. Spencer, M. Vardi, Y. Venema, and S. Weinstein. *Finite Model Theory and Its Applications*. Springer-Verlag, 2007.
- [79] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games*, volume 2500 of *LNCS*. Springer-Verlag, 2002.
- [80] A. Habel. *Hyperedge Replacement: Grammars and Languages*, volume 643 of *Lecture Notes in Computer Science*. Springer, 1992.
- [81] M. Hague, A.S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *LICS'08*. IEEE Computer Society, 2008.
- [82] G. Hjorth, B. Khoussainov, A. Montalbán, and A. Nies. From automatic structures to Borel structures. In *23rd Symposium on Logic in Computer Science (LICS)*, 2008.
- [83] M. Hoffmann, D. Kuske, F. Otto, and R. M. Thomas. Some relatives of automatic and hyperbolic groups, 2002.
- [84] M. Hoffmann and R. M. Thomas. Notions of automaticity in semigroups. *Semigroup Forum*, 66:337–367., (2003).
- [85] L. Kaiser, S. Rubin, and V. Bárány. Cardinality and counting quantifiers on  $\omega$ -automatic structures. In *STACS '08*, volume 08001 of *Dagstuhl Seminar Proceedings*, pages 385–396. Internationales Begegnungs- und

- Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2008.
- [86] B. Khoussainov and M. Minnes. Model theoretic complexity of automatic structures. *Annals of Pure and Applied Logic*, To appear, 2008.
  - [87] B. Khoussainov and A. Nerode. Automatic presentations of structures. In *LCC '94*, volume 960 of *LNCS*, pages 367–392. Springer-Verlag, 1995.
  - [88] B. Khoussainov, A. Nies, S. Rubin, and F. Stephan. Automatic structures: Richness and limitations. In *LICS'04*, pages 44–53, 2004.
  - [89] B. Khoussainov and S. Rubin. Graphs with automatic presentations over a unary alphabet. *Journal of Automata, Languages and Combinatorics*, 6(4):467–480, 2001.
  - [90] B. Khoussainov, S. Rubin, and F. Stephan. Definability and regularity in automatic structures. In *STACS '04*, volume 2996 of *LNCS*, pages 440–451, 2004.
  - [91] B. Khoussainov, S. Rubin, and F. Stephan. Automatic linear orders and trees. *ACM Transactions on Computational Logic*, 6(4):675–700, 2005.
  - [92] T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS'02*, volume 2303 of *LNCS*, pages 205–222, 2002.
  - [93] D. Kuske. Is cantor's theorem automatic? In *LPAR*, volume 2850 of *LNCS*, pages 332–345. Springer, 2003.
  - [94] D. Kuske and M. Lohrey. First-order and counting theories of  $\omega$ -automatic structures. In *FoSSaCS*, pages 322–336, 2006.
  - [95] D. Kuske and M. Lohrey. Automatic structures of bounded degree revisited. arXiv:0810.4998, 2008.
  - [96] D. Kuske and M. Lohrey. Hamiltonicity of automatic graphs. In *FIP TCS 2008*, 2008.
  - [97] H. Lauchli and Ch. Savioz. Monadic Second Order Definable Relations on the Binary Tree. *J. of Symbolic Logic*, 52(1):219–226, 1987.
  - [98] S. Lifsches and S. Shelah. Uniformization and skolem functions in the class of trees. *Journal of Symbolic Logic*, 63:103–127, 1998.
  - [99] Ch. Löding. *Infinite Graphs Generated by Tree Rewriting*. Doctoral thesis, RWTH Aachen, 2003.
  - [100] Christof Löding. Reachability problems on regular ground tree rewriting graphs. *Theor. Comp. Sys.*, 39(2):347–383, 2006.
  - [101] M. Lohrey. Automatic structures of bounded degree. In *LPAR*, volume 2850 of *LNCS*, pages 346–360. Springer, 2003.
  - [102] M. Lohrey. Decidability and complexity in automatic monoids. In *Developments in Language Theory*, pages 308–320, 2004.
  - [103] A. Meyer. Traces of term-automatic graphs. *R.A.I.R.O. Theoretical Informatics and Applications*, 42, 2008.
  - [104] C. Michaux and F. Point. Les ensembles  $k$ -reconnaissables sont définissables dans  $\langle \mathbb{N}, +, V_k \rangle$ . *C. R. Acad. Sci. Paris Sér. I Math.*, 303(19):939–942, 1986.
  - [105] Ch. Morvan. *Les graphes rationnels*. Thèse de doctorat, Université de Rennes 1, Novembre 2001.
  - [106] Ch. Morvan. Classes of rational graphs. Journées Montoises '06, Rennes, 2006.

- [107] Ch. Morvan and Ch. Rispal. Families of automata characterizing context-sensitive languages. *Acta Informatica*, 41(4-5):293–314, 2005.
- [108] Ch. Morvan and C. Stirling. Rational graphs trace context-sensitive languages. In A. Pultr and J. Sgall, editors, *MFCS 01*, volume 2136 of *LNCS*, pages 548–559, 2001.
- [109] A. A. Muchnik. The definable criterion for definability in Presburger arithmetic and its applications. *Theor. Comput. Sci.*, 290(3):1433–1444, 2003.
- [110] D. E. Muller and P. E. Schupp. Context-free languages, groups, the theory of ends, second-order logic, tiling problems, cellular automata, and vector addition systems. *Bull. Amer. Math. Soc.*, 4(3):331–334, 1981.
- [111] D. E. Muller and P. E. Schupp. Groups, the theory of ends, and context-free languages. *J. Comput. Syst. Sci.*, 26(3):295–310, 1983.
- [112] D. E. Muller and P. E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theor. Comput. Sci.*, 37:51–75, 1985.
- [113] A. A. Nabebin. Expressibility in a restricted second-order arithmetic. *Siberian Mathematical Journal*, 18(4):588–593, 1977.
- [114] A. Nies. Describing groups. *Bulletin of Symbolic Logic*, 13(3):305–339, 2007.
- [115] D. Niwiński. On the cardinality of sets of infinite trees recognizable by finite automata. In *Proceedings of the 16th International Symposium on Mathematical Foundations of Computer Science, MFCS'91*, volume 520, pages 367–376. Springer, 1991.
- [116] G. P. Oliver and R. M. Thomas. Finitely generated groups with automatic presentations. In *STACS 2005*, volume 3404 of *LNCS*, pages 693–704. Springer, 2005.
- [117] C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS*, pages 81–90. IEEE Computer Society, 2006.
- [118] J.-J. Pansiot. On various classes of infinite words obtained by iterated mappings. In *Automata on Infinite Words*, pages 188–197, 1984.
- [119] A. Rabinovich. On decidability of monadic logic of order over the naturals extended by monadic predicates. Unpublished note, 2005.
- [120] A. Rabinovich and W. Thomas. Decidable theories of the ordering of natural numbers with unary predicates. Submitted, 2006.
- [121] M. Rigo. Numeration systems on a regular language: Arithmetic operations, recognizability and formal power series. *Theoretical Computer Science*, 269:469, 2001.
- [122] M. Rigo and A. Maes. More on generalized automatic sequences. *J. of Automata, Languages and Combinatorics*, 7(3):351–376, 2002.
- [123] Ch. Rispal. The synchronized graphs trace the context-sensistive languages. *Electronic Notes in Theor. Comp. Sci.*, 68(6), 2002.
- [124] S. Rubin. *Automatic Structures*. Phd thesis, University of Auckland, NZ, 2004.
- [125] S. Rubin. Automata presenting structures: A survey of the finite-string case. *Bulletin of Symbolic Logic*, 14(2):169–209, 2008.

- [126] A. L. Semenov. Decidability of monadic theories. In *Mathematical Foundations of Computer Science, Prague, 1984*, volume 176 of *LNCS*, page 162?175. Springer, Berlin, 1984.
- [127] G. Sénizergues. Semi-groups acting on context-free graphs. In *ICALP '96: Proceedings of the 23rd International Colloquium on Automata, Languages and Programming*, pages 206–218, London, UK, 1996. Springer-Verlag.
- [128] G. Sénizergues. The bisimulation problem for equational graphs of finite out-degree. *SIAM J. Comput.*, 34(5):1025–1106, 2005.
- [129] P. V. Silva and B. Steinberg. A geometric characterization of automatic monoids. *The Quarterly Journal of Mathematics*, 55:333–356, 2004.
- [130] J. Su and S. Grumbach. Finitely representable databases (extended abstract). In *In Proc. 13th ACM Symp. on Principles of Database Systems*, 1994.
- [131] A. Szilard, Sh. Yu, K. Zhang, and J. Shallit. Characterizing regular languages with polynomial densities. In *MFCS*, pages 494–503, 1992.
- [132] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 133–192. Elsevier and MIT Press, 1990.
- [133] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, volume III*, pages 389–455. Springer, New York, 1997.
- [134] W. Thomas. Constructing Infinite Graphs with a Decidable MSO-Theory. In *MFCS*, volume 2747 of *LNCS*, pages 113–124, 2003.
- [135] B.A. Trahtenbrot. Finite automata and the logic of one-place predicates. Russian. *Siberian Mathematical Journal*, 3:103–131, 1962. English translation: American Mathematical Society Translations, Series 2, 59 (1966), 23–55.
- [136] T. Tsankov. The additive group of the rationals is not automatic. manuscript, 2009.
- [137] R. Villemaire. The theory of  $\langle \mathbb{N}, +, V_k, V_l \rangle$  is undecidable. *Theoretical Computer Science*, 106:337–349, 1992.
- [138] I. Walukiewicz. Monadic second-order logic on tree-like structures. *Theoretical Computer Science*, 275:311–346, 2002.
- [139] S. Wöhrle and W. Thomas. Model checking synchronized products of infinite transition systems. In *LICS '04*, pages 2–11, Washington, DC, USA, 2004. IEEE Computer Society.

### **4.3 Publication**

The rest of this page is intentionally left blank

# Verification of Asynchronous Mobile-Robots in Partially-Known Environments<sup>\*</sup>

Benjamin Aminof<sup>1</sup> and Aniello Murano<sup>2</sup> and Sasha Rubin<sup>2</sup> and Florian Zuleger<sup>1</sup>

<sup>1</sup> Technische Universität Wien

<sup>2</sup> Università degli Studi di Napoli "Federico II"

**Abstract.** This paper establishes a framework based on logic and automata theory in which to model and automatically verify that multiple mobile robots, with sensing abilities, moving asynchronously, correctly perform their tasks. The motivation is from practical scenarios in which the environment is not completely known to the robots, e.g., physical robots exploring a maze, or software agents exploring a hostile network. The framework shows how to express tasks in a logical language, and exhibits an algorithm solving the *parameterised* verification problem, where the graphs are treated as the parameter. The main assumption that yields decidability is that the robots take a bounded number of turns. We prove that dropping this assumption results in undecidability, even for robots with very limited ("local") sensing abilities.

## 1 Introduction

Autonomous mobile robots are designed to achieve some task in an environment without a central control. Foundational tasks include, for example, rendezvous (gather all robots in a single position) and reconfiguration (move to a new configuration in a collision-free way) [25, 14, 15]. This paper studies robots in *partially known* environments, i.e., robots do not have global information about the environment, but may know some (often topological) information (e.g., whether the environment is connected, or that it is a ring of some unknown size) [14]. The motivation for studying partially known environments is that in many practical scenarios the robots are unaware of the exact environment in which they are operating, e.g., mobile software exploring a hostile computer network, or physical robots that rendezvous in an environment not reachable by humans.

To illustrate, here is an *example reconfiguration problem*. Suppose that  $k$  robots find themselves on different internal nodes of a binary tree, and each robot has to reach a different leaf in a collision free way. Each robot can sense

---

\* Benjamin Aminof and Florian Zuleger were supported by the Austrian National Research Network S11403-N23 (RiSE) of the Austrian Science Fund (FWF) and by the Vienna Science and Technology Fund (WWTF) through grant ICT12-059. Aniello Murano was supported by FP7 EU project 600958-SHERPA. Sasha Rubin is a Marie Curie fellow of the Istituto Nazionale di Alta Matematica.

if its left (or right) child is occupied by a robot. One protocol for solving this problem (assuming a large enough tree) is for each robot to execute ‘go to the left child, then repeatedly go the right child’ until a leaf is reached. Each move is guarded by a test that the child it wants to move to is not currently occupied.

As the example illustrates, we make the following modeling choices: environments are discrete (rather than continuous), robots have finite memory (rather than being oblivious, or being Turing powerful), robots are nondeterministic (rather than probabilistic), robots move asynchronously (rather than synchronously), and robots can sense the positions and the internal states of each robot no matter where they are, i.e., they can perform “remote” tests (but cannot leave information at visited positions).<sup>3</sup> The assumption that robots move asynchronously is motivated as follows: processes in distributed systems have no common notion of time since they may be running with different internal clocks, and thus a standard abstraction is to assume that processes’ actions are interleaved, see [26]. There are two main ways to interleave the robots: an adversary tries to make sure that the robots do not succeed [16], and a co-operator tries to help the robots succeed (reminiscent of schedulers in strategic reasoning [7]).

In this paper we provide a framework for modeling and verifying that multiple mobile robots achieve a task (under adversarial and co-operative interleavings) in a partially-known environment. We now explain how we model the fact that environments are partially-known. Fix a class  $\mathcal{G}$  of environments (e.g.,  $\mathcal{G}$  is the set of all lines, or all grids, or all rings). The *parameterised verification problem* states: Given robots  $\bar{R}$ , decide if they solve the task  $T$  on all graphs  $G \in \mathcal{G}$ . Requiring the robots to solve the task  $T$  on all graphs from a class  $\mathcal{G}$  is how we model that the robots operate in partially-known environments — the robots know they are in a graph from  $\mathcal{G}$ , but they do not know which one. In contrast, the classic (non-parameterised) verification problem states: Given robots  $\bar{R}$  and a graph  $G \in \mathcal{G}$ , decide if robots  $\bar{R}$  solve the task  $T$  on  $G$ . In that setting, the robots can be designed to exploit the exact structure (size, etc.) of the graph.

**Aims and Contributions.** The aim of this work is to provide a formal framework in which one can reason (both mathematically and computationally) about multiple mobile-robots, with sensing abilities, moving asynchronously in a discrete, static, partially-known environment. We prove that parameterised verification is undecidable already for line-environments, and even for robots with limited tasks (e.g., halting) which can only detect collisions. I.e., a robot can only sense which other robots share its current position. This undecidability result also holds for robots that move synchronously. On the other hand, we prove that parameterised verification is decidable for scenarios in which the number of times that the robots take turns is bounded.<sup>4</sup> This decidability result is very robust:

---

<sup>3</sup> The ability to sense positions is a form of vision, while the ability to sense internal states is a form of communication

<sup>4</sup> In the example reconfiguration problem, there is some ordering of the robots that switches turns at most  $k$  times in which the stated robot-protocol succeeds. Also, for every ordering of the robots that switches turns a sufficiently large number of times, the protocol succeeds (“ordering” and “switching” are formalised in Section 3).

it holds on a very general class of graphs called *context-free sets of graphs* which include e.g., rings, trees, series-parallel graphs, but not grids; it also holds with very powerful abilities called *position-tests* which allow each robot to remotely test the positions of all the robots using formulas which include, e.g., the ability to test connectivity, as well as *state-tests* that allow each robot to remotely test the internal state of the other robots; it holds for tasks that are expressible using a new logic MRTL (Multiple-Robot Task Logic), which can express many natural tasks, e.g., reconfiguration, gathering, and “safe” variations (i.e., complete the task without touching dangerous positions).

**Related Work.** The work closest to ours is [30] which also considered the parameterised verification problem for multi-robot systems. However, in that paper, the decidability result (and the corresponding logic RTL) was only for one-robot systems (i.e.,  $k = 1$ ), and the undecidability result for  $k = 2$  was for multiple robots that move *synchronously* and have *remote* tests.

The distributed-computing community has proposed and studied a number of models of robot systems, e.g., recently [24, 18, 10, 13, 8, 17]. This literature is mostly mathematical, and *theorems from this literature are parameterised*, i.e., they may involve graph-parameters (e.g., the maximum degree, the number of vertices, the diameter), memory parameters (e.g., the number of internal states of the robot protocol), and the number of robots may be a parameter. Only recently has there been emphasis on formal analysis of correctness of robots in a parameterised setting [22, 4, 27, 23, 30, 29]. In these formal analyses, typically it is the *number of agents* that is treated as the parameter [22, 4, 27, 23]. In contrast, in this paper (as in [30]) we apply formal methods to the parameterised verification problem in which it is the *environment that is parameterised*.

Also, the formal-verification community has placed emphasis on distributed models in which processes are stationary and interact by sending messages (e.g., in a broadcast or rendezvous manner) or by using guarded commands. The parameterised verification problem for such distributed processes is, in general, undecidable [31]. By simplifying the systems (e.g., restricting the mode of communication, the abilities of the processes, the specification languages, etc.) one can get decidable parameterised verification, recently e.g., [12, 1–3].

We refer the reader to [30, Section 7] for an up-to-date and detailed discussion of the connections between multi-robot systems, classic automata theory (i.e., graph-walking automata) and distributed systems (in particular, token-passing systems). Finally, we mention that there is a smattering of work on parameterised *synthesis* (called *generalised planning* in the AI literature) [11, 19–21].

## 2 Background: Automata Theory

Write  $B^*$  and  $B^\omega$  for the sets of finite and infinite sequences over alphabet  $B$ , respectively. The empty sequence is denoted  $\epsilon$ . Write  $[n]$  for the set  $\{1, 2, \dots, n\}$ .

**Graphs and Trees.** A  $\Sigma$ -*graph*, or *graph*,  $G$ , is a tuple  $(V, E, \Sigma, \lambda)$  where  $V$  is a finite set of *vertices*,  $E \subseteq V \times V$  is the *edge relation*,  $\Sigma$  is a finite set of *edge*

*labels*, and  $\lambda : E \rightarrow \Sigma$  is the *edge labeling function*. A  $\Delta$ -ary tree (for  $\Delta \in \mathbb{N}$ ) is a  $\Sigma$ -graph  $(V, E, \Sigma, \lambda)$  where  $(V, E)$  is a tree,  $\Sigma = [\Delta] \cup \{\text{up}\}$ , and  $\lambda$  labels the edge leading to the node in direction  $i$  (if it exists) by  $i$ , and the edge leading to the parent of a node (other than the root) is labelled by *up*. We may rename the labels for convenience, e.g., for binary trees ( $\Delta = 2$ ) we let  $\Sigma = \{\text{lc}, \text{rc}, \text{up}\}$  where *lc* replaces 1 and *rc* replaces 2.

**Monadic Second-order Logic.** Formulas are interpreted in  $\Sigma$ -graphs  $G$ . Define the set of monadic second-order formulas  $\text{MSOL}(\Sigma)$  as follows. Formulas of  $\text{MSOL}(\Sigma)$  are built using *first-order variables*  $x, y, \dots$  that vary over vertices, and *set variables*  $X, Y, \dots$  that vary over sets of vertices. The *atomic formulas* (when interpreted over  $\Sigma$ -graphs) are:  $x = y$  (denoting that vertex  $x$  is the same as vertex  $y$ ),  $x \in X$  (denoting that vertex  $x$  is in the set of vertices  $X$ ),  $\text{edg}_\sigma(x, y)$  (denoting that there is an edge from  $x$  to  $y$  labeled  $\sigma \in \Sigma$ ) and *true* (the formula that is always true). The formulas of  $\text{MSOL}(\Sigma)$  are built from the atomic formulas using the Boolean connectives (i.e.,  $\neg, \vee, \wedge, \rightarrow$ ) and variable quantification (i.e.,  $\forall, \exists$  over both types of variables). A variable that is not quantified is called *free*. The fragment of  $\text{MSOL}(\Sigma)$  which does not mention set variables is called *first-order logic*, denoted  $\text{FOL}(\Sigma)$ . Write  $\text{MSOL}_k(\Sigma)$  for formulas with at most  $k$  many free first-order variables and no free set-variables. We abbreviate  $z_1, \dots, z_k$  by  $\bar{z}$ . We write  $\phi(x_1, \dots, x_k)$  to mean that the free variables from the formula  $\phi$  are amongst the set  $\{x_1, \dots, x_k\}$  – note that the formula  $\phi(x_1, \dots, x_k)$  does not need to use all of the variables  $x_1, \dots, x_k$ . For a graph  $G$ , and  $v_1, \dots, v_k \in V$ , we write  $G \models \phi(v_1, \dots, v_k)$  to mean that  $\phi$  holds in  $G$  with variable  $x_i$  simultaneously substituted by vertex  $v_i$  (for all  $i \in [k]$  for which  $x_i$  occurs free in  $\phi$ ). Here are some examples of formulas and their meanings: The formula  $\forall x(x \in X \rightarrow x \in Y)$  means that  $X \subseteq Y$ . Similarly, there are formulas for the set operations  $\cup, \cap, =$ , and relative complement  $X \setminus Y$ . The formula  $\text{edg}(x, y) := \bigvee_{\sigma \in \Sigma} \text{edg}_\sigma(x, y)$  means that there is an edge from  $x$  to  $y$  (here  $\Sigma$  is assumed to be finite). The formula  $E^*(x, y) := \forall Z[(\text{closed}_E(Z) \wedge x \in Z) \rightarrow y \in Z]$  where  $\text{closed}_E(Z)$  is  $\forall a \forall b[(a \in Z \wedge E(a, b)) \rightarrow b \in Z]$  defines the transitive closure of  $E$ . Generally,  $\text{MSOL}$  can express the 1-ary transitive closure operator (e.g., [30]).

**The Validity Problem and Courcelle's Theorem.** A *sentence* is a formula with no free variables. Let  $\Phi$  be a set of sentences, and let  $\mathcal{G}$  be a set of graphs. The  $\Phi$ -*validity problem* of  $\mathcal{G}$  is to decide, given  $\phi \in \Phi$ , whether for all graphs  $G \in \mathcal{G}$ , it holds that  $G \models \phi$ . Unfortunately, the  $\text{MSOL}(\Sigma)$ -validity problem for the set  $\mathcal{G}$  of all  $\Sigma$ -graphs is undecidable. However, Courcelle's Theorem states that  $\text{MSOL}$ -validity of context-free sets of graphs is uniformly decidable, i.e., there is an algorithm that given a description of a context-free set of graphs  $\mathcal{G}$  and an  $\text{MSOL}$ -sentence  $\phi$  decides if every graph in  $\mathcal{G}$  satisfies  $\phi$  [9]. Context-free sets of graphs are the analogue of context-free sets of strings, and can be described by graph grammars, equations using certain graph operations, or  $\text{MSOL}$ -transductions of the set of trees. Formally,  $\mathcal{G}$  is *context-free* if it is  $\text{MSOL}$ -definable and of bounded clique-width [9]. Examples include, for a fixed alphabet, the set of labeled lines, rings, trees, series-parallel graphs, cliques, but not the set of grids.

**Automata and Regular Expressions.** Ordinary *regular-expressions* over a finite alphabet  $B$  are built from the sets  $\emptyset$ ,  $\{\epsilon\}$ , and  $\{b\}$  ( $b \in B$ ), and the operations union  $+$ , concatenation  $\cdot$ , and Kleene-star  $*$ . Kleene’s Theorem states that the languages definable by regular expressions over alphabet  $B$  are exactly those recognised by finite automata over alphabet  $B$ . An  $\omega$ -*regular expression* over alphabet  $B$  is inductively defined to be of the form:  $\text{exp}^\omega$ ,  $\text{exp} \cdot r$ , or  $r + r'$ , where  $\text{exp}$  is an ordinary regular-expression over  $B$ , and  $r, r'$  are  $\omega$ -regular expressions over  $B$ . An  $\omega$ -regular language is one defined by an  $\omega$ -regular expression. A variation of Kleene’s Theorem says that the languages definable by  $\omega$ -regular expressions over alphabet  $B$  are exactly the languages recognised by Büchi automata over alphabet  $B$  (which are like finite automata except they take infinite words as input, and accept if some accepting state occurs infinitely often).

### 3 The Model of Robot Systems

In this section we provide a framework for modeling multi-robot systems parameterised by their environment. Environments are modeled as  $\Sigma$ -graphs  $G$  and robots are modeled as regular languages of instructions. An instruction either tells the robot to move along an edge, or to test robot positions (e.g., a robot can learn which other robots are at the same vertex as it is, or if there is a robot north of it). Tests are formalised as logical formulas.

**Instructions for Robots.** Fix a number  $k$  of robots and a set of edge labels  $\Sigma$ . A *command* is a symbol from  $\{\uparrow_\sigma: \sigma \in \Sigma\} \cup \{\circlearrowleft\}$ . The command  $\uparrow_\sigma$  tells the robot to move from its current vertex along the edge labeled  $\sigma$ , and the command  $\circlearrowleft$  tells the robot to stay at its current vertex. A *position-test* is a formula from  $\text{MSOL}_k(\Sigma)$ . A *state-test* is an expression of the form “robot  $i$  is in state  $q$ ”, where  $i \in [k]$  and  $q$  is a symbol denoting a state of a robot (formally we may assume that all robots have states from  $\mathbb{N}$ , and that  $q \in \mathbb{N}$ )<sup>5</sup>. A position test  $\tau(x_1, \dots, x_k)$  allows the robot to test that  $\tau(x_1, \dots, x_k)$  holds in  $G$ , where  $x_i$  is the current vertex of robot  $R_i$  in  $G$ . Simple tests include “ $x_i = x_j$ ” which tests if robots  $i$  and  $j$  are in the same vertex (i.e., collision detection), and “ $\text{edg}(x_i, x_j) \vee \text{edg}(x_j, x_i)$ ” which tests if robots  $i$  and  $j$  are adjacent. A *test* is a state-test or a position-test. The *instruction set*  $\text{INS}_{\Sigma, k}$  consists of all expressions of the form  $\tau \rightarrow \kappa$  where  $\tau$  is a test and  $\kappa$  is a command.

**Robots, Configurations, Runs.** A  $k$ -*robot ensemble* is a vector of robots  $\langle R_1, \dots, R_k \rangle$  where each *robot*  $R_i = \langle Q_i, \delta_i \rangle$ , each  $Q_i$  is a finite set of *states*, and each  $\delta_i \subset Q_i \times \text{INS}_{\Sigma, k} \times Q_i$  is a finite *transition* relation. For technical convenience, we assume that robot  $i$  does not test its own state, i.e., no  $\text{ins}$  in a transition  $(p, \text{ins}, q) \in \delta_i$  contains any occurrences of state-tests of the form “robot  $i$  is in state  $j$ ”. We designate a subset  $I_i \subseteq Q_i$  of the states of robot  $i$

---

<sup>5</sup> Note that, for ease of exposition, we do not explicitly allow Boolean combinations of state-tests. However, these can be indirectly performed by chaining state-tests (remembering the previous test results in the local state) to perform conjunctions, and using nondeterminism for disjunctions.

as *initial* states, and a subset  $A_i \subseteq Q_i$  of its states as *accepting* states. A state  $p \in Q_i$  is called *halting* if the only transition the robot has from  $p$  is  $(p, \text{true}, p)$ . Thus we model a halting robot as one that forever stays in the same state and does not move. The halting states are denoted  $H_i \subseteq Q_i$ .

Fix a  $\Sigma$ -graph  $G$ . A *configuration*  $c$  of  $\langle R_1, \dots, R_k \rangle$  on graph  $G$  is a pair  $\langle \bar{v}, \bar{q} \rangle \in V^k \times \prod_{i \in [k]} Q_i$ . A configuration is *initial* if  $\bar{q} \in \prod I_i$ . For a test  $\tau$  and a configuration  $c = \langle \bar{u}, \bar{p} \rangle$ , define  $c \Vdash \tau$  to mean that configuration  $c$  makes  $\tau$  true in  $G$ . Formally, if  $\tau$  is a position test then define  $c \Vdash \tau$  iff  $G \models \tau(\bar{u})$ , and if  $\tau$  is a state-test, say “robot  $i$  is in state  $j$ ”, then define  $c \Vdash \tau$  iff  $p_i = j$ . The following definition of  $\vdash_i$  expresses that one configuration results from another after robot  $i$  successfully executes an instruction, while the rest are idle: for  $i \in [k]$  and configurations  $c = \langle \bar{w}, \bar{q} \rangle, d = \langle \bar{v}, \bar{p} \rangle$ , write  $c \vdash_i d$  if  $p_j = q_j$  and  $w_j = v_j$  for all  $j \neq i$ , and there exists a transition  $(p_i, \tau \rightarrow \kappa, q_i) \in \delta_i$  (i.e., of robot  $R_i$ ) such that  $c \Vdash \tau$  (i.e., the current configuration satisfies the test  $\tau$ ) and, if  $\kappa = \uparrow_\sigma$  then  $\lambda(w_i, v_i) = \sigma$ , and if  $\kappa = \circlearrowleft$  then  $w_i = v_i$ .

**Schedules and Runs.** A *schedule* is a finite or infinite sequence  $\mathcal{S} = s_1 s_2 s_3 \dots$  where  $s_i \in [k]$ . A *run*  $\rho$  of  $\langle R_1, \dots, R_k \rangle$  on  $G$  starting with an initial configuration  $c$  according to schedule  $\mathcal{S}$  is a finite or infinite sequence  $c_1 c_2 c_3 \dots$  of configurations such that  $c_1 = c$  and for all  $i$ ,  $c_i \vdash_{s_i} c_{i+1}$ . The *set (resp. sequence) of positions of a run*  $\alpha = \langle \bar{v}_1, \bar{p}_1 \rangle \langle \bar{v}_2, \bar{p}_2 \rangle \dots$  is the set of positions  $\{\bar{v}_1, \bar{v}_2, \dots\}$  (resp. sequence  $\bar{v}_1 \bar{v}_2 \dots$  of positions) of its configurations. In a similar way define the *set (resp. sequence) of positions of robot  $i$  on a run*.

**Orderings.** A *(finite)  $k$ -ordering* is a string  $\alpha \in [k]^+$ , say of length  $N + 1$ , such that  $\alpha_i \neq \alpha_{i+1}$  for  $1 \leq i \leq N$ . Write  $\|\alpha\| = N$  to mean that  $|\alpha| = N + 1$ , and say that  $\alpha$  is  $N$ -switching. E.g., 171 is 2-switching. Say that a schedule  $\mathcal{S}$  follows  $\alpha$  if  $\mathcal{S}$  is in  $\alpha_1^* \alpha_2^* \dots \alpha_N^* \alpha_{N+1}^*$  or  $\alpha_1^* \alpha_2^* \dots \alpha_N^* \alpha_{N+1}^\omega$ , i.e., robot  $\alpha_1$  is scheduled for some (possibly no) time, then robot  $\alpha_2$  is scheduled, and so on, until  $\alpha_{N+1}$  which can be scheduled forever. Similarly, an *infinite  $k$ -ordering* of  $k$  robots is a string  $\alpha \in [k]^\omega$  such that  $\alpha_i \neq \alpha_{i+1}$  for all  $i \in \mathbb{N}$ . In this case write  $\|\alpha\| = \infty$ . A schedule follows  $\alpha$  if the schedule is in the set  $\alpha_1^* \alpha_2^* \dots$ .

**Robot Tasks.** Robots should achieve some task in their environment. We give some examples of foundational robot tasks [25]: A robot ensemble *deploys* or *reconfigures* if they move, in a collision-free way, to a certain target configuration. A robot ensemble *gathers* if, no matter where each robot starts, there is a vertex  $z$ , such that eventually every robot is in  $z$ . A robot ensemble *collaboratively explores* a graph if, no matter where they start, every node is eventually visited by at least one robot. All of these tasks have *safe* variations: the robots complete their task without entering certain pre-designated “bad” nodes.

**Multi-Robot Task Logic — MRTL.** We now define MRTL, a logic for formally expressing robot tasks. We first define the syntax and semantics, and then we give some example formulas. Later (in Lemma 1) we prove that, when restricted to bounded-switching orderings, MRTL formulas (and therefore many interesting natural tasks) can be converted into MSOL formulas over graphs.

**MRTL Syntax.** Fix  $k \in \mathbb{N}$  and  $\Sigma$ . Formulas of  $\text{MRTL}_k$  are built, as in the definition of  $\text{MSOL}(\Sigma)$  from Section 2, from the following atomic formulas:  $x = y$ ,  $\text{edg}_\sigma$  (for  $\sigma \in \Sigma$ ),  $x \in X$ ,  $\text{true}$ , and the following additional atomic formulas (with free variables  $\bar{X}, \bar{x}, \bar{y}$  each of size  $k$ )  $\text{Reach}_Q, \text{Halt}_Q^K, \text{Infty}_Q$  and  $\text{Rept}_Q^K$  where  $Q \in \{\exists, \forall\}$  and  $\emptyset \neq K \subseteq [k]$ . Denote by  $\text{MRTL}$  the set of formulas  $\cup_k \text{MRTL}_k$ .

**MRTL Semantics.** Formulas of  $\text{MRTL}_k$  are interpreted over graphs  $G$ , and with respect to  $k$ -robot ensembles  $\bar{R}$  and a set of orderings  $\Omega$ . Define the satisfaction relation  $\models_{\bar{R}, \Omega}$ :

- $G \models_{\bar{R}, \Omega} \text{Reach}_\exists(\bar{X}, \bar{x}, \bar{y})$  iff there is an ordering  $\alpha \in \Omega$  and there is a finite run of  $\bar{R}$  on  $G$  that uses a schedule that follows  $\alpha$ , such that the run starts with some initial configuration of the form  $\langle \bar{x}, \bar{p} \rangle$  (i.e.,  $\bar{p} \in \prod I_i$ ), ends with a configuration of the form  $\langle \bar{y}, \bar{q} \rangle$  (i.e.,  $\bar{q} \in \prod Q_i$ ), and for each  $i \in [k]$ , the set of positions of robot  $i$  on this run is contained in  $X_i$ .
- $G \models_{\bar{R}, \Omega} \text{Halt}_\exists^K(\bar{X}, \bar{x}, \bar{y})$  means the same as  $\text{Reach}_\exists$  except that the last tuple of states  $\bar{q}$  has the property that  $i \in K$  implies that  $q_i \in H_i$  (i.e., every robot in  $K$  is in a halting state).
- $G \models_{\bar{R}, \Omega} \text{Infty}_\exists(\bar{X}, \bar{x}, \bar{y})$  means the same as  $\text{Reach}_\exists$  except that the run is infinite and, instead of ending in  $\bar{y}$ , it visits  $\bar{y}$  infinitely often.
- $G \models_{\bar{R}, \Omega} \text{Rept}_\exists^K(\bar{X}, \bar{x}, \bar{y})$  means the same as  $\text{Reach}_\exists$  except that the run is infinite, and infinitely often it reaches a configuration of the form  $\langle \bar{y}, \bar{q} \rangle$  such that  $i \in K$  implies that  $q_i$  is an accepting state (i.e.,  $q_i \in A_i$ ).
- $G \models_{\bar{R}, \Omega} \text{Reach}_\forall(\bar{X}, \bar{x}, \bar{y})$  is the same as  $\text{Reach}_\exists$  except replace “there is an ordering  $\alpha \in \Omega$  and there is a finite run...” by “for every ordering  $\alpha \in \Omega$  there is a finite run ...”. In a similar way, define  $\text{Halt}_\forall^K, \text{Infty}_\forall$  and  $\text{Rept}_\forall^K$ .

Extend the satisfaction relation to all formulas of  $\text{MRTL}_k$  in the natural way.

*Example 1.* The statement  $G \models_{\bar{R}, \Omega} (\forall \bar{x})(\exists \bar{y})(\exists \bar{X}) \text{Reach}_\exists(\bar{X}, \bar{x}, \bar{y}) \wedge (\wedge_{i,j} y_i = y_j)$  means that, no matter where the robots start in  $G$ , there is an ordering  $\alpha \in \Omega$ , and a run according to a schedule that follows  $\alpha$ , such that the robots  $\bar{R}$  gather at some vertex of the graph  $G$ . Replacing  $\text{Reach}_\exists$  by  $\text{Reach}_\forall$  means, no matter where the robots start in  $G$ , for every ordering  $\alpha \in \Omega$ , the robots have a run according to a schedule that follows  $\alpha$  such that the robots gather at a vertex of the graph. Note that by conjuncting with  $\wedge_i X_i \cap B = \emptyset$  where  $B$  is an MSOL-definable set of “bad” vertices, one can express “safe gathering”.

*Example 2.* Consider the statement  $G \models_{\bar{R}, \Omega} (\forall \bar{x})(\exists \bar{y})[\text{NONLEAF}(\bar{x}) \wedge \text{DIFF}(\bar{x}) \rightarrow (\text{LEAF}(\bar{y}) \wedge \text{DIFF}(\bar{y}) \wedge \text{Reach}_\forall(V^k, \bar{x}, \bar{y}))]$  where  $G$  is a tree,  $\text{NONLEAF}(\bar{x})$  is an MSOL-formula expressing that every  $x_i$  is not a leaf,  $\text{LEAF}(\bar{y})$  is an MSOL-formula expressing that every  $y_i$  is a leaf, and  $\text{DIFF}(\bar{z})$  is an MSOL-formula expressing that  $z_i \neq z_j$  for  $i \neq j$ . The statement says that, as long as the robots start on different internal nodes of the tree  $G$ , for every ordering  $\alpha \in \Omega$  there is a run of the robots  $\bar{R}$  according to a schedule that follows  $\alpha$  in which the robots reconfigure and arrive at different leaves.

## 4 Reasoning about Robot Systems

We formalise the parameterised verification problem for robot protocols and then study its decidability. The parameterised verification problem depends on a (typically infinite) set of graphs  $\mathcal{G}$ , a set of  $k$ -robot ensembles  $\mathcal{R}$ , a  $k$ -robot task written as an  $\text{MRTL}_k$  formula  $T$ , and a set of  $k$ -orderings  $\Omega$ .

**Definition 1.** *The parameterised verification problem  $\text{PVP}_{T,\Omega}(\mathcal{G}, \mathcal{R})$  is: given a robot ensemble  $\bar{R}$  from  $\mathcal{R}$ , decide whether for every graph  $G \in \mathcal{G}$ ,  $G \models_{\bar{R}, \Omega} T$  (i.e., the robots  $\bar{R}$  achieves the task  $T$  on  $G$  with orderings restricted to  $\Omega$ ).*

*Example 3.* Let  $\mathcal{G}$  be the set of all binary trees,  $\mathcal{R}$  be the set of all  $k$ -robot ensembles, let  $\Omega_b := \{\alpha \in [k]^*: ||\alpha|| = b\}$  be the set of  $b$ -switch orderings, and let  $T$  be the task expressing that if the robots start on different internal nodes of a tree then they eventually reconfigure themselves to be on different leaves of the tree, no matter which ordering from  $\Omega_b$  is chosen (cf. Example 2). We will see later that one can decide  $\text{PVP}_{T,\Omega_b}(\mathcal{G}, \mathcal{R})$  given  $b \in \mathbb{N}$ . So, one can decide, given  $b$ , whether the protocol from the reconfiguration example (in the Introduction) succeeds for every ordering with  $b$  switches.

In Section 4.1 we show that the PVP is undecidable even on lines, for simple tasks, and allowing the robots very restricted testing abilities, i.e., a robot can sense which of the other robots shares the same position with it, called “local collision tests”. In Section 4.2 we show that we can guarantee decidability merely by restricting the scheduling regime while allowing the robots full testing abilities, including testing positions and states of other robots “remotely”.

### 4.1 Undecidability of Multi-Robot Systems on a Line

Our undecidability proof proceeds by reducing the halting problem of two counter machines to the parameterised verification problem. An *input-free 2-counter machine* (2CM) [28] is a deterministic program manipulating two nonnegative integer counters using commands that can increment a counter by 1, decrement a counter by 1, and check whether a counter is equal to zero. We refer to the “line numbers” of the program code as the “states” of the machine. One of these states is called the *halting state*, and once it is entered the machine *halts*. Observe that a 2CM has a single computation, and that if it halts then the values of both counters are bounded by some integer  $n$ . The *non-halting problem for 2CMs* is to decide, given a 2CM  $\mathfrak{M}$ , whether it does not halt. This problem is known to be undecidable [28], and is usually a convenient choice for proving undecidability of problems concerning parameterised systems due to the simplicity of the operations of counter machines.

Let  $\mathcal{G}$  be the set of all graphs that are finite lines. Formally, for every  $n \in \mathbb{N}$  there is a graph  $L_n = (V_n, E_n, \Sigma, \lambda_n) \in \mathcal{G}$ , where  $\Sigma = \{l, r\}$ ,  $V_n = [n]$ ,  $E_n = \cup_{i < n} \{(i, i+1), (i+1, i)\}$ , and the label  $\lambda_n$  of an edge of the form  $(i, i+1)$  is  $r$ , and of the form  $(i+1, i)$  is  $l$ . We now describe how, given a 2CM  $\mathfrak{M}$ , one

can construct a robot ensemble  $\bar{R}$  which can, on long enough lines, simulate the computation of  $\mathfrak{M}$ . Our robots have very limited sensing abilities: a robot can only sense if it at one of the two ends of the line or not, and it can sense which of the other robots are in the same node as it is (“collision detection”). Note that a robot does not know that another robot has collided with it (and then moved on) if it is not scheduled while they both occupy the same node.

The basic encoding uses two counter robots  $C_1$  and  $C_2$ . The current position of  $C_i$  on the line corresponds to the current value of counter  $i$ , and it moves to the right to increment counter  $i$  and to the left to decrement it. Each of these robots also stores in its finite memory the current state of the 2CM. One difficulty with this basic encoding is how to ensure that the two counter robots always stay synchronised in the sense that they both agree on the next command to simulate, i.e., we need to prevent one of them from “running ahead”. A second difficulty is how to update the state of the 2CM stored by a counter robot when it simulates a command that is a test for zero of the other counter. Note that both of these difficulties are very easy to overcome if one robot can remotely sense the state/position of the other robot. Since we disallow such powerful sensing these difficulties become substantially harder to overcome. The basic idea used to overcome the first difficulty is to add synchronisation robots and have a counter robot move only if it has collided with the appropriate synchronization robot. Thus, by arranging that the synchronization robots collide with the counter robots in a round-robin way the latter alternate their simulation turns and are kept coordinated. In order to enforce this round-robin behavior we have to change the encoding such that only every other position on the line is used to encode the counter values. Thus, an increment or a decrement is simulated by a counter robot moving two steps (instead of one) in the correct direction. The basic ingredient in addressing the second difficulty is to add a *zero-test* robot that, whenever one counter is zero, moves to the position of the other counter’s robot, thus signaling to it that the first counter is zero.

**Theorem 1.** *For every 2CM  $\mathfrak{M}$ , there is a robot ensemble  $\bar{R}$  which, for every  $n \geq 5$ , simulates on the line  $L_n$  any prefix of the computation of  $\mathfrak{M}$  in which the counters never exceed the value  $(n - 3)/2$ .*

*Proof.* The ensemble  $\bar{R}$  consists of 9 robots: the *counter* robots  $C_0, C_1$ , four *synchronisation* robots  $R_0, R_1, R_2, R_3$ , a *zero-test* robot  $T$ , a *zero* robot  $Z$  that marks the zero position of the counters, and a *mover* robot  $M$  whose role is to ensure that the robots can simulate more than one command of  $\mathfrak{M}$  only if their starting positions on the line are as in the *initialised configuration* escribed below ((‡)). The value of a counter is encoded as half the distance between the corresponding counter robot and the  $Z$  robot (e.g., if  $Z$  is in node 3 and  $C_1$  is in node 7 then the value of counter 1 is 2).

((‡)) (*initialised configuration*): robots  $R_2, R_3$  are in node 1, robots  $R_0, R_1$  are in node 2, and the rest are in node 3.

The definition of the transitions of the robots has the important property that there is only one possible run starting from the initialised configuration,

i.e., at each point in time exactly one robot has exactly one transition with a test that evaluates to *true*. We assume that each robot remembers if it is at an odd or even node. This can be done even without looking at the node by storing the parity of the number of steps taken since the initialised configuration ( $\ddagger$ ).

Each command of the 2CM  $\mathfrak{M}$  is simulated by the ensemble using 4 phases. For every  $i \in \{0, 1, 2, 3\}$ , phase  $i$  has the following internal stages: (1) the synchronization robots arrange themselves to signal to robot  $R_i$  that it can start moving to the right (this mechanism is described below  $(\star)$ ). (2) robot  $R_i$  moves to the right until it collides with robot  $C_j$  (where  $j = i \bmod 2$ ). It is an invariant of the run that this collision is at an even node if  $i$  is odd, and vice-versa. (3) robot  $C_j$  moves one step to the left or to the right, in order to simulate the relevant half of the current command of  $\mathfrak{M}$ , as described below  $(\dagger)$ . (4) robot  $R_i$  moves to the right until it reaches the end of the line. Observe that if during this stage  $R_i$  collides with  $C_j$  then (unlike in stage 2) it is on a node with the same parity as  $i$  (by the invariant, and since  $C_j$  moved one step in stage 3). This parity information is used by  $C_j$  to know that it should not move, and by  $R_i$  to know that it can continue moving to the right. (5) robot  $R_i$  moves to the left until it reaches the beginning of the line (see  $(\star)$ ), which ends the phase (here, as in the previous stage, the parity information is used to ignore collisions with  $C_j$ ). In case the other counter (i.e., counter  $1 - j$ ) is zero, stage (2) of phases 0, 1 are modified as follows: when robot  $R_i$  enters node 3 from the left it collides with  $Z, C_{1-j}$  and  $T$ ; then,  $T$  and  $R_i$  move to the right together, where  $T$  always goes first, and then  $R_i$  follows in lock-step; at the end of stage 2 both  $T$  and  $R_i$  collide with  $C_j$ , thus signalling to the latter that counter  $1 - j$  is zero. A similar modification to stages (4) and (5) makes  $T$  and  $R_i$  move in lock-step fashion all the way to the right and then back to the left depositing  $T$  back in node 3.

$(\dagger)$ : The operation performed by  $C_j$  in stage (3) of each phase is as follows. In phase 0 robot  $C_0$  simulates the first half of the command, in phase 1 robot  $C_1$  simulates the first half of the same command, in phase 2 robot  $C_0$  simulates the second half of the command and in phase 3  $C_1$  does so. For example, if the command is “increment counter 0” then in phase 0 robot  $C_0$  moves right one step (and updates its simulated state of  $\mathfrak{M}$  to be the next command of  $\mathfrak{M}$ ), in phase 1 robot  $C_1$  moves right one step (and updates its simulated state of  $\mathfrak{M}$ ), in phase 2 robot  $C_0$  moves again one step to the right (thus encoding an incremented counter), and in phase 3 robot  $C_0$  moves left one step (thus, returning to its previously encoded value). Simulating the other three increment and decrement commands is done similarly. The only other command we need to simulate is of the form ”if counter  $i$  is zero goto state  $p$  else goto state  $q$ ”. Since this command does not change the value of any counter it is simulated by each counter robot going right in the first half of the simulation and left in the second half. The internal state of  $\mathfrak{M}$  is updated to  $p$  or  $q$  depending on the value of the counter. When simulating the first half of the command, robot  $C_j$  knows that counter  $j$  (resp.  $1 - j$ ) is zero iff it sees  $Z$  (resp.  $T$ ) with it.

$(\star)$ : We now show that every arrangement of the synchronization robots uniquely determines which one of them its turn it is to move. Let  $\text{NEXT}(i) :=$

$i + 1 \bmod 4$ ,  $\text{PREV}(i) := i - 1 \bmod 4$ . An *initial arrangement for phase  $i$*  is of the following form:  $R_{\text{PREV}(\text{PREV}(i))}, R_{\text{PREV}(i)}$  are in node 1, and  $R_i, R_{\text{next}(i)}$  are in node 2. Note that the initialised configuration ( $\ddagger$ ) contains the initial arrangement for phase 0. We let the initial arrangement for phase  $i$  signal that the next robot to move is  $R_{\text{PREV}(\text{PREV}(i))}$ , which moves to the right, thus completing stage (1) of phase  $i$ . Hence, at the beginning of stage 2 the arrangement is such that only  $R_{\text{PREV}(i)}$  is left in node 1, which signals that  $R_i$  is the next robot to move, as needed for stage (2). Just before the end of stage (5), robot  $R_i$  returns to node 2 from the left, and the above arrangement repeats itself. Hence, again it is  $R_i$  that moves, however, this time to the left (as indicated by its now different internal memory). The resulting arrangement at the end of phase  $i$  is thus:  $R_{\text{prev}(i)}, R_i$  are in node 1 and  $R_{\text{NEXT}(i)}, R_{\text{PREV}(\text{PREV}(i))}$  are in node 2. Observe that this is exactly the initial arrangement for phase  $\text{NEXT}(i)$ , as required. Note that since robots have collision tests a robot can tell by sensing which other robots are with it (and which are not) exactly which arrangement of the ones described above it is in, and thus if it is allowed to move or not.

Finally, we describe how to amend the construction above by incorporating the robot  $M$  to ensure that robots can only simulate the 2CM if they happen to begin in the initialised configuration ( $\ddagger$ ), and otherwise the system deadlocks after a few steps without any robot entering a halting state.<sup>6</sup> Add to every transition of robot  $R_i$ , for  $i \in \{0, 1, 2, 3\}$ , the additional guard that  $M$  is on the same node with it. Thus,  $M$  enables the synchronisation robots to move, and if  $M$  ever stops, then so does the simulation. Robot  $M$  behaves as follows. It first verifies that the rest of the robots are in the initialised configuration by executing the following sequence (and stopping forever if any of the conditions in the sequence fail to hold): check that it is alone on the right-hand side of the line, move left until it collides with  $C_0, C_1, Z, T$ , move one step left and check that it collides with  $R_0, R_1$ , move one step left and check it is on the left-hand side of the line and collides with  $R_2, R_3$ . Once it verified that the robots are on the nodes specified by ( $\ddagger$ ), it starts “chasing after” the currently active synchronisation robot, i.e., it remembers which robot is active and the direction it moves in, and moves in that direction (if it does not currently collide with that robot).  $\square$

From the previous theorem we can easily deduce that  $\mathfrak{M}$  halts iff there is a run of the ensemble  $\overline{R}$  (on a long enough line, and that fully simulates the run of  $\mathfrak{M}$ ) and in which the robots  $C_0, C_1$  halt. We thus get:

**Corollary 1.** *Let  $k = 9$ ,  $\mathcal{G}$  be the set of lines,  $\mathcal{R}$  the set of  $k$ -robot ensembles consisting of robots whose only tests are local collision tests and the ability to test the left (resp. right) end of the line,  $\Omega$  the set of all  $k$ -orderings, and  $\text{T}$  the MRTL formula  $(\forall \bar{x})(\forall \bar{y})(\forall \bar{X})\neg Halt_{\exists}^{\{1\}}(\bar{X}, \bar{x}, \bar{y})$ .<sup>7</sup> Then  $\text{PVP}_{\text{T}, \Omega}(\mathcal{G}, \mathcal{R})$  is undecidable.*

---

<sup>6</sup> One can modify the construction to remove the need for the  $M$  robot, however we find the exposition with  $M$  clearer.

<sup>7</sup> The formula expresses “for every initial configuration, and every scheduling of the robots, robot 1 never enters a halting state”.

Suitable changes to the construction in Theorem 1 yield that other tasks, such as “certain robots gather” or “certain robots reconfigure”, are also undecidable.

*Remark 1.* Note that in the construction, starting from the initialised configuration, at most one robot can move at any time. Thus, allowing all robots that can act to act, as in the synchronous model, does not change anything. So, with minor modifications to deal with the initialisation phase, the theorem also holds for the synchronous model. This strengthens the previously known fact that the PVP is undecidable for synchronous robots on a line with remote testing abilities (i.e., robot  $l$  can test if “robots  $i$  and  $j$  are in the same node”) [30].

#### 4.2 Decidability of Multi-Robot Systems with Bounded Switching

The previous section shows that decidability cannot be achieved in very limited situations. However, we now suggest a limitation on the *orderings* which guarantees decidability without requiring any other restrictions. Thus it works on many classes of graphs, robots, and tasks. We first describe, at a high level, the approach we use to solve (restricted cases of) the parameterised verification problem  $\text{PVP}_{T,\Omega}(\mathcal{G}, \mathcal{R})$ , cf. [30]. Suppose we can build, for every  $k$ -ensemble  $\bar{R}$  of robots, a formula  $\phi_{\bar{R}, T, \Omega}$  such that for all graphs  $G \in \mathcal{G}$  the following are equivalent: i)  $G \models \phi_{\bar{R}, T, \Omega}$  and ii)  $\bar{R}$  achieves task  $T$  on  $G$  with orderings restricted to  $\Omega$ . Then, for every  $\mathcal{R}$  and  $\mathcal{G}$ , we would have reduced the parameterised verification problem  $\text{PVP}_{T,\Omega}(\mathcal{G}, \mathcal{R})$  to the  $\Phi_{\mathcal{R}, T, \Omega}$ -validity problem for  $\mathcal{G}$  where  $\Phi_{\mathcal{R}, T, \Omega}$  is the set of formulas  $\{\phi_{\bar{R}, T, \Omega} : \bar{R} \in \mathcal{R}\}$ . We now show how to build an MSOL-formula  $\phi_{\bar{R}, T, \Omega}$  in case  $T$  is a formula of MRTL and  $\Omega$  is a finite set of finite orderings.

We begin with a lemma that will be used as a building block. In the simplest setting, the lemma says that for every  $i \in [k]$  there is an MSOL formula with free variables  $\bar{x}, \bar{y}$  that holds on a graph  $G$  if and only if robot  $i$  can move in  $G$  from  $x_i$  to  $y_i$  while all the other robots are frozen, i.e.,  $x_j = y_j$  for  $j \neq i$ .

**Lemma 1 (From Robots to MSOL).** *Fix  $k$ , and let  $\bar{R}$  be a  $k$ -robot ensemble over instruction set  $\text{INS}_{\Sigma,k}$ . For every  $\bar{p}, \bar{q} \in \prod Q_i$  ( $k$ -tuples of states) and ordering  $\alpha \in [k]^+$ , one can effectively construct an  $\text{MSOL}(\Sigma)$  formula  $\psi_{\alpha, \bar{p}, \bar{q}}(\bar{X}, \bar{x}, \bar{y})$  with free variables  $X_i, x_i, y_i$  ( $i \in [k]$ ) such that for every graph  $G$ :  $G \models \psi_{\alpha, \bar{p}, \bar{q}}(\bar{X}, \bar{x}, \bar{y})$  if and only if there exists a run  $c$  of  $\bar{R}$  on  $G$  according to a schedule that follows  $\alpha$ , starting from configuration  $c_1 = \langle \bar{x}, \bar{p} \rangle$  and reaching, for some  $T \in \mathbb{N}$ , the configuration  $c_T = \langle \bar{y}, \bar{q} \rangle$ , such that for all  $i \in [k]$ , the set of positions of robot  $i$  on  $c_1 c_2 \dots c_T$  is contained in  $X_i$ .*

*Similarly one can construct  $\psi_{\alpha, \bar{p}, \bar{q}}^\infty(\bar{X}, \bar{x}, \bar{y})$  so that for every graph  $G$ :  $G \models \psi_{\alpha, \bar{p}, \bar{q}}^\infty(\bar{X}, \bar{x}, \bar{y})$  if and only if there exists a run  $c$  of  $\bar{R}$  on  $G$  according to a schedule that follows  $\alpha$ , starting from configuration  $c_1 = \langle \bar{x}, \bar{p} \rangle$  and reaching the configuration  $\langle \bar{y}, \bar{q} \rangle$  infinitely often, and such that the set of positions of robot  $i$  on the run is contained in  $X_i$  ( $i \in [k]$ ).*

*Proof.* Fix  $k$  and  $\bar{R}$ . We start with an auxiliary step. For  $i \in [k]$ , states  $p_i, q_i \in Q_i$ , and  $\bar{s} = (s_1, \dots, s_{i-1}, s_{i+1}, \dots, s_k)$  with  $s_j \in Q_j$ , we define an MSO-formula

$\phi_{i,p_i,q_i,\bar{s}}$  with free variables  $X, x, y, \bar{z}$  where  $\bar{z} = (z_1, \dots, z_{i-1}, z_{i+1}, \dots, z_k)$  such that  $G \models \phi_{i,p_i,q_i,\bar{s}}$  if and only if the  $k$ -ensemble robot  $\bar{R}$  has a run according to a schedule in  $i^*$  in which robot  $i$  starts in position  $x$  and state  $p_i$ , and reaches position  $y$  and state  $q_i$  while only visiting vertices in  $X$ , and for  $j \neq i$ , robot  $j$  is in vertex  $z_j$  and state  $s_j$  and does not move or change state. This is done as follows. Each robot  $R_i = \langle Q_i, \delta_i \rangle$  is a finite automaton (without initial or final states) over the finite alphabet  $\text{INS}_{\Sigma,k}$ . By Kleene's theorem, we can build a regular expression  $\exp_i$  (that depends on  $p_i, q_i, \bar{s}$ ) over  $\text{INS}_{\Sigma,k}$  for the language of the automaton  $R_i$  with initial state  $p_i$  and final state  $q_i$ . By induction on the regular expressions we build MSOL formulas (with free variables  $X, x, y, \bar{z}$ ):

- $\varphi_\emptyset := \text{false}$  and  $\varphi_\epsilon := x = y \wedge x \in X$ ,
- if  $\tau \in \text{MSOL}_k(\Sigma)$  is a position-test then
  - $\varphi_{\tau \rightarrow \uparrow_\sigma} := \tau(z_1, \dots, z_{i-1}, x, z_{i+1}, \dots, z_k) \wedge \text{edg}_\sigma(x, y) \wedge x, y \in X$ ,
  - $\varphi_{\tau \rightarrow \circlearrowleft} := \tau(z_1, \dots, z_{i-1}, x, z_{i+1}, \dots, z_k) \wedge x = y \wedge x \in X$ ,
- if  $\tau$  is a state-test, say “robot  $j$  is in state  $l$ ” (for  $j \neq i$  and  $l \in Q_j$ ) then, if  $s_j = l$  then
  - $\varphi_{\tau \rightarrow \uparrow_\sigma} := \text{edg}_\sigma(x, y) \wedge x, y \in X$ ,
  - $\varphi_{\tau \rightarrow \circlearrowleft} := x = y \wedge x \in X$ ,
 and otherwise if  $s_j \neq l$ , then  $\varphi_{\tau \rightarrow \uparrow_\sigma}$  and  $\varphi_{\tau \rightarrow \circlearrowleft}$  are defined to be **false**,
- $\varphi_{r+s} := \varphi_r \vee \varphi_s$ ,
- $\varphi_{r \cdot s} := \exists w [\varphi_r(X, x, w, \bar{z}) \wedge \varphi_s(X, w, y, \bar{z})]$ ,
- $\varphi_{r^*} := \forall Z [(cl_{\varphi_r}(X, Z, \bar{z}) \wedge x \in Z) \rightarrow y \in Z]$  where  $cl_{\varphi_r}(X, Z, \bar{z})$  is defined as  $\forall a, b [(a \in Z \wedge \varphi_r(X, a, b, \bar{z})) \rightarrow b \in Z]$ .

Then, define  $\phi_{i,p_i,q_i,\bar{s}}$  to be  $\varphi_{\exp_i}(X, x, y, \bar{z})$ . To prove the lemma proceed by induction on the length  $l$  of  $\alpha$ . Base case: For  $\alpha = i \in [k]$ , define  $\psi_{i,\bar{p},\bar{q}}(\bar{X}, \bar{x}, \bar{y})$  by  $\bigwedge_{j \neq i} x_j = y_j \wedge X_j = \{x_j\} \wedge \phi_{i,p_i,q_i,\bar{s}}(X_i, x_i, y_i, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_k)$ , where  $\bar{s} = (p_1, \dots, p_{i-1}, p_{i+1}, \dots, p_k)$ , if  $q_j = p_j$  for all  $j \neq i$ , and otherwise the formula is defined as **false**. Inductive case: For  $\alpha \in [k]^+, i \in [k]$ , define  $\psi_{\alpha \cdot i, \bar{p}, \bar{q}}(\bar{X}, \bar{x}, \bar{y})$  by  $\exists \bar{z} \bigvee_{\bar{r}} [\psi_{\alpha, \bar{p}, \bar{r}}(\bar{X}, \bar{x}, \bar{z}) \wedge \psi_{i, \bar{r}, \bar{q}}(\bar{X}, \bar{z}, \bar{y})]$  and  $\bar{r}$  varies over  $\prod Q_i$ . This completes the construction of  $\psi_\alpha$ . The construction of  $\psi_\alpha^\infty$  is similar.  $\square$

In Lemma 1, a variable  $X_i$  designates a set containing – but not necessarily equal to – the positions of robot  $i$  along the run. If one wishes  $X_i$  to designate the exact set of positions visited by robot  $i$  (in order to express, e.g., “exploration”), then one needs to modify the construction of  $\phi_{i,p_i,q_i,\bar{s}}$  in the proof of the lemma.<sup>8</sup> The required modifications are straightforward except for those to the definition of  $\varphi_{r^*}$ , which are more complicated.<sup>9</sup>

<sup>8</sup> In [30] it is wrongly stated that one can transform an MSOL formula that says that there is a run (satisfying some property) that stays within a set  $X$ , to one that says that it also visits all of  $X$ , by simply requiring that  $X$  be a minimal set for which a run satisfying the property exists.

<sup>9</sup> Recall that  $\varphi_{r^*}$  has free variables  $X, x, y, \bar{z}$ , and its semantic in this case is that robot  $i$  can reach  $y$  from  $x$  (with the other robots' positions being  $\bar{z}$ ), visiting exactly  $X$ , using a concatenation of sub-paths each satisfying  $\varphi_r$ . Intuitively,  $\varphi_{r^*}$  existentially quantifies over the stitching points of these sub-paths and uses appropriate sub-formulas that are all satisfied iff one can find sub-paths that can be stitched to lead from  $x$  to  $y$  and that cover all the positions in  $X$ .

Observe that this lemma can be used to express both collaborative and adversarial scheduling. For instance, if  $\Omega$  is a finite set of orderings, the formula  $\bigvee_{\alpha \in \Omega} \psi_{\alpha, \bar{p}, \bar{q}}(\bar{X}, \bar{x}, \bar{y})$  says that there is an ordering  $\alpha \in \Omega$  that the robots can follow to go from  $\bar{x}$  to  $\bar{y}$  while staying in  $\bar{X}$ , i.e., the ordering is chosen collaboratively, while  $\bigwedge_{\alpha \in \Omega} \psi_{\alpha, \bar{p}, \bar{q}}(\bar{X}, \bar{x}, \bar{y})$  expresses that the ordering is chosen adversarially.

Putting everything together, we solve the PVP for finite sets of orderings (and thus for adversarial or co-operative  $b$ -switch orderings  $\Omega_b := \{\alpha : \|\alpha\| = b\}$ ).

**Theorem 2.** *There is an algorithm that given an edge-label set  $\Sigma$ , a number of robots  $k \in \mathbb{N}$ , a formula  $T$  of MRTL $_k$ , a finite set  $\Omega$  of finite  $k$ -orderings, and a description of a context-free set of  $\Sigma$ -graphs  $\mathcal{G}$ , decides PVP $_{T, \Omega}(\mathcal{G}, \mathcal{R})$ , where  $\mathcal{R}$  is the set of all  $k$ -ensembles of robots over  $\text{INS}_{\Sigma, k}$ .*

*Proof.* Given  $\bar{R} \in \mathcal{R}$  build the formula  $\phi_{\bar{R}, T, \Omega}$  by replacing every atomic formula in  $T$  by its definition with respect to  $\bar{R}$ . E.g.,  $\text{Reach}_{\exists}(\bar{X}, \bar{x}, \bar{y})$  is replaced by  $\bigvee_{\alpha \in \Omega} \bigvee_{\bar{p}} \bigvee_{\bar{q}} \psi_{\alpha, \bar{p}, \bar{q}}(\bar{X}, \bar{x}, \bar{y})$ , where  $\bar{p}$  varies over  $\prod_{i \in [k]} I_i$  and  $\bar{q}$  varies over  $\prod_{i \in [k]} Q_i$ . Now, a routine induction on the structure of the formula  $T$  shows that  $G \models_{\bar{R}, \Omega} T$  if and only if  $G \models \phi_{\bar{R}, T, \Omega}$ . By Lemma 1 the formula  $\phi_{\bar{R}, T, \Omega}$  is in MSOL( $\Sigma$ ). Finally, apply the fact that the MSOL-validity problem for context-free sets of graphs  $\mathcal{G}$  is uniformly decidable [9].  $\square$

## 5 Discussion

In [6, 30] (see also the discussion before Theorem 1) it was shown that the PVP is undecidable for two synchronous robots on a line, reachability tasks, and allowing the robots “remote” position-tests. In Section 4.1 we substantially strengthen this result and prove that the problem is still undecidable even if we only allow robots “local” position-tests or even just local “collision tests”, both for robots that move synchronously and asynchronously. The fact that the proof works for both the synchronous and asynchronous models (Remark 1), strongly suggests that limiting the robots’ sensing capabilities may not be a very fruitful direction for decidability. In Section 4.2 we showed that for asynchronous robots, if one imposes a bound on the number of times the robots can switch, then PVP is decidable for very general tasks (i.e., those expressible in a new logic called MRTL), large classes of graphs (i.e., the context-free sets of graphs), and allowing robots very powerful testing abilities (i.e., MSOL position-tests and state-tests). This is the first parameterised decidability result of the PVP for *multiple* robots where the environment is the parameter. Thus, our work indicates that if practitioners want formal guarantees on the correctness of the robot protocols they design, then they could design them in the framework given in this paper (i.e., finite-state, bounded-switching, powerful testing abilities).

A main limitation of our decidability result is the fact that the set of grids is not context-free — grids are the canonical workspaces since they abstract 2D and 3D real-world scenarios. However, this limitation is inherent and not

confined to our formalisation since the parameterised verification problem even for one robot ( $k = 1$ ) on a grid with only “local” tests is undecidable [6, 30]. A second limitation is that robots do not have a rich memory (e.g., they cannot remember a map of where they have visited). Extending the abilities to allow for richer memory and communication will result in undecidability, unless it is done in a careful way. Also, the complexity of the decision procedure we gave is very high. Again this is inherent in the problem since, e.g., already for one robot on trees the PVP with the “explore and halt” task is EXPTIME-complete [30]. We leave for future research the problem of finding decidability results with reasonable complexity for multi-robot systems that are rich enough to capture protocols found in the distributed computing literature, e.g., [5, 24, 14, 15].

## References

1. B. Aminof, S. Jacobs, A. Khalimov, and S. Rubin. Parameterized model checking of token-passing systems. In *VMCAI*, volume 8318 of *LNCS*, pages 262–281. Springer, 2014.
2. B. Aminof, T. Kotek, F. Spegni, S. Rubin, and H. Veith. Parameterized model checking of rendezvous systems. In *CONCUR*, volume 8704 of *LNCS*, pages 109–124. Springer, 2014.
3. B. Aminof, S. Rubin, F. Zuleger, and F. Spegni. Liveness of parameterized timed networks. In M. M. Halldórsson, K. Iwama, N. Kobayashi, and B. Speckmann, editors, *ICALP*, volume 9135 of *LNCS*, pages 375–387. Springer, 2015.
4. C. Auger, Z. Bouzid, P. Courtieu, S. Tixeuil, and X. Urbain. Certified impossibility results for byzantine-tolerant mobile robots. In *SSS*, volume 8255 of *LNCS*, pages 178–190. Springer, 2013.
5. M. A. Bender and D. K. Slonim. The power of team exploration: Two robots can learn unlabeled directed graphs. Technical report, MIT, 1995.
6. M. Blum and C. Hewitt. Automata on a 2-dimensional tape. *SWAT (FOCS)*, pages 155–160, 1967.
7. P. Čermák, A. Lomuscio, F. Mogavero, and A. Murano. Mcmas-slk: A model checker for the verification of strategy logic specifications. In *CAV*, volume 8559 of *LNCS*, pages 525–532. Springer, 2014.
8. R. Cohen, P. Fraigniaud, D. Ilcinkas, A. Korman, and D. Peleg. Label-guided graph exploration by a finite automaton. *T. on Algorithms (TALG)*, 4(4):42, 2008.
9. B. Courcelle and J. Engelfriet. Book: Graph structure and monadic second-order logic. a language-theoretic approach. *Bull. EATCS*, 108:179, 2012.
10. S. Das. Mobile agents in distributed computing: Network exploration. *Bull. EATCS*, 109:54–69, 2013.
11. G. De Giacomo, P. Felli, F. Patrizi, and S. Sardiña. Two-player game structures for generalized planning and agent composition. In M. Fox and D. Poole, editors, *AAAI*, pages 297–302, 2010.
12. G. Delzanno. Parameterized verification and model checking for distributed broadcast protocols. In *ICGT*, volume 8571 of *LNCS*, pages 1–16. Springer, 2014.
13. K. Diks, P. Fraigniaud, E. Kranakis, and A. Pelc. Tree exploration with little memory. *Journal of Algorithms*, 51(1):38–63, 2004.
14. P. Flocchini, G. Prencipe, and N. Santoro. Computing by mobile robotic sensors. In S. Nikoletseas and J. D. Rolim, editors, *Theoretical Aspects of Distributed Computing in Sensor Networks*, EATCS, pages 655–693. Springer, 2011.

15. P. Flocchini, G. Prencipe, and N. Santoro. *Distributed Computing by Oblivious Mobile Robots*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool, 2012.
16. P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Hard tasks for weak robots: The role of common knowledge in pattern formation by autonomous mobile robots. In *Algorithms and Computation*, volume 1741 of *LNCS*, pages 93–102. Springer, 1999.
17. P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. *Theoretical Computer Science*, 345:331 – 344, 2005.
18. L. Gasieniec and T. Radzik. Memory efficient anonymous graph exploration. In H. Broersma, T. Erlebach, T. Friedetzky, and D. Paulusma, editors, *Graph-Theoretic Concepts in Computer Science*, volume 5344 of *LNCS*, pages 14–29. Springer, 2008.
19. Y. Hu and G. De Giacomo. Generalized planning: Synthesizing plans that work for multiple environments. In T. Walsh, editor, *IJCAI*, pages 918–923. AAAI, 2011.
20. A. Khalimov, S. Jacobs, and R. Bloem. PARTY parameterized synthesis of token rings. In N. Sharygina and H. Veith, editors, *CAV*, volume 8044 of *LNCS*, pages 928–933. Springer, 2013.
21. A. Khalimov, S. Jacobs, and R. Bloem. Towards efficient parameterized synthesis. In R. Giacobazzi, J. Berdine, and I. Mastroeni, editors, *VMCAI*, volume 7737 of *LNCS*, pages 108–127. Springer, 2013.
22. P. Kouvaros and A. Lomuscio. Automatic verification of parameterised multi-agent systems. In M. L. Gini, O. Shehory, T. Ito, and C. M. Jonker, editors, *AAMAS*, pages 861–868, 2013.
23. P. Kouvaros and A. Lomuscio. A counter abstraction technique for the verification of robot swarms. In B. Bonet and S. Koenig, editors, *AAAI*, pages 2081–2088, 2015.
24. E. Kranakis, D. Krizanc, and S. Rajsbaum. Mobile agent rendezvous: A survey. In P. Flocchini and L. Gasieniec, editors, *SIROCCO*, volume 4056 of *LNCS*, pages 1–9. Springer, 2006.
25. E. Kranakis, D. Krizanc, and S. Rajsbaum. Computing with mobile agents in distributed networks. In S. Rajasekaran and J. Reif, editors, *Handbook of Parallel Computing: Models, Algorithms, and Applications*, CRC Computer and Information Science Series, pages 8–1 to 8–20. Chapman Hall, 2007.
26. N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
27. L. Millet, M. Potop-Butucaru, N. Sznajder, and S. Tixeuil. On the synthesis of mobile robots algorithms: The case of ring gathering. In P. Felber and V. K. Garg, editors, *SSS*, volume 8756 of *LNCS*, pages 237–251. Springer, 2014.
28. M. L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., 1967.
29. A. Murano and L. Sorrentino. A game-based model for human-robots interaction. In *Workshop "From Objects to Agents" (WOA)*, volume 1382 of *CEUR Workshop Proceedings*, pages 146–150. CEUR-WS.org, 2015.
30. S. Rubin. Parameterised verification of autonomous mobile-agents in static but unknown environments. In G. Weiss, P. Yolum, R. H. Bordini, and E. Elkind, editors, *AAMAS*, pages 199–208, 2015.
31. I. Suzuki. Proving properties of a ring of finite-state machines. *Inf. Process. Lett.*, 28(4):213–214, July 1988.

#### **4.4 Publication**

The rest of this page is intentionally left blank

# Decidability of Parameterized Verification

# Synthesis Lectures on Distributed Computing Theory

## Editor

Jennifer Welch, *Texas A&M University*

Nancy Lynch, *Massachusetts Institute of Technology*

Synthesis Lectures on Distributed Computing Theory is edited by Jennifer Welch of Texas A&M University and Nancy Lynch of the Massachusetts Institute of Technology. The series publishes 50- to 150-page publications on topics pertaining to distributed computing theory. The scope largely follows the purview of premier information and computer science conferences, such as ACM PODC, DISC, SPAA, OPODIS, CONCUR, DialM-POMC, ICDCS, SODA, Sirocco, SSS, and related conferences. Potential topics include, but are not limited to: distributed algorithms and lower bounds, algorithm design methods, formal modeling and verification of distributed algorithms, and concurrent data structures.

## Decidability of Parameterized Verification

Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder

2015

## Impossibility Results for Distributed Computing

Hagit Attiya and Faith Ellen

2014

## Distributed Graph Coloring: Fundamentals and Recent Developments

Leonid Barenboim and Michael Elkin

2013

## Distributed Computing by Oblivious Mobile Robots

Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro

2012

## Quorum Systems: With Applications to Storage and Consensus

Marko Vukolic

2012

## Link Reversal Algorithms

Jennifer L. Welch and Jennifer E. Walter

2011

[\*\*Cooperative Task-Oriented Computing: Algorithms and Complexity\*\*](#)

Chryssis Georgiou and Alexander A. Shvartsman

2011

[\*\*New Models for Population Protocols\*\*](#)

Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis

2011

[\*\*The Theory of Timed I/O Automata, Second Edition\*\*](#)

Dilsun K. Kaynar, Nancy Lynch, Roberto Segala, and Frits Vaandrager

2010

[\*\*Principles of Transactional Memory\*\*](#)

Rachid Guerraoui and Michal Kapalka

2010

[\*\*Fault-tolerant Agreement in Synchronous Message-passing Systems\*\*](#)

Michel Raynal

2010

[\*\*Communication and Agreement Abstractions for Fault-Tolerant Asynchronous Distributed Systems\*\*](#)

Michel Raynal

2010

[\*\*The Mobile Agent Rendezvous Problem in the Ring\*\*](#)

Evangelos Kranakis, Danny Krizanc, and Euripides Markou

2010

Copyright © 2015 by Morgan & Claypool

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means—electronic, mechanical, photocopy, recording, or any other except for brief quotations in printed reviews, without the prior permission of the publisher.

Decidability of Parameterized Verification

Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder

[www.morganclaypool.com](http://www.morganclaypool.com)

ISBN: 9781627057431      paperback  
ISBN: 9781627057448      ebook

DOI 10.2200/S00658ED1V01Y201508DCT013

A Publication in the Morgan & Claypool Publishers series  
*SYNTHESIS LECTURES ON DISTRIBUTED COMPUTING THEORY*

Lecture #13

Series Editors: Jennifer Welch, *Texas A&M University*

Nancy Lynch, *Massachusetts Institute of Technology*

Series ISSN

Print 2155-1626   Electronic 2155-1634

# Decidability of Parameterized Verification

Roderick Bloem and Ayrat Khalimov  
Graz University of Technology, Austria

Swen Jacobs  
Graz University of Technology, Austria  
Saarland University, Saarbrücken, Germany

Igor Konnov, Helmut Veith, and Josef Widder  
Vienna University of Technology, Austria

Sasha Rubin  
University of Naples “Federico II”, Italy

*SYNTHESIS LECTURES ON DISTRIBUTED COMPUTING THEORY #13*



MORGAN & CLAYPOOL PUBLISHERS

## ABSTRACT

While the classic model checking problem is to decide whether a finite system satisfies a specification, the goal of parameterized model checking is to decide, given finite systems  $\mathbf{M}(n)$  parameterized by  $n \in \mathbb{N}$ , whether, for all  $n \in \mathbb{N}$ , the system  $\mathbf{M}(n)$  satisfies a specification. In this book we consider the important case of  $\mathbf{M}(n)$  being a concurrent system, where the number of replicated processes depends on the parameter  $n$  but each process is independent of  $n$ . Examples are cache coherence protocols, networks of finite-state agents, and systems that solve mutual exclusion or scheduling problems. Further examples are abstractions of systems, where the processes of the original systems actually depend on the parameter.

The literature in this area has studied a wealth of computational models based on a variety of synchronization and communication primitives, including token passing, broadcast, and guarded transitions. Often, different terminology is used in the literature, and results are based on implicit assumptions. In this book, we introduce a computational model that unites the central synchronization and communication primitives of many models, and unveils hidden assumptions from the literature. We survey existing decidability and undecidability results, and give a systematic view of the basic problems in this exciting research area.

## KEYWORDS

parametrized model checking, concurrent systems, distributed systems, formal verification, model checking, decidability, cutoffs

# Contents

Acknowledgments .....	xi	
<b>1</b>	<b>Introduction .....</b>	<b>1</b>
1.1	Motivation .....	2
1.2	Who Should Read This Book? .....	4
1.3	Organization of the Book .....	4
<b>2</b>	<b>System Model and Specification Languages .....</b>	<b>5</b>
2.1	Preliminary Terminology and Definitions .....	5
2.2	System Model .....	6
2.2.1	Some Standard Synchronization Primitives .....	9
2.2.2	Runs and Deadlocks .....	12
2.3	Parameterized Family of Uniform Concurrent Systems .....	13
2.4	Parameterized Specifications .....	13
2.4.1	Indexed Temporal Logics .....	14
2.4.2	Action-based Specifications .....	18
2.4.3	Specifications in the Literature .....	18
2.5	Model Checking Problems for Concurrent Systems .....	19
2.5.1	Computability Assumptions .....	20
<b>3</b>	<b>Standard Proof Machinery .....</b>	<b>23</b>
3.1	Techniques to Prove Undecidability of PMCP .....	23
3.2	How to Prove Decidability of the PMCP .....	24
3.2.1	Well-structured Transition Systems .....	25
3.2.2	Vector Addition Systems with States (and Petri Nets) .....	27
3.2.3	Decompositions and Cutoffs .....	28
<b>4</b>	<b>Token-passing Systems .....</b>	<b>31</b>
4.1	System Model .....	32
4.1.1	Direction-aware Parameterized Systems .....	32
4.1.2	Token-passing Systems .....	34
4.2	Results for Direction-unaware Token-passing Systems .....	37

4.2.1	Decidability for Simple Token-passing in Uni-directional Rings . . . . .	38
4.2.2	Decidability for Simple Token-passing in Graphs . . . . .	40
4.2.3	Undecidability Results for Multi-valued Tokens . . . . .	42
4.3	Results for Direction-aware Token-passing Systems . . . . .	44
4.3.1	Cutoffs for Change-bounded Tokens in Bi-directional Rings . . . . .	44
4.3.2	Undecidability for Direction-aware TPSs . . . . .	45
4.4	Discussion . . . . .	46
4.4.1	Variations of the Model . . . . .	48
<b>5</b>	<b>Rendezvous and Broadcast . . . . .</b>	<b>51</b>
5.1	System Model . . . . .	51
5.2	Decidability Results . . . . .	55
5.2.1	Counter Representation . . . . .	55
5.2.2	Decidability for All Three Primitives . . . . .	58
5.2.3	Decidability for Pairwise Rendezvous . . . . .	58
5.3	Undecidability Results . . . . .	59
5.3.1	Undecidability for Broadcast . . . . .	59
5.3.2	Undecidability for Asynchronous Rendezvous . . . . .	62
5.4	Discussion . . . . .	62
5.4.1	Variations of the Model . . . . .	64
<b>6</b>	<b>Guarded Protocols . . . . .</b>	<b>65</b>
6.1	Motivating Example . . . . .	65
6.2	System Model . . . . .	69
6.2.1	Classes of Guarded Protocols . . . . .	73
6.2.2	Specifications . . . . .	74
6.3	Undecidability: Boolean and Conjunctive Guards . . . . .	76
6.4	Decidability: Init-Conjunctive and Disjunctive Guards . . . . .	81
6.4.1	Preliminaries . . . . .	82
6.4.2	Proof Schemas . . . . .	84
6.4.3	Init-conjunctive Guards . . . . .	86
6.4.4	Disjunctive Guards . . . . .	89
6.5	Disjunctive Guards vs. Rendezvous . . . . .	96
6.6	Variations on the Model: Guards and Synchronization Primitives . . . . .	99
6.7	Discussion . . . . .	100

<b>7</b>	<b>Ad Hoc Networks .....</b>	<b>103</b>
7.1	Running Example .....	103
7.2	System Model.....	104
7.3	PMC Problems for Ad Hoc Networks .....	105
7.4	Parameterized Connectivity Graphs $\mathbf{BP}_k$ , $\mathbf{BPC}_k$ , $\mathbf{BD}_k$ , $\mathbf{C}$ , All .....	107
7.5	Results for (Non-lossy) AHNs .....	108
7.5.1	Undecidability Results for (Non-lossy) AHNs .....	108
7.5.2	Decidability Results for (Non-lossy) AHNs .....	116
7.6	Decidability Results for Lossy Ad Hoc Networks.....	123
7.7	Discussion .....	127
7.7.1	Variations of the Model .....	128
<b>8</b>	<b>Related Work .....</b>	<b>129</b>
8.1	Abstraction Techniques .....	129
8.2	Regular Model Checking .....	131
8.3	Symbolic Techniques .....	132
8.4	Dynamic Cutoff Detection .....	134
8.5	Network Invariants .....	135
8.6	Invisible Invariants .....	137
8.7	Other Aspiring Approaches .....	137
<b>9</b>	<b>Parameterized Model Checking Tools .....</b>	<b>139</b>
<b>10</b>	<b>Conclusions .....</b>	<b>143</b>
	<b>Bibliography .....</b>	<b>145</b>
	<b>Authors' Biographies .....</b>	<b>157</b>



## Acknowledgments

We are grateful to Paul Attie, Giorgio Delzanno, Sayan Mitra, and Kedar Namjoshi for carefully reading an earlier draft of this manuscript, and providing detailed and constructive comments.

Supported by the Austrian National Research Network RiSE (S11403, S11405, S11406) and project PRAVDA (P27722) of the Austrian Science Fund (FWF), by the Vienna Science and Technology Fund (WWTF) through grant PROSEED, by the German Research Foundation (DFG) through SFB/TR 14 AVACS and project ASDPS (JA 2357/2-1), and by the Istituto Nazionale di Alta Matematica through INdAM-COFUND-2012, FP7-PEOPLE-2012-COFUND (Proj. ID 600198).

Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and  
Josef Widder  
August 2015



## CHAPTER 1

# Introduction

Reasoning about the correctness of concurrent and distributed systems is an inherently difficult task. A main challenge comes from the fact that many processes run concurrently and interact with each other, which results in non-determinism and large execution and state spaces. Thus, it is easy for a system designer to miss a bug related to concurrency, e.g., an unforeseen race condition, deadlock, livelock, etc. Finding these kinds of bugs motivated research in model checking for many years [Baier and Katoen, 2008, Clarke et al., 1999, Emerson and Clarke, 1980, Grumberg and Veith, 2008, Queille and Sifakis, 1982]. The central problem for model checking (and of many other techniques) is state explosion. Research in model checking made several breakthroughs and developed methods to deal with this problem. Over the years, these methods have been implemented in industrial tools, which are used to verify hardware designs (e.g., microprocessors and cache coherence protocols), predominantly sequential software (e.g., device drivers), and network protocols [Grumberg and Veith, 2008]. The main line of research in model checking of concurrent systems considers systems with a fixed and *a priori* known number of processes.

Some designs, however, need to be proven to work independently of the system size (e.g., hardware protocols, network protocols, distributed algorithms). The engineering intuition says that if a system has been carefully debugged on a small number of processes, then it should also work with a large number of processes. This intuition is sometimes wrong. For instance, Clarke et al. [1992] applied model checking on the Futurebus+ cache coherence protocol for small sizes. After discovering several bugs, Clarke et al. proposed an improved version of the protocol, and verified that the new version of the protocol is correct on various instances of small size. Later, Kesten et al. [1997] performed model checking on the new protocol for larger system sizes and discovered new bugs.

This book focuses on the verification of systems where the number of processes is not known *a priori*. Because the number of processes is not known, the challenge is to verify a system for all system sizes. This motivates the study of *parameterized model checking*. In this book we survey results regarding the decidability of this problem for concurrent systems, and we thus make explicit the limits and strengths of automated verification methods. For instance, we will see cases where one can prove that it is sufficient to verify only system instances of small size, that is, special cases where the intuition mentioned above is actually justified. However, we will also see that parameterized model checking is undecidable for many interesting classes.

These decidability results, and the border between decidability and undecidability of the parameterized model checking problem, are the topics of this book. In the decidable cases we,

## 2 1. INTRODUCTION

in general, do not cover complexity results. Many existing formalisms (e.g., Petri-nets, process algebra) are related to concurrent systems, and complexity results regarding these formalisms may carry over to the parameterized model checking problem. As there is a wealth of complexity results related to such formalisms we cannot cover them in this book. Also, we do not cover infinite-state systems such as automata with unbounded channels or external storage, or concurrent systems in all generality. We focus on concurrent systems that are composed of a parameterized number of identical finite-state processes, where the local state space is independent of the parameter. Consequently, each system instance we consider has a finite state space.

The rationale of this book is to give an overview of the positive and negative results, and to ease the understanding by expressing system models and proofs in a unified model. Thereby, we hope to shed some light on the border between decidability and undecidability of the parameterized model checking problem.

### 1.1 MOTIVATION

If the number of components in a concurrent system is known a priori and every component has finitely many states, then the verification problem corresponds to finite-state *model checking* [Baier and Katoen, 2008, Clarke et al., 1999]: given the specification as a temporal logic formula  $\phi$ , and given the system with  $n$  components as a finite-state transition system  $M$ , check whether  $M$  satisfies  $\phi$ . In practice, reasoning about properties of concurrent systems is complicated because the interleaving of steps leads to combinatorial explosion.

Many protocols like cache coherency or bus protocols are defined as *parameterized* systems. Intuitively, a parameterized system is a sequence  $(\mathbf{M}(n))_{n \in \mathbb{N}}$  of systems, where  $\mathbf{M}(n)$  consists of  $n$  copies of similar or identical processes. The *parameterized model-checking problem* (PMCP) is then to decide whether  $\mathbf{M}(n)$  satisfies the specification for every  $n$ . We survey the work in this area, focusing on decidability and undecidability results, and putting less emphasis on pragmatic solutions for real-life systems.

A tempting approach to the PMCP is to check the system for small values of  $n$  and *assume* that if there is a bug in a system with a large number of components, then it already appears in a small system. This assumption is widely used; for instance, Lamport [2006, p. 13] writes “Almost every error manifests itself on a very small instance—one that may or may not be too large to model check.” For specific classes of systems, this approach can be formalized as a *cutoff* or *decomposition* statement [Clarke et al., 2004, Emerson and Namjoshi, 1995] that reduces the PMCP to a finite collection of classic model checking problems.

Unfortunately, not all bugs of large systems can be found in small ones, and the PMCP is undecidable in general. Indeed, Apt and Kozen [1986] formalized the PMCP and were the first to address the question of its decidability. However, they considered systems where the implementation of a single process  $S$  depends on the parameter  $n$ , and treated concurrency only superficially. To prove undecidability of the PMCP, Apt and Kozen considered the following program: Given

a deterministic Turing Machine  $T$ , let  $S(n)$  be the finite-state system represented by the following code:

```
flag := false
for i := 1 to n do
    simulate one step of T
if T has not halted then flag := true
```

Hence, the system  $S(n)$  simulates the first  $n$  steps of the Turing machine  $T$ . Let  $\phi$  be the formula stating that eventually the flag is set to true—in linear temporal logic this is written as  $\mathbb{F}flag$ . Then  $T$  does not halt if and only if  $\forall n \in \mathbb{N}. S(n) \models \phi$ . As the non-halting problem is undecidable, one immediately finds the PMCP to be undecidable. Furthermore, as  $S(n)$  is a degenerate case of a concurrent system (consisting of only one process), one may conclude that the PMCP for concurrent systems is undecidable in general.

However, the undecidability result by [Apt and Kozen \[1986\]](#) does not directly apply to a particularly interesting class of concurrent systems, namely those that are composed of  $n$  copies of a finite-state process  $P$ , where  $P$  is independent of  $n$ . We call these *uniform concurrent systems*. Examples of such systems solve classic problems such as mutual exclusion [[Pnueli et al., 2002](#), [Wolper and Lovinfosse, 1989](#)], or scheduling [[Milner, 1989](#)]. For specific fault-tolerant distributed broadcasting algorithms [[Srikanth and Toueg, 1987](#)], [John et al. \[2013\]](#) obtained a uniform concurrent system after a data abstraction of the parameterized process code of the broadcasting algorithm. Hence, PMCP of specific uniform concurrent systems can also be used as part of a tool chain in the verification of concurrent systems where  $P$  actually depends on  $n$ .

Whether the PMCP for a uniform concurrent system is decidable depends on several factors, the most important being the underlying communication graph (e.g., rings, stars, cliques), and the means of synchronization (e.g., token passing with/without information-carrying tokens, handshake). For instance, [Suzuki \[1988\]](#) proved that [Apt and Kozen](#)'s idea of reducing the PMCP to the non-halting problem can be extended to token rings in which tokens carry information. The basic idea is to use each of the  $n$  replicated finite-state processes to store the content of one cell of the tape of a Turing machine, and implement the movement of the head by shifting information around the ring using tokens. As  $n$  cells can be stored in a system with  $n$  processes, such a system can be used to simulate  $n$  steps of a Turing machine, and undecidability follows as in [Apt and Kozen \[1986\]](#). Suzuki's construction is prototypical of the undecidability results surveyed in this book.

However, the parameterized model checking results in the literature are not limited to token rings. Rather, there are many different computational models that differ in the underlying graph, and the means of communication. These computational models are scattered over the literature, use different terminology, with slightly different assumptions.

Hence, if one is faced with a PMCP in a given system, it is difficult to tell whether there is a published computational model that naturally captures the system's semantics. The main goal of this work is to provide a *one-stop source* for existing models for asynchronous *uniform concurrent*

#### 4 1. INTRODUCTION

*systems*, along with known (un)decidability results. To this end, we provide a single definition that incorporates many of the foundational computational models that have appeared in the parameterized model checking literature on undecidability and decidability. The later chapters then specialize specific features of the general model to systems that can be found in the literature, and discuss the applicable (un)decidability results.

### 1.2 WHO SHOULD READ THIS BOOK?

This book was written for people who are interested in the principles of concurrent systems and parameterized model checking. This includes graduate students and senior undergraduate students in computer science who are interested in the basic principles of parameterized model checking for concurrent systems. For researchers this book surveys the principles that have been used to obtain decidability and undecidability results in the context of parameterized model checking. We are convinced that many of the techniques surveyed in this book can be used to derive new results in the field.

### 1.3 ORGANIZATION OF THE BOOK

In Chapter 2 we introduce definitions of a computational model and parameterized specifications that cover most of the existing decidability results in parameterized model checking. In Chapter 3 we explain foundational ideas from the literature for proving decidability or undecidability of different subclasses of the PMCP.

Subsequently, we consider four classes of systems where decidability results have been obtained.

1. The well-studied class of *token-passing systems*, where communication is restricted to passing a single token among all processes. This includes the important special case of *token rings* (Chapter 4).
2. Systems where communication is based on synchronized steps between two processes, called *pairwise rendezvous*, or one-to-many synchronization, called *broadcast* (Chapter 5).
3. *Guarded protocols*, where local transitions of one process can be restricted with respect to the global state of the system (Chapter 6). This includes combinations with other forms of synchronization, like broadcast and rendezvous.
4. *Ad-hoc networks*, where broadcast communication between processes may be lossy, or equivalently, communication networks can change during runtime (Chapter 7).

For all classes, we survey decidability and undecidability results from the literature. After briefly discussing related work that is not surveyed in this book in Chapter 8 and giving an overview of existing parameterized model checking tools in Chapter 9, we conclude in Chapter 10.

## CHAPTER 2

# System Model and Specification Languages

In this chapter we present definitions of parameterized systems, parameterized specifications and the parameterized model-checking problem. The rationale behind our definitions is to make explicit many of the commonalities of the different system models and results found in the literature.

A concurrent system  $\overline{P}^G$  (called a *system instance*) is composed of a vector of *process templates*  $\overline{P} = (P^1, \dots, P^d)$ , with copies of the templates arranged on a graph  $G$  (called the *connectivity graph*). We consider discrete time, and at each time step either one process acts alone, or some process  $v$  initiates an action and some set of processes that are connected to  $v$  in  $G$  simultaneously synchronize with  $v$ .

Synchronization primitives result from restricting the number of processes that can simultaneously synchronize with the initiating process. We capture a given synchronization primitive (e.g., pairwise rendezvous, broadcast) by a *synchronization constraint*  $\text{card} \subseteq \mathbb{N}_0$ . Roughly, the number of synchronizing processes should be a number in  $\text{card}$ . For instance, pairwise rendezvous is captured by defining  $\text{card} = \{1\}$ : every synchronous transition must be taken by the initiating process and exactly one other process.

We define  $\overline{P}^G$  as a labeled transition system, and thus need to specify its labeling function. If  $p$  is an atomic proposition of the process template and  $v$  is a vertex in  $G$ , then  $p_v$  is an atomic proposition of the composed system—an *indexed atomic proposition*—meaning that  $p$  holds in the current state of the process at vertex  $v$ .

Then, a *parameterized system* is a sequence of system instances formed from a fixed vector of process templates and a sequence of graphs  $\mathbf{G}$ . Typical sequences of graphs are rings of size  $n$ , cliques of size  $n$ , or stars of size  $n$ , where  $n$  is the parameter. The  $n$ th instance of a parameterized system is the system instance  $\overline{P}^{\mathbf{G}(n)}$ . *Parameterized specifications* make statements about parameterized systems by quantifying over process indices (i.e., vertices of  $\mathbf{G}(n)$ ), e.g., “every process  $v$  of  $\mathbf{G}(n)$  eventually satisfies  $p$ .” The *parameterized model checking problem* is to decide whether for all  $n \in \mathbb{N}$  the instance  $\overline{P}^{\mathbf{G}(n)}$  satisfies the specification.

## 2.1 PRELIMINARY TERMINOLOGY AND DEFINITIONS

A *labeled transition system (LTS)* over  $\text{AP}$  is a tuple  $(Q, Q_0, \Sigma, \delta, \lambda)$ , where  $\text{AP}$  is a set of *atomic propositions* or *atoms*,  $Q$  is the set of *states*,  $Q_0 \subseteq Q$  are the *initial states*,  $\Sigma$  is the set of *transition*

## 6 2. SYSTEM MODEL AND SPECIFICATION LANGUAGES

*labels* (also-called *action labels*),  $\delta \subseteq Q \times \Sigma \times Q$  is the *transition relation*, and  $\lambda : Q \rightarrow 2^{\text{AP}}$  is the *state-labeling* and satisfies that  $\lambda(q)$  is finite (for every  $q \in Q$ ). A *finite LTS* is an LTS in which  $\text{AP}$  and  $Q$  are finite. Transitions  $(q, a, q') \in \delta$  may be written  $q \xrightarrow{a} q'$ . A *transition system* is an LTS without the state-labeling  $\lambda$ , and sometimes without initial states.

A *path of an LTS*  $(Q, Q_0, \Sigma, \delta, \lambda)$  is a finite sequence of the form  $q_0 a_0 q_1 a_1 \dots q_n \in (Q \Sigma)^* Q$  or an infinite sequence of the form  $q_0 a_0 q_1 a_1 \dots \in (Q \Sigma)^\omega$  such that  $(q_i, a_i, q_{i+1}) \in \delta$  for all  $i$ . A *run of an LTS* is a path that starts in an initial state, i.e.,  $q_0 \in Q_0$ . A *state-labeled path* of an LTS is the projection  $q_0 q_1 \dots$  of a path onto states  $Q$ . An *action-labeled path* of an LTS is the projection  $a_0 a_1 \dots$  of a path onto transition labels  $\Sigma$ .

When it is clear from the context, a path may omit either actions or states (e.g., LTL formulas are interpreted over infinite state-labeled paths  $q_0 q_1 \dots$ ).

## 2.2 SYSTEM MODEL

Given a finite set of atomic propositions  $\text{AP}_{\text{pr}}$  (for individual processes), and process identifiers from the set  $\mathbb{N}$ , define the set of *indexed atomic propositions*  $\text{AP}_{\text{sys}} := \text{AP}_{\text{pr}} \times \mathbb{N}$ . The set of atomic propositions for a system instance with  $k$  processes is  $\text{AP}_{\text{pr}} \times [k] \subseteq \text{AP}_{\text{sys}}$ . For  $(p, i) \in \text{AP}_{\text{sys}}$  we may also write  $p_i$ .

### INGREDIENTS OF SYSTEM INSTANCE

**Transition labels.** Write  $\Sigma_{\text{int}}$  for a finite non-empty set of *internal transition labels* and  $\Sigma_{\text{sync}}$  for a finite non-empty set of *synchronous transition labels*. Throughout this book, when we use  $\tau$ , we assume that is an action with  $\tau \in \Sigma_{\text{int}}$ . Define  $\Sigma_{\text{pr}}$  as the disjoint union  $\Sigma_{\text{int}} \cup \{out_a : a \in \Sigma_{\text{sync}}\} \cup \{in_a : a \in \Sigma_{\text{sync}}\}$ , where  $out_a$  and  $in_a$  are new symbols. An action  $out_a$  will be called an *initiate action* and an action  $in_a$  a *receive action*. Formally,  $in_a$  may be defined as  $(a, 0)$  and  $out_a$  as  $(a, 1)$ .

**System-arity d.** Let  $d$  be a positive integer called the *system-arity* or just *arity*.

**d-ary system template  $\bar{P}$ .** A  $d$ -ary system template is a  $d$ -tuple  $\bar{P} = (P^1, \dots, P^d)$  where each  $P^\ell$ , called a *process template*, is a finite LTS  $(Q^\ell, Q_0^\ell, \Sigma_{\text{pr}}, \delta^\ell, \lambda^\ell)$  over atomic propositions  $\text{AP}_{\text{pr}}$ . The elements of  $Q^\ell$  are called *local states* — the  $Q$ 's are assumed to be pairwise disjoint — and the transitions in  $\delta^\ell$  are called *local transitions of  $P^\ell$* . Furthermore, we require the modest restriction that every  $\delta^\ell$  is total in the first coordinate: for every  $q \in Q^\ell$  there exists  $\sigma \in \Sigma_{\text{pr}}, q' \in Q^\ell$  such that  $(q, \sigma, q') \in \delta^\ell$ . In case  $d = 1$  we write  $P$  instead of  $\bar{P}$ .

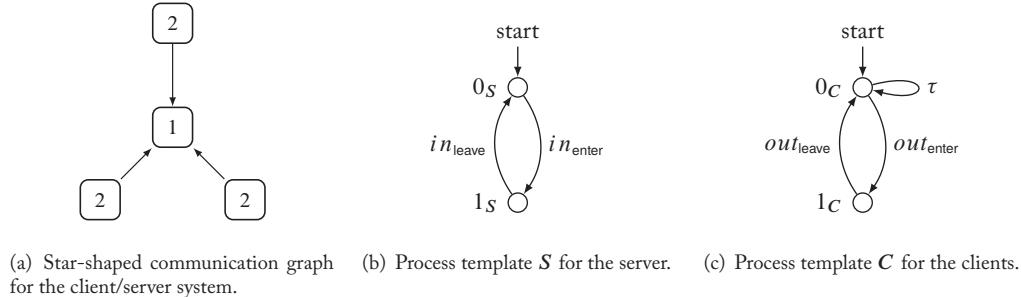
**d-ary connectivity graph G.** A  $d$ -ary connectivity graph is a  $d$ -colored directed graph  $G = (V, E, \text{type})$  where  $V = [k]$  for some  $k \in \mathbb{N}$ ,  $E \subseteq V^2$  and  $\text{type} : V \rightarrow [d]$ . Vertices are called *process indices*. For  $i \leq d$  let  $n_i$  denote the cardinality of  $\{v \in V : \text{type}(v) = i\}$ . If  $(v, w) \in E$  then we say that  $w$  is a *recipient* of  $v$ , and also write  $w \in E(v)$ . In case  $d = 1$  we may write  $G = (V, E)$  instead of  $G = (V, E, \text{type})$ . To distinguish vertices and edges of different graphs, we introduce the

notation  $V(G)$  and  $E(G)$  to refer to the set of vertices and the set of edges of graph  $G$ , respectively. A connectivity graph is sometimes also-called a *topology*.

**Example 2.1 Simple Client/Server System.** As an example, consider a simple client/server system consisting of a server and three clients. In our model, this is captured by a 2-ary connectivity graph with  $V = \{1, 2, 3, 4\}$ : vertex 1 is the server and thus  $\text{type}(1) = 2$ , and vertices 2, 3, and 4 are clients so that  $\text{type}(i) = 1$  for  $i \in \{2, 3, 4\}$ . To model that only the clients can initiate synchronization, we set  $E = \{(2, 1), (3, 1), (4, 1)\}$ . The connectivity graph is depicted in Figure 2.1(a).

Now, suppose the implementations of server and client (as LTSs) are based on internal transition labels  $\Sigma_{int} = \{\tau\}$  and synchronous transition labels  $\Sigma_{sync} = \{\text{enter}, \text{leave}\}$ . Very simple implementations  $S$  and  $C$  for the server and client, respectively, are depicted in Figure 2.1(b) and Figure 2.1(c) (omitting state labels).

In the composed system, client processes can take internal transitions (labeled with  $a$ ) independently of the others, while for the other transitions they need to synchronize with the server. The details of synchronization are defined in the following.



**Figure 2.1:** Communication graph and process templates for simple client/server system. Circles represent local states of a process, labeled with state names.

**Synchronization constraint card.** A *synchronization constraint* is a set  $\text{card} \subseteq \mathbb{N}_0$  of natural numbers. This set is used to define the number of processes that participate in a synchronized action: The idea is that if a process with index  $v$  takes an  $out_a$  action then simultaneously some subset of  $v$ 's recipients take  $in_a$  actions. The cardinality of this subset must be in  $\text{card}$ . See Section 2.2.1 for synchronization cardinalities that correspond to standard synchronization primitives such as pairwise rendezvous, asynchronous rendezvous, and broadcast, in which  $\text{card}$  are very simple sets such as  $\{1\}$ ,  $\{0, 1\}$  and  $\mathbb{N}_0$ .

**Example 2.2 Synchronization in Client/Server System.** In our client/server system in Figure 2.1, consider the synchronization constraint  $\{1\}$ . Intuitively, this means that whenever a client

## 8.2. SYSTEM MODEL AND SPECIFICATION LANGUAGES

wants to take a synchronous transition, labeled  $out_{\text{enter}}$  or  $out_{\text{leave}}$ , it can only do so when there is at least one other process which synchronizes with the client by taking a transition labeled with  $in_{\text{enter}}$  or  $in_{\text{leave}}$ , respectively. In the given communication graph, this other process can only be the server.

In the following definition of a system instance, we formalize the notion of synchronization.

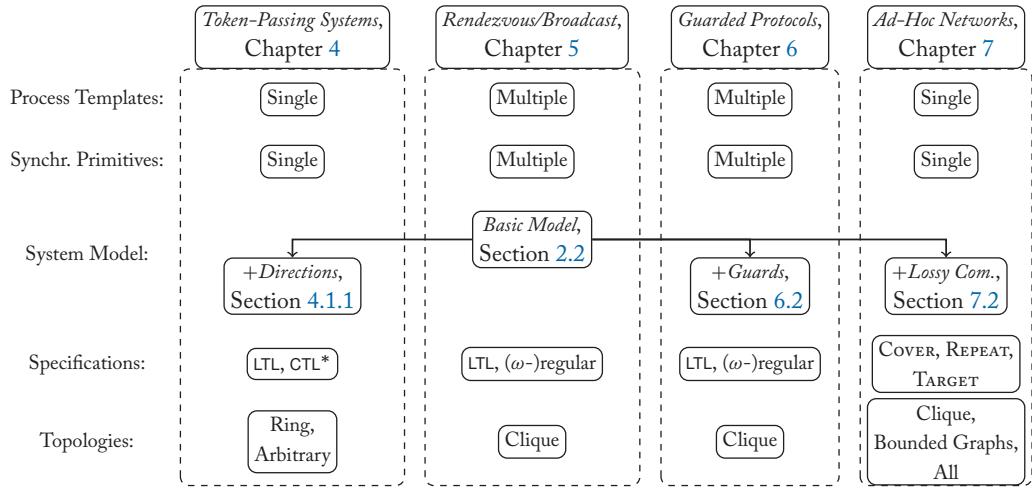
### SYSTEM INSTANCE $\overline{P}^G$

Given a system arity  $d$ , a  $d$ -ary system template  $\overline{P}$ , a  $d$ -ary connectivity graph  $G = (V, E, \text{type})$  with  $|V| = k$ , and a synchronization constraint card, define the *system instance*  $\text{sys}(\overline{P}, G, \text{card})$ , also written  $\overline{P}^G$  if card is clear, as the *finite LTS*  $(S, S_0, \Sigma_{\text{int}} \cup \Sigma_{\text{sync}}, \Delta, \Lambda)$  over indexed atomic propositions  $\text{AP}_{\text{pr}} \times [k]$ , where the following holds true.

- The state set  $S$  consists of all  $|V|$ -tuples  $s = (q_1, \dots, q_{|V|}) \in Q^{\text{type}(1)} \times \dots \times Q^{\text{type}(|V|)}$ . For  $v \in V$ , let  $s(v)$  denote  $q_v$ , the local state of the process at  $v$ . Each  $s$  is called a *global state*.
- The set of *global initial states* is defined as  $S_0 := Q_0^{\text{type}(1)} \times \dots \times Q_0^{\text{type}(|V|)}$ .
- The *global transition relation*  $\Delta \subseteq S \times (\Sigma_{\text{int}} \cup \Sigma_{\text{sync}}) \times S$  is defined as the set of all internal transitions (in which a single process acts) and synchronous transitions (in which some set of processes act simultaneously), where we have the following.
  - An *internal transition* is an element  $(s, a, s')$  of  $S \times \Sigma_{\text{int}} \times S$  for which there exists a process index  $v \in V$  satisfying the following two conditions:
    - (**int-step**)  $s(v) \xrightarrow{a} s'(v)$  is a local transition of process template  $P^{\text{type}(v)}$ .
    - (**int-frame**) For all  $w \in V \setminus \{v\}$ ,  $s(w) = s'(w)$ .
  - A *synchronous transition* is an element  $(s, a, s')$  of  $S \times \Sigma_{\text{sync}} \times S$  for which there exists a process index  $v \in V$ , called the *initiator*, and a set  $\mathcal{I} \subseteq \{w \in V : E(v, w)\}$  of recipients of  $v$  in  $G$  such that:
    - (**card**)  $|\mathcal{I}| \in \text{card}$ .
    - (**step**)  $s(v) \xrightarrow{out_a} s'(v)$  is a local transition of process template  $P^{\text{type}(v)}$ .
    - (**I-step**) For every  $w \in \mathcal{I}$ ,  $s(w) \xrightarrow{in_a} s'(w)$  is a local transition of process template  $P^{\text{type}(w)}$ .
    - (**frame**) For every  $w \in V \setminus (\mathcal{I} \cup \{v\})$ ,  $s'(w) = s(w)$ .
    - (**max**) There does not exist a strict superset  $\mathcal{I}' \supset \mathcal{I}$  of recipients of  $v$  in  $G$  such that
      - (i)  $|\mathcal{I}'| \in \text{card}$ , and (ii) for all  $w \in \mathcal{I}'$  there exists  $r \in Q^{\text{type}(w)}$  such that  $s(w) \xrightarrow{in_a} r$  is a local transition of process template  $P^{\text{type}(w)}$ .
- The labeling  $\Lambda(s) \subseteq \text{AP}_{\text{pr}} \times [k]$  for  $s \in S$  is defined as follows: for  $v \in V$ ,  $p_v \in \Lambda(s)$  if and only if  $p \in \lambda^{\text{type}(v)}(s(v))$ , and for  $v \notin V$ ,  $p_v \notin \Lambda(s)$  for all  $s \in S$ .

### 2.2.1 SOME STANDARD SYNCHRONIZATION PRIMITIVES

We give various synchronization constraints card that model standard forms of communication, namely pairwise rendezvous, asynchronous rendezvous, and broadcast (Chapter 5). Variations of the primitives, along with extensions of our basic system model, are used to define the Token-Passing Systems (Chapter 4), Guarded Protocols (Chapter 6), and Ad-Hoc Networks (Chapter 7); see Figure 2.2.



**Figure 2.2:** Basic system model and extensions in this survey. For topologies, “All” refers to the parameterized topology with all possible topologies in it, while “Arbitrary” refers to an arbitrary parameterized topology.

**Pairwise Rendezvous.** The synchronization constraint  $\text{card} = \{1\}$  captures pairwise rendezvous—i.e., synchronization between pairs of processes. In this case it is customary to write  $a!$  instead of  $\text{out}_a$  and  $a?$  instead of  $\text{in}_a$ .

To illustrate, we instantiate the definition of synchronous transition for  $\text{card} = \{1\}$ . *Pairwise-rendezvous transitions* are of the form  $(s, a, s')$  for which there exists  $(v, w) \in E$  such that:

- (STEP)  $(s(v), a!, s'(v))$  is a local transition of process template  $P^{\text{type}(v)}$ ;
- ( $\mathcal{I}$ -STEP)  $(s(w), a?, s'(w))$  is a local transition of process template  $P^{\text{type}(w)}$ ; and
- (FRAME) for all  $z \notin \{v, w\}$ ,  $s(z) = s'(z)$ .

Note that this incorporates the (CARD) condition with  $\mathcal{I} = \{w\}$ , and the (MAX) condition is trivially satisfied since  $\mathcal{I}' \supset \{w\}$  implies  $|\mathcal{I}'| \not\leq \text{card}$ .

## 10 2. SYSTEM MODEL AND SPECIFICATION LANGUAGES

Figure 2.3(a) illustrates some possible configurations for a pairwise-rendezvous step.

**Asynchronous Rendezvous.** The synchronization constraint  $\text{card} = \{0, 1\}$  captures asynchronous rendezvous. The idea is that a process can synchronize with zero or one other process. The maximality condition (MAX) ensures that asynchronous rendezvous is like pairwise rendezvous except that a process taking a transition labeled  $out_a$  is not blocked when there are no corresponding processes that are able to take a transition labeled  $in_a$ .

Figure 2.3(b) illustrates some possible configurations for an asynchronous-rendezvous step.

**Broadcast.** The synchronization constraint  $\text{card} = \mathbb{N}_0$  captures broadcast. The idea is that if vertex  $v$  wants to broadcast message  $a$  then every recipient of  $v$  that is ready to receive the broadcast message  $a$  does so. If no process is ready to receive the broadcast then since  $0 \in \text{card}$ , the initiating process makes its local transition anyway. For broadcast,  $out_a$  is written  $a!!$  and  $in_a$  is written  $a??$ .

Again, we instantiate the definition of synchronous transition to illustrate. *Broadcast transitions* are of the form  $(s, a, s')$  where there exists  $v \in V$  and a subset  $\mathcal{I} \subseteq V$  of recipients of  $v$ , such that (STEP)  $s(v) \xrightarrow{a!!} s'(v)$  is a local transition of  $P^{\text{type}(v)}$ , ( $\mathcal{I}$ -STEP) for all  $w \in \mathcal{I}$ ,  $s(w) \xrightarrow{a??} s'(w)$  is a local transition of  $P^{\text{type}(w)}$ , (FRAME) for all other  $z$ ,  $s'(z) = s(z)$ , and (MAX) there is no recipient  $w$  of  $v$  not already in  $\mathcal{I}$  such that  $s(w) \xrightarrow{a??} r$  is a local transition of  $P^{\text{type}(w)}$  for some state  $r$ .

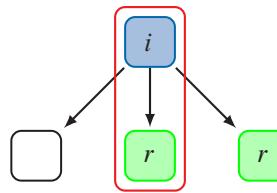
Thus, the (MAX) condition ensures that when a process  $v$  takes a transition labeled  $a!!$  then every recipient  $w$  of  $v$  that can take a local transition labeled  $a??$  does so. As a special case we get the broadcast protocols of [Esparza et al. \[1999, Section 2.1\]](#) in which “a process is always willing to receive a broadcast message.”

Figure 2.3(c) illustrates some possible configurations for a broadcast step.

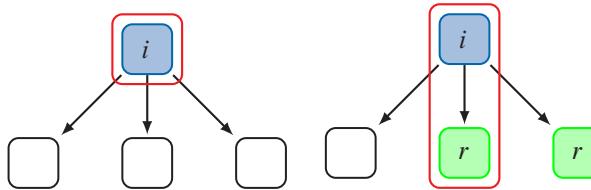
**Variations on standard primitives.** To model token passing systems, in Chapter 4 we introduce a variation of pairwise-rendezvous synchronization which keeps track of who has the token, and only allows this process to initiate a synchronization. To model synchronization failures in ad hoc networks, in Chapter 7 we introduce a *lossy* variant of broadcast synchronization by removing condition (MAX) from the definition of synchronous transition. Informally, lossy broadcast is to broadcast what asynchronous rendezvous is to pairwise rendezvous.

Figure 2.3(d) illustrates some possible configurations for a lossy broadcast step.

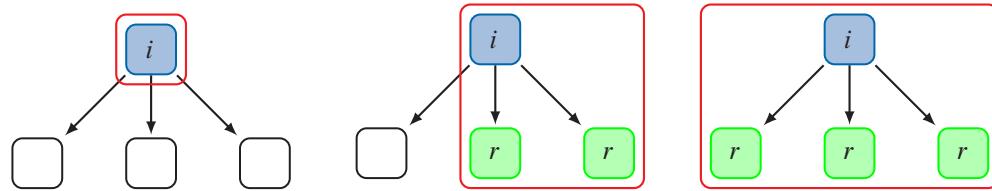
**Notation.** Note that, in general, the definition of a process template, including its transition labels, does not determine a synchronization primitive. This is highlighted in Examples 2.1 and 2.2, where the transition systems for server and client use the generic notation for actions  $in_{\text{enter}}$ ,  $in_{\text{leave}}$ ,  $out_{\text{enter}}$ ,  $out_{\text{leave}}$ , and one can consider systems that use these process templates, but differ in the synchronization primitive.



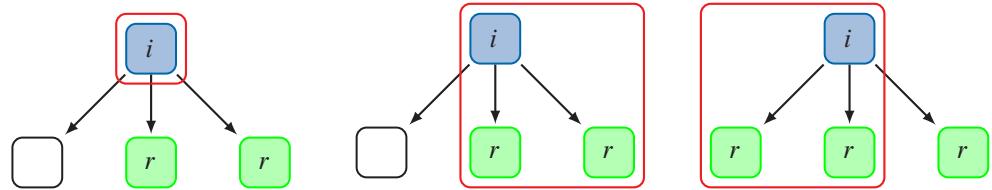
(a) **Pairwise rendezvous**: synchronization with exactly one recipient. If more than one process is ready to synchronize, only one of them can take the synchronous transition.



(b) **Asynchronous rendezvous**: synchronization with at most one recipient. If no process is ready, the initiator can take the transition on its own. Otherwise, exactly one of the recipients synchronizes.



(c) **Broadcast**: synchronization with all recipients that are ready. If no process is ready, the initiator can take the transition on its own. Otherwise, all recipients that are ready synchronize.



(d) **Lossy broadcast**: synchronization with some of the recipients that are ready. Even if other processes are ready, the initiator can also take the transition on its own.

**Figure 2.3: Synchronization primitives:** pictures show initiator of a synchronous transition, marked with  $i$ , with all its recipients. Processes that are ready to take the receive action are marked with  $r$ . The processes in the box take the synchronous transition.

## 12 2. SYSTEM MODEL AND SPECIFICATION LANGUAGES

In the following, when we use the special notation of the form  $a!, a?$  for pairwise-rendezvous actions, or  $a!!, a??$  for broadcast actions in a process template, then we implicitly fix the synchronization primitive to correspond to these action labels.

**Systems with multiple primitives  $\text{sys}(\overline{P}, G, \overline{\text{card}})$ .** By introducing a little more notation into Definition 2.2 we may define a parameterized system that allows more than one synchronization primitive. In this case, we have a tuple of synchronization constraints  $\overline{\text{card}} = (\text{card}_1, \dots, \text{card}_k)$  and an equal number of sets of synchronizing-action labels  $\Sigma_{\text{card}_i} \subset \Sigma_{\text{sync}}$ . Then define  $\text{card}_i$ -*synchronous transition* as in the definition of synchronous transition but replace  $\Sigma_{\text{sync}}$  with  $\Sigma_{\text{card}_i}$ . Now define  $\Delta$  to be the union of the internal transitions and all the  $\text{card}_i$ -synchronous transitions. Write  $\text{sys}(\overline{P}, G, \overline{\text{card}})$  for the resulting system instance.

### 2.2.2 RUNS AND DEADLOCKS

A state  $s$  of an LTS  $M$  is *deadlocked* if there is no transition with source  $s$ . Given that the synchronization primitives discussed above are non-trivial, deadlocks may occur in a system instance. The existence of deadlocks leads to two problems. First, a deadlock is usually considered as an error that we would like to detect. Second, a deadlocked run is *finite*, and the specifications we consider are usually interpreted over *infinite* runs (see Section 2.4). Therefore, we need special ways to interpret runs that end in a deadlock.

**Deadlock detection.** Deadlock detection, i.e., checking whether deadlocked states are reachable, is a problem on its own, and solutions depend heavily on the underlying synchronization primitives. We are interested in *parameterized deadlock detection*, i.e., checking if there exists an instance of a parameterized system in which a deadlocked state is reachable. For some of the systems that we consider, certain assumptions guarantee that there can never be deadlocks. This is the case for token-passing systems without direction-awareness (cf. Section 4). In a few cases, deadlocks are possible and there exist decidability results for parameterized deadlock detection. For example, some of the cutoffs for parameterized model checking of guarded protocols in Section 6 are also cutoffs for the parameterized deadlock detection problem. In most of the cases, however, there are no known results for parameterized deadlock detection.

**Interpreting deadlocked runs.** The literature we survey here has three ways to deal with deadlocked runs, i.e., finite paths (starting in initial states) which end in deadlocked states.

1. Define a run to be an infinite path starting in the initial state. Thus, we evaluate specifications over infinite paths only and ignore maximal finite paths. This is the approach taken by German and Sistla [1992].
2. Define a run to be a maximal (finite or infinite) path that starts in the initial state. This requires one to use a specification language that can also be interpreted over finite paths. This is the approach taken by Emerson and Namjoshi [1995, 2003], Baier and Katoen [2008], and Delzanno et al. [2010].

3. Change the LTS  $M$  to get an LTS  $M'$  that does not contain deadlocked states by (i) introducing a new symbol  $\tau$  into the set of action labels and (ii) for every deadlocked global state  $s$  add the transition  $(s, \tau, s)$ . Evaluate specifications over infinite paths of  $M'$  that start in an initial state. This is the approach taken by Clarke et al. [1999].

Throughout this book we will explain in every chapter how deadlocked runs are interpreted, and mention known results for the parameterized deadlock detection problem.

## 2.3 PARAMETERIZED FAMILY OF UNIFORM CONCURRENT SYSTEMS

A *parameterized d-ary connectivity graph* is a sequence  $\mathbf{G}$  of d-ary connectivity graphs. Observe that as a special case,  $\mathbf{G}(n)$  may contain  $n$  vertices and  $d = 1$ , e.g.,  $\mathbf{G}(n)$  may be the ring of size  $n$ . In general  $\mathbf{G}(n)$  need not have  $n$  vertices.

Fix a d-ary system template  $\overline{P}$ , a parameterized d-ary connectivity graph  $\mathbf{G}$ , and a synchronization constraint card. These determine a *parameterized family of uniform concurrent systems* or simply a *parameterized system*, defined as the sequence

$$n \mapsto \text{sys}(\overline{P}, \mathbf{G}(n), \text{card}).$$

**Notation**  $\overline{P}^{\mathbf{G}(n)}$ . As a shorthand we may write  $\overline{P}^{\mathbf{G}(n)}$  instead of  $\text{sys}(\overline{P}, \mathbf{G}(n), \text{card})$  if card is clear from the context.

## 2.4 PARAMETERIZED SPECIFICATIONS

Specifications express the required behavior of systems. Typical specifications for concurrent systems are that no bad *global* state is ever reached (safety specifications), and that individual processes make progress (liveness specifications). Both kinds of specifications are naturally expressed by quantifying over the processes of the system: the canonical approach is to index processes as well as atomic propositions, so that one can express, e.g., that process  $i$  is in the critical section by “ $\text{critical}_i$ ,” and further one can express that some process is in the critical section by  $\bigvee_{i \in [n]} \text{critical}_i$ . Observe that this Boolean formula is actually parameterized by  $n$ , and further that the formula—and the set of atomic propositions—grows with  $n$ . Using indexed propositions and parameterized Boolean operations in temporal logic, we obtain *indexed temporal logics*. In this book, instead of parameterized Boolean operations we will use the more intuitive notation of quantification over indices. That is,  $\bigvee_{i \in [n]} \text{critical}_i$  will be expressed as  $\exists i. \text{critical}_i$ .

Apart from being a natural formalism, the question arises whether it is *necessary* to resort to a logic that induces an unbounded number of atomic propositions to express properties of parameterized systems. To answer this question, first consider the mutual exclusion property, stating that two distinct processes are never in the critical section at the same time, i.e.,  $\mathbf{G}(\forall i, j, i \neq j. (\neg \text{critical}_i \vee \neg \text{critical}_j))$ . This formula belongs to the interesting fragment of in-

## 14 2. SYSTEM MODEL AND SPECIFICATION LANGUAGES

dexed temporal logic in which temporal operators do not appear inside the scope of index quantifiers. If a given specification is in this fragment, we can also express it in (non-indexed) temporal logic, for a modified system that can be obtained from the original one by introducing additional atomic propositions. For this example, introduce a proposition  $p$  and define a labeling function that maps a global state  $s$  to  $p$  if at most one process is in the critical section in  $s$ . (Browne et al. [1989] made this observation for the property that *exactly* one process is in the critical section.) Then, mutual exclusion is expressed as  $Gp$ , which is a formula in (non-indexed) temporal logic. Other specifications in this fragment can be found in distributed algorithms. Examples are the consensus problem [Lynch, 1996], where several processes need to agree on a common output after limited communication, or the reliable broadcast specification [Srikanth and Toueg, 1987], which ensures that all processes deliver the same set of messages.

However, one can show that even if we allow to redefine systems in such a way, non-indexed temporal logic is not sufficiently expressive for other specifications: for instance, consider the property that any process that tries to enter the critical section will eventually enter it, i.e.,  $\forall i. G(\text{trying}_i \rightarrow (\text{Fcritical}_i))$ . Note that in this case, quantified subformulas contain temporal operators. To express this specification, we need to distinguish global states according to the processes that are in specific local states (e.g., trying or critical) in order to evaluate that some process actually makes progress along a path of a computation. As the number of processes is a priori not fixed in parameterized model checking, a fixed finite set of atomic propositions is not sufficient for this purpose. We are hence forced to use a logic where the number of propositions grows with the system size, and indexed temporal logic is a very natural choice for the formalization of such specifications.

### 2.4.1 INDEXED TEMPORAL LOGICS

*Indexed temporal logics* were introduced by Browne et al. [1989] to model specifications of certain concurrent systems. They are obtained by adding *index quantifiers* to a given temporal logic over indexed atomic propositions.

For example, we can specify a mutual exclusion property as  $G(\forall i, j : i \neq j. (\neg\text{critical}_i \vee \neg\text{critical}_j))$ . Furthermore, in a uni-directional ring, we can express that if a property  $p$  holds for some process, then  $p'$  should hold for its successor in the ring by writing  $\forall i, j : j \in E(i). p_i \rightarrow p'_j$ .<sup>1</sup>

In the following, we first recall the syntax and semantics of CTL\*. Then we extend CTL\* to *indexed CTL\**. By restricting the temporal logic one gets fragments such as *indexed LTL*\x. All variations of indexed temporal logics from the PMC literature are fragments of indexed CTL\*.

#### CTL\*

We briefly describe syntax and semantics of CTL\* (see, for instance, Clarke et al. [1999], and Baier and Katoen [2008]).

<sup>1</sup>Emerson and Namjoshi [1995, 2003] write such formulas as  $\forall i. p_i \rightarrow p'_{i+1}$  where addition is modulo the size of the ring.

**Syntax.** The syntax of CTL\* over a set AP of atomic propositions is defined as follows.

*State formulas* are formed according to the grammar

$$\Phi ::= \top \mid p \mid \Phi_1 \wedge \Phi_2 \mid \neg \Phi \mid E\varphi,$$

where  $p \in AP$ ,  $\Phi_1$  and  $\Phi_2$  are state formulas and  $\varphi$  is a path formula.

*Path formulas* are formed according to the grammar

$$\varphi ::= \Phi \mid \varphi_1 \wedge \varphi_2 \mid \neg \varphi \mid X\varphi \mid \varphi_1 U \varphi_2,$$

where  $\Phi$  is a state formula and  $\varphi, \varphi_1, \varphi_2$  are path formulas. From these we derive, as usual, Boolean operators (like  $\vee$ ), temporal operators F (eventually) and G (always), and the universal path quantifier A.

**Semantics.** Formulas are interpreted in labeled transition systems  $M = (Q, Q_0, \Sigma, \delta, \lambda)$  over atomic propositions AP. For a state  $s$  of  $M$ , define

- $M, s \models p$  iff  $p \in \lambda(s)$ ,
- $M, s \models \Phi_1 \wedge \Phi_2$  iff  $M, s \models \Phi_1$  and  $M, s \models \Phi_2$ ,
- $M, s \models \neg \Phi$  iff not  $M, s \models \Phi$ , and
- $M, s \models E\varphi$  iff  $M, \pi \models \varphi$  for some infinite path  $\pi$  of  $M$  starting in  $s$ .

For an infinite path  $\pi$  of  $M$  starting in  $s$ , write  $\pi^j$  for suffix of  $\pi$  starting at position  $j \geq 0$ , and define

- $M, \pi \models \Phi$  iff  $M, s \models \Phi$ ,
- $M, \pi \models \varphi_1 \wedge \varphi_2$  iff  $M, \pi \models \varphi_1$  and  $M, \pi \models \varphi_2$ ,
- $M, \pi \models \neg \varphi$  iff not  $M, \pi \models \varphi$ ,
- $M, \pi \models X\varphi$  iff  $M, \pi^1 \models \varphi$ , and
- $M, \pi \models \varphi_1 U \varphi_2$  iff  $\exists j \geq 0 \forall k < j. (M, \pi^j \models \varphi_2 \text{ and } M, \pi^k \models \varphi_1)$ .

Finally, define  $M \models \Phi$  if for all initial states  $s$  of  $M$ , it holds that  $M, s \models \Phi$ .

Note that action labels  $\Sigma$  are not used in the semantics.

**Fragments.** An important fragment of CTL\* is LTL, which consists of all CTL\* formulas that start with a universal path quantifier A, followed by a CTL\* formula without path quantifiers E or A. If it is clear that a formula is in LTL, then the leading path quantifier A may be omitted. From CTL\* and LTL, we obtain fragments CTL\* $\setminus X$  and LTL $\setminus X$ , respectively, by only considering formulas that do not contain the next-state operator X.

## 16 2. SYSTEM MODEL AND SPECIFICATION LANGUAGES

### Indexed CTL\*

Fix an infinite set  $I = \{i, j, \dots\}$  of index variables.

**Syntax.** The syntax of indexed CTL\* over a set AP of atomic propositions and set I of index variables is defined as follows.

*State formulas* are formed according to the grammar

$$\Phi ::= \top \mid p_i \mid \forall i : r. \Phi \mid \exists i : r. \Phi \mid \Phi_1 \wedge \Phi_2 \mid \neg \Phi \mid E\varphi,$$

where  $p \in AP$ ,  $i \in I$ ,  $\Phi$ ,  $\Phi_1$ ,  $\Phi_2$  are state formulas,  $\varphi$  is a path formula, and  $r$  is a list of *range expressions* from

$$\text{type}(i) = t \mid neq(i_1, \dots, i_n) \mid i \in E(j),$$

where  $t \in \mathbb{N}$  and  $i, i_1, \dots, i_n, j \in I$ .

*Path formulas* are formed according to the grammar

$$\varphi ::= \Phi \mid \forall i : r. \varphi \mid \exists i : r. \varphi \mid \varphi_1 \wedge \varphi_2 \mid \neg \varphi \mid X\varphi \mid \varphi_1 \cup \varphi_2$$

where  $\Phi$  is a state formula and  $\varphi, \varphi_1, \varphi_2$  are path formulas.

Our notation is inspired by that of Emerson and Namjoshi [1995, 2003]. The symbols  $\forall$  and  $\exists$  are called *index quantifiers*. If  $r$  is empty, we just write  $\forall i. \phi$  or  $\exists i. \phi$ . The expression  $neq(i, j)$  may also be written  $i \neq j$ . Variable  $i$  and atoms  $p_i$  in an indexed CTL\* formula are *bound* if they are in the scope of a quantifier  $\forall i : r$  or  $\exists i : r$ . An indexed CTL\* formula  $\Phi$  is a *sentence* if every atom in it is bound. An indexed CTL\* formula  $\phi$  in which index variables  $i_1, \dots, i_k$  are not bound by index quantifiers may be written  $\phi(i_1, \dots, i_k)$ . We say that  $i_1, \dots, i_k$  are *free* in  $\phi$ .

**Semantics.** An indexed CTL\* formula  $\phi$  is interpreted over a system instance  $\overline{P}^G$  (with  $\overline{P}$  a system template and  $G = (V, E, \text{type})$  a connectivity graph) and a *valuation of the index variables*  $e : I \rightarrow V$ .

A *valuation e* satisfies the range expression  $\text{type}(i) = t$  if  $\text{type}(e(i)) = t$ . Similarly, it satisfies  $neq(i_1, \dots, i_n)$  if  $e(i_1), \dots, e(i_n)$  are pairwise distinct, and it satisfies  $i \in E(j)$  if  $(e(i), e(j)) \in E$ . Finally, it satisfies a list of range expressions if it satisfies every element of the list. A valuation  $e : I \rightarrow V$  is an *i-variant* of a valuation  $e'$  if  $e'(j) = e(j)$  for all  $j \in I \setminus \{i\}$ .

Define  $\overline{P}^G, e, s \models \Phi$  inductively:

- $\overline{P}^G, e, s \models p_i$  iff  $p_{e(i)} \in \Lambda(s)$ ,
- $\overline{P}^G, e, s \models \forall i : r. \Phi$  iff for all *i*-variants  $e'$  of  $e$  that satisfy  $r$  it holds that  $\overline{P}^G, e', s \models \Phi$ ,

- $\overline{P}^G, e, s \models \exists i : r. \Phi$  iff there exists an  $i$ -variant  $e'$  of  $e$  that satisfies  $r$  and it holds that  $\overline{P}^G, e', s \models \Phi$ ,
- $\overline{P}^G, e, s \models \Phi_1 \wedge \Phi_2$  iff  $\overline{P}^G, e, s \models \Phi_1$  and  $\overline{P}^G, e, s \models \Phi_2$ ,
- $\overline{P}^G, e, s \models \neg\Phi$  iff not  $\overline{P}^G, e, s \models \Phi$ , and
- $\overline{P}^G, e, s \models \text{E}\varphi$  iff  $\overline{P}^G, e, \pi \models \varphi$  for some infinite path  $\pi$  of  $\overline{P}^G$  starting in  $s$ .

For an infinite path  $\pi$  of  $\overline{P}^G$  starting in  $s$ , define

- $\overline{P}^G, e, \pi \models \Phi$  iff  $\overline{P}^G, e, s \models \Phi$ ,
- $\overline{P}^G, e, \pi \models \forall i : r. \varphi$  iff for all  $i$ -variants  $e'$  of  $e$  that satisfy  $r$  it holds that  $\overline{P}^G, e', \pi \models \varphi$ ,
- $\overline{P}^G, e, \pi \models \exists i : r. \varphi$  iff there exists an  $i$ -variant  $e'$  of  $e$  that satisfies  $r$  and it holds that  $\overline{P}^G, e', \pi \models \varphi$ ,
- $\overline{P}^G, e, \pi \models \varphi_1 \wedge \varphi_2$  iff  $\overline{P}^G, e, \pi \models \varphi_1$  and  $\overline{P}^G, e, \pi \models \varphi_2$ ,
- $\overline{P}^G, e, \pi \models \neg\varphi$  iff not  $\overline{P}^G, e, \pi \models \varphi$ ,
- $\overline{P}^G, e, \pi \models \text{X}\varphi$  iff  $\overline{P}^G, e, \pi^1 \models \varphi$ , and
- $\overline{P}^G, e, \pi \models \varphi_1 \cup \varphi_2$  iff  $\exists j \geq 0 \ \forall k < j. (\overline{P}^G, e, \pi^j \models \varphi_2 \text{ and } \overline{P}^G, e, \pi^k \models \varphi_1)$ .

If  $\Phi$  is a sentence, define  $\overline{P}^G, s \models \Phi$  if for all evaluations (equivalently, for some evaluation)  $e : I \rightarrow V$  it holds that  $\overline{P}^G, e, s \models \Phi$ . Finally we define:

$$\overline{P}^G \models \Phi$$

if for every initial state  $s$  of  $\overline{P}^G$  it holds that  $\overline{P}^G, s \models \Phi$ .

**Notation.** We write  $P^G, s \models \Phi(c_1, \dots, c_k)$  if for all evaluations  $e$  in which  $i_j$  maps to  $c_j$  (for  $j \leq k$ ) it holds that  $P^G, e, s \models \Phi(i_1, \dots, i_k)$ .

**Fragments.** A *prenex* formula is of the form  $Q_1 i_1, \dots, Q_k i_k : r. \phi(i_1, \dots, i_k)$ , where the  $Q_j$  are index quantifiers and  $\phi(i_1, \dots, i_k)$  has no index quantifiers. Formulas of the form  $Q_1 i_1, \dots, Q_k i_k : r. \phi(i_1, \dots, i_k)$ , for some range expression  $r$ , will be referred to as *k-indexed*.<sup>2</sup>

<sup>2</sup>Note that in these formulas only the last quantifier has a range expression. This is not an additional restriction, however, since semantically  $Q_1 i_1 : r_1 \dots Q_k i_k : r_k. \phi(i_1, \dots, i_k)$  with separate range expressions  $r_1, \dots, r_k$  is equivalent to  $Q_1 i_1, \dots, Q_k i_k : r_1 \wedge \dots \wedge r_k. \phi(i_1, \dots, i_k)$  with the conjunction of all range expressions.

## 18 2. SYSTEM MODEL AND SPECIFICATION LANGUAGES

Since the results in the literature concentrate on the prenex fragment of indexed-temporal logic, we introduce some shorthands for important fragments. Write  $\text{ICTL}^*$  for the set of all prenex indexed  $\text{CTL}^*$  sentences, and  $k\text{-CTL}^*$  for the set of all  $k$ -indexed formulas in  $\text{ICTL}^*$ .

Prenex indexed LTL, written  $\text{ILTL}$ , is defined as the linear-time fragment of  $\text{ICTL}^*$ , i.e., formulas of the form  $Q_1 i_1, \dots, Q_k i_k : r. A\phi(i_1, \dots, i_k)$ , where the  $Q_j$  are index quantifiers and  $\phi(i_1, \dots, i_k)$  has no index quantifiers and no path quantifiers.<sup>3</sup> Write  $k\text{-LTL}$  for the set of all  $k$ -indexed formulas in  $\text{ILTL}$ .

Fragments of  $\text{ICTL}^*$  without the next operator, such as  $\text{ICTL}^* \setminus X$  and  $\text{ILTL} \setminus X$ , are obtained by disallowing formulas that contain the  $X$  operator. Specifications without the next-time operator  $X$  are common in the literature, since they are stuttering-insensitive, and thus are a natural specification language for asynchronous concurrent systems. Moreover, the presence of  $X$ , even with weak communication primitives, leads to undecidability of the PMCP [Emerson and Kahlon, 2003b].

### 2.4.2 ACTION-BASED SPECIFICATIONS

In contrast to specifications in index temporal logic, which are formulas over indexed state labels, an action-based specification evaluates the sequences of action labels of  $\overline{P}^G$ . An  $\omega$ -regular action-based specification is an  $\omega$ -regular language  $\mathcal{L}$  over actions  $\Sigma_{int} \cup \Sigma_{sync}$ .

It is interpreted as follows: write  $\overline{P}^G \models \mathcal{L}$  if the projection onto  $\Sigma_{int} \cup \Sigma_{sync}$  of every infinite path of  $\overline{P}^G$  starting in an initial state is in the language  $\mathcal{L}$ .

We also find regular action-based specifications which are (ordinary finite-word) regular languages  $\mathcal{L}$  over actions  $\Sigma_{int} \cup \Sigma_{sync}$ . These are interpreted as follows:  $\overline{P}^G \models \mathcal{L}$  if every finite prefix of every projection onto  $\Sigma_{int} \cup \Sigma_{sync}$  of every path of  $\overline{P}^G$  starting in an initial state is in the language  $\mathcal{L}$ .

We write  $\text{Reg}(A)$  and  $\omega\text{Reg}(A)$  for regular and  $\omega$ -regular action-based specifications, respectively.

### 2.4.3 SPECIFICATIONS IN THE LITERATURE

Most of the decidability results in the literature are based on one of the following forms of specifications.

- Plain temporal logics (or  $\omega$ -regular properties) over state labels of a specific process template [Emerson and Kahlon, 2003b, Emerson and Namjoshi, 1996, 1998, German and Sistla, 1992]. These are used for systems with a special *control* process and an unbounded number of *user* processes, and specify only the behavior of the controller.

<sup>3</sup>Note that the index variables are bound *outside* of the temporal path quantifier. In particular, for an existentially quantified formula to be satisfied there must exist a valuation of the variables such that  $\phi$  holds for all paths (and not one valuation for each path).

- Indexed temporal logics over the state labels of all processes [Clarke et al., 2004, Emerson and Kahlon, 2000, 2003a,c, 2004, Emerson and Namjoshi, 1995, 2003, German and Sistla, 1992]. These are used for systems with or without a unique control process.
- Action-based specifications on global transitions [Emerson and Kahlon, 2003b, Emerson and Namjoshi, 1998, Esparza et al., 1999]. Specifications on the actions of the system do not directly talk about the state of local processes. However, as argued by Esparza et al. [1999], in many cases one can construct a modified system that satisfies an adapted specification over state labels if and only if the original system satisfies the original action-based specification.

In addition, we consider special parameterized model checking problems COVER (reachability of a local state in one of the components), REPEAT (repeated reachability of a local state in the same component), and TARGET (reachability of a certain state in all components simultaneously, not definable in prenex  $\text{CTL}^*$ ) [Abdulla et al., 2013a, Delzanno et al., 2010, 2011, 2012b]. These are used when reasoning about ad-hoc networks with an arbitrary (and possibly changing) graph structure, and constitute small and important fragments that may be decidable even for systems where more general languages are undecidable (see Chapter 7).

## 2.5 MODEL CHECKING PROBLEMS FOR CONCURRENT SYSTEMS

Typical questions from the PMC literature are of the form: Is the PMCP decidable for a given class of systems (defined by system template, connectivity graph, and synchronization constraint) and a given class of specifications? We formalize such questions in the following. Fix:

- a set of system templates  $\mathcal{P}$  (such as the  $d$ -tuples of finite process templates),
- a parameterized connectivity graph  $\mathbf{G}$  (such as the sequence of rings),
- a tuple of synchronization constraints  $\overline{\text{card}}$  (which is often a singleton card), and
- a set of indexed-temporal logic or action-based specifications  $\mathcal{F}$  (such as  $k\text{-CTL}^*$ ).

First, we define the (non-parameterized) model checking problem for our class of systems:

*The model-checking problem  $\text{MCP}(\mathcal{P}, \mathbf{G}, \mathcal{F}, \overline{\text{card}})$  is as follows:*

**Input.**  $\overline{P} \in \mathcal{P}$ ,  $\phi \in \mathcal{F}$  and  $n \in \mathbb{N}$ .

**Output.** “Yes” if  $\text{sys}(\overline{P}, \mathbf{G}(n), \overline{\text{card}}) \models \phi$ ; and “No” otherwise.

## 20 2. SYSTEM MODEL AND SPECIFICATION LANGUAGES

Unless explicitly stated otherwise, in this book we assume that all system instances  $\text{sys}(\overline{P}, \mathbf{G}(n), \text{card})$  are finite-state, and thus that the (non-parameterized) model checking problem is decidable.

In contrast to this, in the parameterized model checking problem we do not give  $n$  as an input, but ask whether the specification holds for *all*  $n$ :

*The parameterized model-checking problem PMCP( $\mathcal{P}, \mathbf{G}, \mathcal{F}, \overline{\text{card}}$ ) is as follows:*

**Input.**  $\overline{P} \in \mathcal{P}$ , and  $\phi \in \mathcal{F}$ .

**Output.** “Yes” if  $\text{sys}(\overline{P}, \mathbf{G}(n), \overline{\text{card}}) \models \phi$  for all  $n \in \mathbb{N}$ ; and “No” otherwise.

When  $\overline{\text{card}}$  is clear from the context, we write  $\text{PMCP}(\mathcal{P}, \mathbf{G}, \mathcal{F})$ .

**Example 2.3** Consider the question: Is the PMCP decidable for token-passing systems on uni-directional rings, with specifications in  $\text{ILTL} \setminus X$ ? This problem is considered in Section 4.2.1 and written  $\text{PMCP}(\mathcal{P}_{\text{simpTok}}, \mathbf{R}, \text{ILTL} \setminus X, \{1\})$  where  $\mathcal{P}_{\text{simpTok}}$  is a set of processes that are designed to simulate token-passing using pairwise rendezvous, i.e.,  $\text{card} = \{1\}$ , and  $\mathbf{R}(n)$  is the uni-directional ring of size  $n$ .

### 2.5.1 COMPUTABILITY ASSUMPTIONS

In parameterized model checking one usually makes the following very general assumptions. In the literature they are typically not made explicit.

- Each process template  $P$  is computable, meaning there is an algorithm that computes exactly the states and transitions of  $P$ .
- Each set  $\text{card} \subseteq \mathbb{N}_0$  is computable.
- The parameterized connectivity graph is computable, meaning that there is an algorithm that on input  $n \in \mathbb{N}$  outputs a list of the vertices and edges in  $\mathbf{G}(n)$ .

Note that for the systems under consideration in this book, the first two are satisfied by definition, since  $P$  is finite-state, and  $\text{card}$  is finite or co-finite for the standard communication primitives. Furthermore, note that the first assumption is also satisfied by various classes of infinite-state processes that are considered in the parameterized verification literature, such as pushdown machines or processes with variables that range over the natural numbers. From these assumptions we can conclude that:

- (i) the inputs to the PMCP are finite objects, e.g.,  $\overline{P}$  may be finite state, or (the configuration space of) a counter machine; and

- (ii) the parameterized system is computable, meaning that there is an algorithm that given  $n$  returns a finitary representation of the system instance  $\text{sys}(\overline{P}, \mathbf{G}(n), \text{card})$ , e.g., a list of the states and transitions, or in case the system instance is infinite it may be an algorithm that computes the states and transitions.

The first item is needed for the PMCP to be an algorithmic problem, and the second item is needed if there is to be any hope of deciding the PMCP (for particular cases).



## CHAPTER 3

# Standard Proof Machinery

The purpose of this section is to briefly summarize the typical principles and ingredients in proofs of (un)decidability for the PMC problem.

### 3.1 TECHNIQUES TO PROVE UNDECIDABILITY OF PMCP

To prove that  $\text{PMCP}(\mathcal{P}, \mathbf{G}, \mathcal{F})$  is undecidable it is sufficient to computably transform a given Turing Machine  $T$  into a tuple  $(\overline{P}, \mathbf{G}, \phi) \in \mathcal{P} \times \mathcal{G} \times \mathcal{F}$  such that  $T$  does not halt on empty input if and only if for all  $n \in \mathbb{N}$ ,  $\overline{P}^{\mathbf{G}(n)} \models \phi$ .

The first undecidability proof that applies to uniform parameterized systems is by [Suzuki \[1988\]](#), for systems consisting of identical processes arranged in a uni-directional ring with a single multi-valued token. The proof reduces non-halting of Turing machines to this problem. A neater exposition by [Emerson and Namjoshi \[1995\]](#) goes via two-counter machines. Informally, a two-counter machine [[Minsky, 1967](#)] has a read-only tape, finite control, and two counters, with operations to increment a counter, decrement a counter, test a counter for zero, and change its internal state. It accepts the word on its input tape if there is a run starting with zero in both of its counters that leads to a halting state. Two-counter machines are Turing-complete. Obviously,  $k$ -counter machines for  $k > 2$  are also Turing-complete.

We use a formal definition of counter machine close to the one given by [Emerson and Kahlon \[2003b\]](#). Although it may look more complicated than the usual Minsky machines, we picked their definition for the purpose of presentation, as it typically makes PMCP undecidability proofs easier.

**Counter Machines.** An *input-free  $k$ -counter machine* ( $k$ CM)  $\mathcal{M}$  consists of a set of locations  $[m]$  (for some  $m \in \mathbb{N}$ ), and action set  $\mathcal{A} = \{inc(c_q), dec(c_q), zero(c_q): q \in [k]\}$ , and a set  $\Delta \subseteq [m] \times \mathcal{A} \times [m]$  of *commands*. A *configuration* of  $\mathcal{M}$  is a tuple  $(i, \bar{v}) \in [m] \times \mathbb{N}_0^k$ ; so  $i$  is a location and  $v_j$  indicates the value of the  $j$ -th counter. The *initial configuration* is  $(1, 0, \dots, 0)$ ; so the machine starts in location 1 with empty counters). The *halting configurations* are  $\{m\} \times \mathbb{N}_0^k$ ; so the machine halts whenever it reaches location  $m$ ). The *configuration graph* of  $\mathcal{M}$  is the directed graph whose vertices are all the configurations, and edges are defined as follows. There is an edge from configuration  $(i, \bar{v})$  to configuration  $(j, \bar{w})$  if (a)  $i \neq m$  and (b) there exists a command  $(i, \sigma, j) \in \Delta$  such that:

- if  $\sigma$  is  $inc(c_q)$  then  $w_q = v_q + 1$  and  $w_p = v_p$  for  $p \neq q$ ;

### 24 3. STANDARD PROOF MACHINERY

- if  $\sigma$  is  $dec(c_q)$  then  $w_q = v_q - 1$  and  $w_p = v_p$  for  $p \neq q$ ; and
- if  $\sigma$  is  $zero(c_q)$  then  $w_q = v_q = 0$  and  $w_p = v_p$  for  $p \neq q$ .

Note that if  $\sigma$  is  $dec(c_q)$  then the described edge exists only if  $v_q > 0$  (i.e., there is an implicit test for non-zero).

This definition differs from Minsky's original definition of multi-counter machine in two ways: (i) a Minsky machine is deterministic while this definition is non-deterministic and (ii) a Minsky machine has no deadlocked configurations while this definition allows non-halting configurations with no outgoing edge. The translation of a Minsky machine into our definition results in a machine such that for every location  $l \in [m]$  either there is exactly one outgoing transition, and it is labeled  $inc(c_q)$  for some  $q$ , or there are exactly two outgoing transitions and they are labeled  $dec(c_q)$  and  $zero(c_q)$  for some  $q$ . From this translation and undecidability of the halting problem for Minsky machines [Minsky, 1967], the following theorem is immediate.

**Theorem 3.1** *The following problem is undecidable: given an input-free  $k$ -counter machine  $\mathcal{M}$  (for  $k \geq 2$ ), where the counters are initialized with zeroes, does there exist a path in the configuration space of  $\mathcal{M}$  from the initial configuration to a halting configuration?*

**Undecidability of PMCP.** Thus, typical undecidability proofs in this area show how to simulate a 2CM by the composition of (uniform) finite-state processes. Typically one process, the *controller* process, simulates the internal state of the 2CM, while the remaining  $n$  *storage* processes collectively encode the counter values (the simulation either covers  $n$  steps of the 2CM, or a prefix of the computation as long as the counter values are bounded by  $n$ ). The work in the proof is typically showing how the controller can issue commands to increment/decrement counters and test counters for zero. The prototypical such proof, following Emerson and Namjoshi [1995, 2003], is sketched in Section 4.2.3.

## 3.2 HOW TO PROVE DECIDABILITY OF THE PMCP

In this section we describe techniques that help one prove that the PMCP is decidable, i.e., symmetry arguments, reductions to well-structured transition systems or vector addition systems, and cutoff techniques.

A natural first step of a decidability proof is to use symmetry of the systems  $\overline{P}^G$  as well as the specifications to simplify the problem. For instance:

- if the connectivity graph  $G$  is a clique, and all processes are instances of the same process template  $P$ , then  $P^G \models \forall i. \phi(i)$  is equivalent to  $P^G \models \phi(1)$ , or
- as noted by Emerson and Namjoshi [1995, 2003], if the connectivity graph  $G$  is a ring, and all processes are instances of the same process template  $P$ , then  $P^G \models \forall i, j : i \neq j. \phi(i, j)$  is equivalent to  $P^G \models \forall j. \phi(1, j)$ .

Such intuitions can be formalized following the approach by [Emerson and Sistla \[1996\]](#). Formal treatment of symmetry is tedious and thus often only addressed implicitly in the literature (and this survey). Note that [Emerson and Namjoshi \[1995, 2003\]](#) treated symmetry explicitly.

We can reduce the PMCP to model checking a single infinite-state LTSs by combining  $\overline{P}^G$  for  $G \in \mathbf{G}$ . Model checking is decidable for certain classes of infinite-state systems and certain specifications. Notably, the coverability problem (defined below) is decidable for well-structured transition systems (WSTS); see [Finkel and Schnoebelen \[2001\]](#) or [Abdulla et al. \[1996\]](#). Vector addition systems with states (or equivalently Petri Nets) are special kinds of WSTSs for which repeated coverability and reachability problems are decidable. For classical results in Petri nets, see [Esparza \[1998\]](#) and [Yen \[1992\]](#). Thus, for example, if the combined system is a WSTS or a VASS, then we can decide the PMCP. We review the definitions and decidability results for WSTS and VASS in Section 3.2.1.

Another approach is to reduce PMCP to model checking finitely many system instances, see e.g., [Emerson and Namjoshi \[1995, 2003\]](#) and [Clarke et al. \[2004\]](#). Such results are often described as cutoffs and are typically proved by exhibiting suitable simulation relations between almost all system instances of the parameterized system and a fixed system instance. We formalize this in Section 3.2.3.

### 3.2.1 WELL-STRUCTURED TRANSITION SYSTEMS

Following the work by [Abdulla et al. \[1996\]](#) and [Finkel and Schnoebelen \[2001\]](#), well-structured transition systems became a popular framework for reasoning about infinite-state systems and parameterized systems. We begin by recalling some required notions.

A binary relation  $\leq$  on a set  $X$  is a *quasi-order* if it is reflexive and transitive. For  $T \subseteq X$  write  $\uparrow T := \{x \in X \mid \exists t \in T, t \leq x\}$ , called the *upward closure* of  $T$ . The *downward closure* of  $T$  is the set  $\{x \in X \mid \exists t \in T, x \leq t\}$ . A set  $T$  is *upward closed* if  $T = \uparrow T$ .

A quasi-order is a *well quasi-order* if for every infinite sequence  $x_1, x_2, \dots$  there exist indices  $i < j$  with  $x_i \leq x_j$ . In particular, there are no infinite decreasing chains and no infinite anti-chains.

A typical example of a well quasi-order on  $\mathbb{N}^n$  (for  $n \in \mathbb{N}$ ) is defined by  $(x_1, \dots, x_n) \leq (y_1, \dots, y_n)$  if for every  $i \leq n$  it holds that  $x_i \leq y_i$ .

The following properties follow from the definition of well quasi-order  $\leq$ . If  $C$  is upward closed then there is a finite set  $B \subseteq C$ , called a *basis of  $C$* , such that  $C$  is the upward closure of  $B$ :  $C = \uparrow B$  (the word *basis* is used to mean that  $B$  generates  $C$ , and not that  $B$  is unique with this property). If  $C_1 \subseteq C_2 \subseteq C_3 \subseteq \dots$  is an infinite chain of upward closed sets then there exists  $k$  such that  $C_k = \cup_{n \in \mathbb{N}} C_n$ .

Let  $\leq$  be a quasi-order on the set of states of a transition system  $M$ . For states  $s, t$  of  $M$ , write  $s \xrightarrow{a} t$  if there is an edge in  $M$  from  $s$  to  $t$  labeled  $a$ , write  $s \rightarrow t$  if  $s \xrightarrow{a} t$  for some  $a$ , and write  $s \xrightarrow{*} t$  if there is a path in  $M$  from  $s$  to  $t$ . For a set  $C$  of states of  $M$ , define the set  $Pred(C)$  of states  $M$  as follows:  $s \in Pred(C)$  iff either  $s \in C$  or there exists  $t \in C$  such that

### 26 3. STANDARD PROOF MACHINERY

$s \rightarrow t$ . Say that  $\leq$  is *monotonic* with respect to  $M$  if for all states  $s, s', t$  of  $M$  and all action labels  $a$ , if  $s \leq s'$  and  $s \xrightarrow{a} t$ , then there exists  $t' \geq t$  such that  $s' \xrightarrow{a} t'$ .<sup>1</sup> The following property holds if  $\leq$  is monotonic: if  $C$  is an upward-closed set of states of  $M$ , then  $\text{Pred}(C)$  is upward closed.

**Definition 3.2** Let  $M$  be a transition system and let  $\leq$  be an ordering on the state set of  $M$ . Then  $M$  is a *well-structured transition system* (WSTS) with respect to  $\leq$  if the following conditions hold:

1.  $\leq$  is a well-quasi ordering,
2.  $\leq$  is monotonic with respect to  $M$ , and
3. one can compute, from a basis for an upward closed set  $C$ , a basis for  $\text{Pred}(C)$ .

Finkel and Schnoebelen [2001] gave a detailed discussion of fundamental computation models that are well-structured transition systems. Examples are variants of, e.g., communication finite state machines, basic process algebra, and Petri nets.

The following problem is parameterized by an infinite-state LTS  $M$  with initial states  $I$  and a quasi-ordering  $\leq$  on the states  $S$  of  $M$ .

- The *coverability* problem:  
**input:** finite set of states  $T \subset S$ .  
**output:** ‘Yes’ if and only if there exist  $i \in I$  and  $t \in \uparrow T$  such that  $i \xrightarrow{*} t$ .

The following theorem can be easily obtained from the work by Abdulla et al. [1996] and Finkel and Schnoebelen [2001].

**Theorem 3.3** *The coverability problem is decidable for WSTSs, if the initial state set  $I$  satisfies one of the following conditions.*

1.  $I$  is finite.
2.  $I$  is downward closed, and has a computable set membership.

**Proof.** The proof idea is similar in both cases. From a basis for  $C$  and  $n$ , one can compute a basis for  $\text{Pred}^n(C)$ . Thus, one can compute, given  $n$ , whether  $\text{Pred}^n(C) = \text{Pred}^{n+1}(C)$ : they are equal iff for every  $s'$  in a basis of  $\text{Pred}^{n+1}(C)$  there is  $s$  in a basis of  $\text{Pred}^n(C)$  such that  $s \leq s'$ . Thus, one can compute a number  $k$  such that  $\text{Pred}^k(C) = \bigcup_{n \in \mathbb{N}} \text{Pred}^n(C)$ , and hence a basis for  $\bigcup_{n \in \mathbb{N}} \text{Pred}^n(C)$ .

Now consider the coverability problem for state set  $T$  and initial state set  $I$ . Let  $B$  be a basis for  $\bigcup_{n \in \mathbb{N}} \text{Pred}^n(\uparrow T)$ . There exists  $i \in I$  and  $t \in \uparrow T$  such that  $i \xrightarrow{*} t$  if and only if  $I \cap \uparrow B \neq \emptyset$ .

<sup>1</sup>Monotonicity is called “strong compatibility” by Finkel and Schnoebelen [2001].

In other words, we have reduced the coverability problem to checking if there is some  $i \in I$  and some  $b \in B$  such that  $b \leq i$ . If  $I$  is finite then there are only finitely many pairs to check. If  $I$  is downward closed then  $I \cap \uparrow B \neq \emptyset$  is equivalent to  $I \cap B \neq \emptyset$ . If  $I$  also has decidable membership then one simply needs to check if one of the finitely many elements of  $B$  is in  $I$ .  $\square$

This theorem is used to prove decidability of PMCP for safety properties in Theorem 5.7, Theorem 6.31, and Theorem 7.19.

### 3.2.2 VECTOR ADDITION SYSTEMS WITH STATES (AND PETRI NETS)

We define a *vector addition system with states* (VASS)  $M$  to consist of a finite set  $Q$  of states, an initial state  $\iota \in Q$ , a finite action set  $A \subset \mathbb{Z}^d$ , and finite transition relation  $T \subseteq Q \times A \times Q$ . The *configuration space* of a VASS  $M$  is the transition system with state set  $Q \times \mathbb{N}_0^d$ , initial state  $(\iota, 0, \dots, 0)$ , and transitions defined by  $(q, \bar{x}) \xrightarrow{a} (r, \bar{y})$  if  $\bar{y} = \bar{x} + a$  where  $(q, a, r) \in T$ . Write  $s \xrightarrow{*} t$  if there is a path in the configuration space (ignoring actions) from  $s$  to  $t$ .

The VASS with ordering  $\leq$  defined as follows is a WSTS:  $(q, \bar{x}) \leq (r, \bar{y})$  if  $q = r$  and  $x_i \leq y_i$  for  $1 \leq i \leq d$ . Besides the coverability problem, we also consider the following.

- The *reachability* problem:  
**input:** configuration  $t = (q, \bar{v}) \in Q \times \mathbb{N}_0^d$ .  
**output:** ‘Yes’ if and only if  $\iota \xrightarrow{*} t$  for some  $\iota \in I$ .
- The *control-state repeated coverability* problem:  
**input:**  $q \in Q$ .  
**output:** ‘Yes’ if and only if there exist infinitely many configurations  $t_1, t_2, \dots$  each of whose first coordinate is  $q$  and such that  $(\iota, 0, \dots, 0) \xrightarrow{*} t_1$  and  $t_i \xrightarrow{*} t_{i+1}$  for all  $i \in \mathbb{N}$ .

The following theorem summarizes results on Petri nets and VASS important for the exposition in this book, cf. [Esparza, 1998] for a detailed survey of this area.

**Theorem 3.4** *The control state repeated coverability problem for VASS is EXPSPACE-complete. Moreover, the problem can be solved in space which is polynomial in the number of states of the VASS and exponential in the dimension of the VASS. The reachability problem is decidable (and EXPSPACE-hard).*

This theorem is used to prove decidability of special PMCPs of broadcast and pairwise systems for liveness properties in Theorem 5.8, Theorem 5.10, and Theorem 7.21.

A VASS in which  $|Q| = 1$  is called a vector-addition system (VAS), which is a notational variant of Petri nets. Conversely, every VASS can be represented as a VAS (since the state component can be coded in  $|Q|$  many additional coordinates).

## 28 3. STANDARD PROOF MACHINERY

### 3.2.3 DECOMPOSITIONS AND CUTOFFS

Some proofs that  $\text{PMCP}(\mathcal{P}, \mathbf{G}, \mathcal{F})$  is decidable also yield a computable reduction to a finite set of finite-state model checking problems. This is formalized as follows.

A *decomposition* for  $(\mathcal{P}, \mathbf{G}, \mathcal{F})$  is a function mapping every pair  $(\bar{P}, \phi) \in \mathcal{P} \times \mathcal{F}$  (where  $\bar{P}$  and  $\mathbf{G}$  have the same arity  $d$ ) to a pair  $(\text{MC}_{fin}, \Phi_{\mathbb{B}})$ , where

1.  $\text{MC}_{fin}$  is a set  $\{(K_1, \phi_1), \dots, (K_m, \phi_m)\}$ , where each  $K_i$  is a  $d$ -ary connectivity-graph, each  $\phi_i$  is a formula,  $m$  is a positive integer, and
  2.  $\Phi_{\mathbb{B}}$  is a Boolean formula  $\Phi_{\mathbb{B}}(x_1, \dots, x_m)$  (with  $m$  variables)
- such that the following are equivalent:

- $\forall n \in \mathbb{N}, \bar{P}^{\mathbf{G}(n)} \models \phi$ ,
- $\Phi_{\mathbb{B}}(c_1, \dots, c_m)$  evaluates to  $\top$  under the assignment  $c_i = \top$  iff  $\bar{P}^{K_i} \models \phi_i$ .

Thus, a decomposition reduces the problem  $\text{PMCP}(\mathcal{P}, \mathbf{G}, \mathcal{F})$  to a Boolean combination of the finitely many model checking problems from the set  $\text{MC}_{fin}$ . Note that if the decomposition is a computable function, then  $\text{PMCP}(\mathcal{P}, \mathbf{G}, \mathcal{F})$  is decidable.

We identify a special case of this definition that appears in the literature: we say  $m$  is a *cutoff* (for  $(\mathcal{P}, \mathbf{G}, \mathcal{F})$ ) if for all  $(\bar{P}, \phi) \in \mathcal{P} \times \mathcal{F}$ , the image of the decomposition function is the pair  $(\text{MC}_{fin}, \Phi_{\mathbb{B}})$  where  $\text{MC}_{fin} = \{(\mathbf{G}(1), \phi), \dots, (\mathbf{G}(m), \phi)\}$  and  $\Phi_{\mathbb{B}} = x_1 \wedge \dots \wedge x_m$ . In other words,  $m$  is a cutoff means that for all  $(\bar{P}, \phi) \in \mathcal{P} \times \mathcal{F}$ :  $\forall n, \bar{P}^{\mathbf{G}(n)} \models \phi$  if and only if  $\forall n \leq m, \bar{P}^{\mathbf{G}(n)} \models \phi$ . Cutoff results appear in Section 4 and Section 6.

**Proposition 3.5** *If  $(\mathcal{P}, \mathbf{G}, \mathcal{F})$  has a cutoff, then  $\text{PMCP}(\mathcal{P}, \mathbf{G}, \mathcal{F})$  is decidable.*

*Proof.* Suppose  $m$  is a cutoff. This means that for all  $(\bar{P}, \phi) \in \mathcal{P} \times \mathcal{F}$ :  $\forall n, \bar{P}^{\mathbf{G}(n)} \models \phi$  if and only if  $\forall n \leq m, \bar{P}^{\mathbf{G}(n)} \models \phi$ . Thus, an algorithm solving the PMCP is: given  $\bar{P} \in \mathcal{P}, \phi \in \mathcal{F}$ , output “Yes” if for every  $n \leq m$ , it holds that  $\bar{P}^{\mathbf{G}(n)} \models \phi$ , and “No” otherwise.  $\square$

**Remark 3.6** To solve PMCP, it is not enough to know that there exists a cutoff, one also needs to know the value of the cutoff in order to know when to stop enumerating the system instances.

For instance, suppose that for every  $k \in \mathbb{N}$  the data  $(\mathcal{P}, \mathbf{G}, k\text{-CTL}^*)$  has a cutoff. This means that for every  $k$  there is an algorithm deciding  $\text{PMCP}(\mathcal{P}, \mathbf{G}, k\text{-CTL}^*)$ . However, it does not imply that there is an algorithm that can decide  $\text{PMCP}(\mathcal{P}, \mathbf{G}, \text{ICTL}^*)$ . However, if the cutoffs are computable (i.e., there is an algorithm that given  $k \in \mathbb{N}$  outputs a cutoff for  $(\mathcal{P}, \mathbf{G}, k\text{-CTL}^*)$ ), then  $\text{PMCP}(\mathcal{P}, \mathbf{G}, \text{ICTL}^*)$  is decidable. Such a situation occurs in Section 4.

Cutoff results for  $(\mathcal{P}, \mathbf{G}, \mathcal{F})$  are often obtained by showing that for any  $\bar{P} \in \mathcal{P}$ , the set of runs of all system instances of  $\bar{P}^{\mathbf{G}}$  cannot be distinguished by any specification in  $\mathcal{F}$  from the

set of runs of the system instance  $\overline{P}^{\mathbf{G}(m)}$ , for some  $m \in \mathbb{N}$ . This is the case if one can show, for instance, that (i) for sufficiently many components, adding an extra component to the system only adds runs that can already be simulated in the smaller system, and (ii) a run in a system instance with many components can be simulated by a run in a system with a smaller number of components. This is often possible when the connectivity-graph is “homogeneous” (like a star or a clique), see [Emerson and Kahlon \[2000\]](#) and [German and Sistla \[1992\]](#). For the case of unidirectional rings and specifications in indexed CTL\* $\backslash X$ , [Emerson and Namjoshi \[1995, 2003\]](#) use stuttering bisimulations (see, e.g., [Browne et al. \[1988\]](#)) between tuples of processes in system instances of different size. To prove that two system instances are bisimilar, one has to find one or several maps (depending on the number of indexed variables) from components of the big system to components of the small system, such that any projection of a path of the big system with respect to these maps is stuttering equivalent to a path of the small system, and vice versa.



## CHAPTER 4

# Token-passing Systems

In a token-passing system (TPS), processes communicate by passing a token to their neighbors in the connectivity graph. We will define the passing of a token (with or without a value) as a special case of pairwise-rendezvous synchronization. In contrast to most of the other classes of systems in this survey, TPSs have been analyzed on complex connectivity graphs, where connections may or may not be labeled with directions. Thus, in order to define TPSs we extend our basic system model to *direction-aware parameterized systems*.

One special case has received attention in the literature, in particular in the work by Emerson and Namjoshi [1995, 2003] and Clarke et al. [2004] and recently by Aminof et al. [2014a]: connections in the graphs are not labeled with directions and the token does not hold a value. In this case, the only purpose of the token is to distinguish the process that currently holds it from all the other processes. This allows us to realize systems with mutual exclusion properties as TPSs, where the process with the token is the only one to access the shared resource. Connectivity graphs can then be used to model certain scheduling, e.g., token rings model a round-robin scheduling scheme.

Environment assumptions can be added to the specification in order to further restrict token passing, e.g., to only consider runs with fair token passing. Emerson and Namjoshi [1995, 2003] show that in this case the PMCP is decidable when the considered graphs are rings and specifications are in a fragment of  $\text{CTL}^*\backslash X$  with purely universal index quantification, and Clarke et al. [2004] show decidability for arbitrary graphs and specifications in  $k\text{-LTL}\backslash X$  (for any fixed  $k$ ). Recently, these two decidability results have been unified and extended by Aminof et al. [2014a].

We also consider systems where the token can hold different values, and, in later sections, look at systems where processes are aware of direction labels in the connectivity graph. While it is known that multi-valued tokens lead to an undecidable PMCP in general [Suzuki, 1988], there are decidability results for systems with multi-valued tokens and direction-awareness, subject to additional restrictions on the process and the specifications.

**Running Example.** As the running example of this chapter, we use a version of Milner's scheduler [Milner, 1989]. The scheduler is composed of an arbitrary number of components, each of which is responsible for activating one (unspecified) task and receives confirmation when its task has been executed. Scheduling should ensure that the tasks are activated in a round-robin scheme, and that every task has terminated before being activated again. This can naturally be modeled in token-passing systems, as noted by Emerson and Namjoshi [1995, 2003].

## 4.1 SYSTEM MODEL

To model TPSs, we first extend our general system model to *direction-aware parameterized systems*. Then we define TPSs as a special case, where token passing is modeled by restricted pairwise-rendezvous synchronization. Not all results of this chapter require this extension of the system model: results in Section 4.2 consider direction-unaware systems, while results in Section 4.3 consider direction-aware systems.

### 4.1.1 DIRECTION-AWARE PARAMETERIZED SYSTEMS

We extend the definition of parameterized systems to include additional labels on edges, called *directions*. The idea is that processes are allowed to restrict, depending on their local state, which edges of the connectivity graph can be used to synchronize with other processes.

A *direction-aware parameterized system* consists of the same ingredients as a (direction-unaware) parameterized system (see Section 2.2), except for the following modifications.

**Directions.** Fix finite sets  $Dir_{out}, Dir_{in}$  of *outgoing* and *incoming directions*.

**Directed connectivity graph.** A *d-ary directed connectivity graph* is a tuple  $G = (V, E, \text{type}, dir_{out}, dir_{in})$ , where  $(V, E, \text{type})$  is a *d-ary connectivity graph*, and  $dir_{out} : E \rightarrow Dir_{out}$  and  $dir_{in} : E \rightarrow Dir_{in}$  are additional labeling functions. A *parameterized d-ary directed connectivity graph* is a sequence  $\mathbf{G}$  of *d-ary directed connectivity graphs*.

**Example 4.1 Bi-directional Ring.** Let  $Dir_{out} = Dir_{in} = \{\text{cw}, \text{ccw}\}$ , standing for clockwise and counterclockwise directions in a ring. A *bi-directional ring* is a directional connectivity graph  $B = (V, E, \text{type}, dir_{out}, dir_{in})$  with

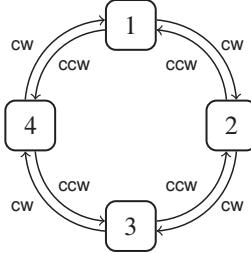
- $V = [k]$  for some  $k \geq 2$ ,
- $E = \{(i, i +_k 1), (i, i -_k 1) \mid i \in V\}$ , and
- $dir_{out}(i, i +_k 1) = dir_{in}(i, i +_k 1) = \text{cw}$ , and  $dir_{out}(i, i -_k 1) = dir_{in}(i, i -_k 1) = \text{ccw}$  for  $i \in V$ .

In general, *type* is arbitrary, but usually we consider bi-directional rings with a single process template.

A *parameterized bi-directional ring*  $\mathbf{B}$  is a sequence of  $B_1, B_2, \dots$  of bi-directional rings, usually with  $V(B_n) = [n]$ . A bi-directional ring with  $V = [4]$  is depicted in Figure 4.1.

**Direction-aware Transition Labels.** In a direction-aware parameterized system, process templates use transition labels from

$$\Sigma_{pr} := \Sigma_{int} \cup \{(out_a, d) : a \in \Sigma_{sync}, d \in Dir_{out}\} \cup \{(in_a, d) : a \in \Sigma_{sync}, d \in Dir_{in}\}.$$



**Figure 4.1:** A bi-directional ring with 4 nodes. Since  $dir_{out}(e) = dir_{in}(e)$  for all edges  $e$ , the direction is only displayed once.

**Direction-aware Synchronous Transitions.** Internal transitions remain as before, but synchronous transitions are now direction-aware. That is, a synchronous transition must satisfy the same properties as before, and additionally the direction  $d$  in the local transition label of the sender must match the labeling of the used edge by  $dir_{out}$ , and similarly with  $dir_{in}$  for the receiver(s).

Formally, *direction-aware synchronous transitions* are elements  $(s, t, s')$  of  $S \times \Sigma_{sync} \times S$  for which there exists a process index  $v \in V$  and a set  $\mathcal{I} \subseteq \{w \in V : E(v, w)\}$  of recipients of  $v$  in  $G$  such that:<sup>1</sup>

(CARD)  $|\mathcal{I}| \in \text{card}.$

(STEP)  $s(v) \xrightarrow{o_{out}, d} s'(v)$  is a local transition of  $P^{\text{type}(v)}$ .

(DIR) For every  $w \in \mathcal{I}$ ,  $dir_{out}(v, w) = d$ .

( $\mathcal{I}$ -STEP-DIR) For every  $w \in \mathcal{I}$ ,  $s(w) \xrightarrow{i_{in}, d_w} s'(w)$  is a local transition of  $P^{\text{type}(w)}$ , with  $d_w = dir_{in}(v, w)$ .

(FRAME) For every  $w' \in V \setminus \{v, w\}$ ,  $s'(w') = s(w')$ .

(MAX) There does not exist a strict superset  $\mathcal{I}' \supset \mathcal{I}$  of recipients of  $v$  in  $G$  such that the conditions above are satisfied, with  $\mathcal{I}'$  replacing  $\mathcal{I}$ , and  $s''$  replacing  $s'$ .

**Example 4.2 Directional Pairwise Rendezvous.** In a bi-directional ring, pairwise rendezvous as defined in Section 2.2.1 allows any process to synchronize with either of its two neighbors. We can define *directional pairwise rendezvous*, that may restrict which of its neighbors a process is allowed to synchronize with.

<sup>1</sup>Note that (CARD), (FRAME), and (MAX) are as before.

### 34 4. TOKEN-PASSING SYSTEMS

To obtain directional rendezvous, we instantiate the definition of directional synchronous transition for  $\text{card} = \{1\}$ . *Directional pairwise-rendezvous transitions* are of the form  $(s, a, s')$  for which there exists  $(v, w) \in E$  such that

**(STEP)**  $s(v) \xrightarrow{a!,d} s'(v)$  is a local transition of  $P^{\text{type}(v)}$ .

**(DIR)**  $dir_{out}(v, w) = d$ .

**(I-STEP-DIR)**  $s(w) \xrightarrow{a?,d_w} s'(w)$  is a local transition of  $P^{\text{type}(w)}$ , with  $d_w = dir_{in}(v, w)$ .

**(FRAME)** For every  $w' \in V \setminus \{v, w\}$ ,  $s'(w') = s(w')$ .

As with non-directional pairwise rendezvous, this incorporates the **(CARD)** condition with  $\mathcal{I} = \{w\}$ , and the **(MAX)** condition is trivially satisfied since  $\mathcal{I}' \supset \{w\}$  implies  $|\mathcal{I}'| \notin \text{card}$ .

**Remark.** Note that a direction-aware parameterized system degenerates to a parameterized system if  $|Dir_{out}| = |Dir_{in}| = 1$ . Also, by having either  $|Dir_{out}| = 1$  or  $|Dir_{in}| = 1$ , we can define parameterized systems where processes can choose which edges to use for synchronization only for incoming or only for outgoing edges.

#### 4.1.2 TOKEN-PASSING SYSTEMS

Fix a finite set  $\mathsf{T}$  of token values. If  $|\mathsf{T}| = 1$ , we call the token a *simple token*, otherwise a *multi-valued token*.

**Token-passing system template.** For token-passing systems, we consider only 1-ary system templates, and therefore do not distinguish between process templates and system templates. A *token-passing process template* with token values  $\mathsf{T}$  is a finite<sup>2</sup> LTS  $P = (Q, Q_0, \Sigma_{pr}, \delta, \lambda)$  with the following restrictions.

1. The state set  $Q$  is partitioned into two non-empty sets:  $Q = T \cup N$ . States in  $T$  are said to *have the token*.
2. The initial state set is  $Q_0 = \{\iota_T, \iota_N\}$  for some  $\iota_T \in T, \iota_N \in N$ .
3. The synchronous transition labels are given by the token values, i.e.,  $\Sigma_{sync} = \mathsf{T}$ .
4. Every transition  $q \xrightarrow{out_t,d} q'$  with  $t \in \mathsf{T}$  satisfies that  $q$  has the token and  $q'$  does not. We say that  $P$  *sends the token* with value  $t$  in direction  $d$ .
5. Every transition  $q \xrightarrow{in_t,d} q'$  with  $t \in \mathsf{T}$  satisfies that  $q'$  has the token and  $q$  does not. We say that  $P$  *receives the token* with value  $t$  from direction  $d$ .

<sup>2</sup>Note that all results of this chapter that state existence of a cutoff hold even for infinite-state process templates.

6. Every transition  $q \xrightarrow{a} q'$  with  $a \in \Sigma_{int}$  satisfies that  $q$  has the token if and only if  $q'$  has the token.
7. Every infinite action-labeled run  $a_0 a_1 \dots$  of  $P$  is in the set  $(\Sigma_{int}^* Dir_{out} \Sigma_{int}^* Dir_{in})^\omega \cup (\Sigma_{int}^* Dir_{in} \Sigma_{int}^* Dir_{out})^\omega$ . That is, the token is sent and received infinitely often in every run.

**A More Liberal Token-passing Restriction.** Restriction 6 above was introduced for direction-unaware systems by Emerson and Namjoshi [1995, 2003]. Aminof et al. [2014a] extended it to direction-aware systems, and showed that all the results in this chapter that state existence of cutoffs for TPSs also hold for the following, more liberal restriction:

- (†) From every state  $q$  that has the token there must be a path  $q \dots q'$  such that  $q'$  does not have the token, and from every state  $q$  that does not have the token there must be a path  $q \dots q'$  such that  $q'$  has the token.

**Notation.** Let  $\mathcal{P}_T$  be the set of all token-passing process templates with token values  $T$ . If  $|T| = 1$  then we write  $\mathcal{P}_{simp}$  instead of  $\mathcal{P}_T$ . Moreover, let  $\mathcal{P}_T^u$  denote the set of all process templates for which  $|Dir_{out}| = |Dir_{in}| = 1$ , i.e., direction-unaware processes.<sup>3</sup> If we require  $|Dir_{in}| = 1$ , then processes cannot choose from which directions to receive the token, but possibly in which direction to send it. Denote the set of all such process templates by  $\mathcal{P}_T^{snd}$ . Similarly, define  $\mathcal{P}_T^{rev}$  to be all process templates where  $|Dir_{out}| = 1$  — processes cannot choose where to send the token, but possibly from which direction to receive it.

**Example 4.3 Processes in Milner’s Scheduler.** Milner’s scheduler is composed of a variable number of processes, each responsible for scheduling one of the tasks. An implementation of such a process is depicted in Figure 4.2, where label  $A$  stands for activation of the task, and label  $C$  for completion of the task. Transitions to a state labeled with  $C$  are assumed to synchronize with actual completion of the task by an external device.

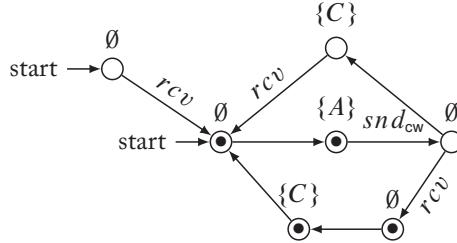
After activating the task, the process sends the token in direction  $cw$ , and afterwards waits for the return of the token, which may happen before or after completion of the task. Only after both of these events, the process will activate its task again.

**Token-passing System Instance.** To define token-passing systems, we instantiate the definition of a direction-aware parameterized system with

- $\Sigma_{sync} = T$ ,
- $d = 1$ ,

<sup>3</sup>For notational simplicity, we assume that  $Dir_{out}$  of the directed connectivity graph and the direction-aware process template are always identical, and similarly for  $Dir_{in}$ . One can easily define process templates to run on connectivity graphs with differing sets of directions, at the cost of notational overhead.

### 36 4. TOKEN-PASSING SYSTEMS



**Figure 4.2:** Process template for Milner’s scheduler. States with token are depicted with a dot inside the circle. State labels are depicted above states. Label  $A$  stands for activation of the task,  $C$  for completion of the task.

- $\text{card} = \{1\}$  (i.e., directional pairwise rendezvous), and
- process templates from  $\mathcal{P}_T$ .

Additionally, the global initial states  $S_0$  are restricted to those states  $s$  for which there exists a unique  $v \in V$  such that local state  $s(v)$  has the token.<sup>4</sup>

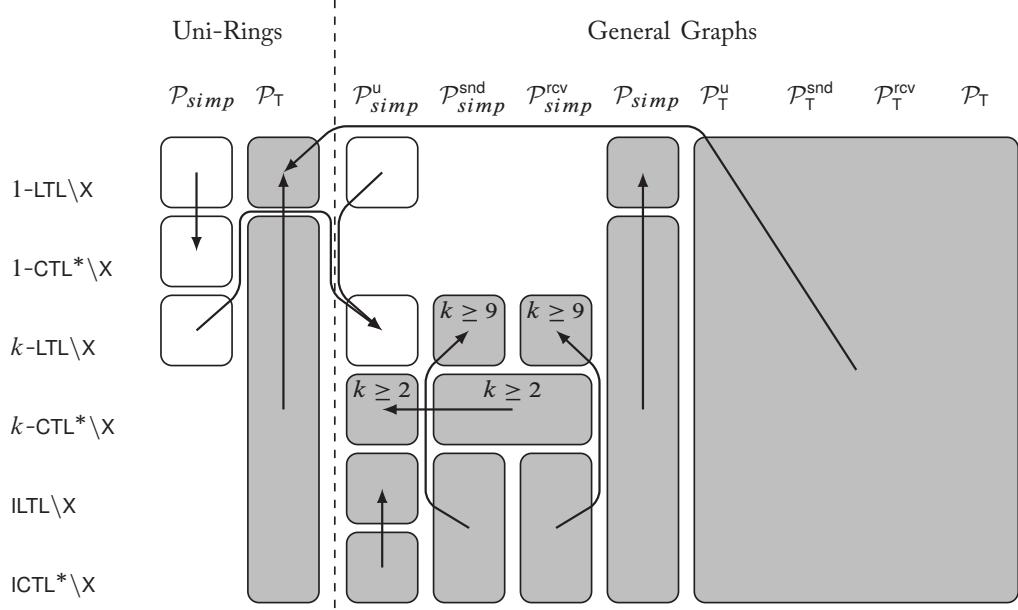
**Example 4.4 Milner’s Scheduler for 4 Tasks.** We obtain an implementation of Milner’s scheduler for four tasks by constructing a system instance with four process templates as given in Figure 4.2, arranged in a bi-directional ring as given in Figure 4.1.

Note that since the processes always send the token in direction  $cw$ , their behavior would be the same in a uni-directional ring. Our system model allows us to define variants of this example where, e.g., the token is non-deterministically sent in one of the directions  $cw$ ,  $ccw$ , and where the receiving process may accept the token from both directions, or only from one.

**Deadlocks.** Restrictions 6 and  $(\dagger)$  for token-passing process templates both guarantee the absence of deadlocks for direction-unaware systems. Note that the same is not true for direction-aware systems: the processes which are ready to receive might accept the token only from certain directions, which may be different from the directions into which the process that has the token can send it. This can lead to a deadlock if all process run into states from which the only possible transition is a token-passing transition. There are no known decidability results for parameterized deadlock detection in direction-aware TPSs.<sup>5</sup>

<sup>4</sup>Note that there is a slight difference between our model and that of Emerson and Namjoshi [1995, 2003]: in our model, one of the processes starts with the token, while in the original model nobody has the token, and the first transition of the system is a receive action of one process. There is, however, no difference in decidability of the two models, and the only difference in satisfaction of properties is in properties that mention the initial position of the token.

<sup>5</sup>Absence of deadlocks is guaranteed if we require that either (a) the process must always be able to reach a state such that it can receive the token from any direction, or (b) the process must always be able to reach a state such that it can send the token into any direction. Both (a) and (b) are more restrictive than  $(\dagger)$ , and incomparable to 6. The undecidability results in



**Figure 4.3:** Decidability results and reductions for token-passing systems, distinguished into connectivity graphs and classes of process templates on top, and fragments of  $\text{ICTL}^*\setminus X$  on the left. White boxes are decidability results, dark boxes are undecidability results. An arrow from box  $A$  to box  $B$  means that the result of  $A$  follows from the result, or a variation of the proof, of  $B$ . Blank items are not covered in the literature.

**Specifications.** For token-passing systems, we use indexed temporal logic specifications in fragments of  $\text{ICTL}^*\setminus X$ . Unless a different fragment is specified, in the following  $\varphi(i_1, \dots, i_n)$  stands for a formula in  $\text{ICTL}^*\setminus X$  with free index variables  $i_1, \dots, i_n$ .

## 4.2 RESULTS FOR DIRECTION-UNAWARE TOKEN-PASSING SYSTEMS

Figure 4.3 and Table 4.1 (on page 47) give an overview of the decidability results for TPSs. We consider results for direction-unaware systems (with process templates from  $\mathcal{P}_{simp}^u$  or  $\mathcal{P}_T^u$ ) in this section, and for direction-aware systems (with process templates from  $\mathcal{P}_T^{snd}$ ,  $\mathcal{P}_T^{rcv}$ , or  $\mathcal{P}_T$ ) in Section 4.3.

Theorem 4.17 and the first part of Theorem 4.18 also hold for templates that satisfy restriction (a), while the second part of Theorem 4.18 also holds for templates that satisfy (b).

## 38 4. TOKEN-PASSING SYSTEMS

### 4.2.1 DECIDABILITY FOR SIMPLE TOKEN-PASSING IN UNI-DIRECTIONAL RINGS

Consider the parameterized connectivity graph  $\mathbf{R}$  of uni-directional rings of size  $n$ , i.e.,  $\mathbf{R}(n) = (V, E)$  with  $V = [n]$  and  $E = \{(i, j) : j = i + 1 \bmod n\}$ . In a cornerstone paper, [Emerson and Namjoshi \[1995, 2003\]](#) prove that in uni-directional rings with a simple token it is enough to check a given property for all rings up to a given size, where the size depends on the quantification in the formula.

**Theorem 4.5** [[Emerson and Namjoshi, 2003](#)].  $\text{PMCP}(\mathcal{P}_{simp}^u, \mathbf{R}, \mathcal{F})$  has a cutoff if  $\mathcal{F}$  is one of the following fragments of  $\text{CTL}^* \setminus X$ .<sup>6</sup>

- (i) for  $\mathcal{F} = \{\forall i. \varphi(i)\}_{\varphi(i) \in \text{CTL}^* \setminus X}$ , the cutoff is 2.
- (ii) for  $\mathcal{F} = \{\forall i, j : j \in E(i). \varphi(i, j)\}_{\varphi(i, j) \in \text{CTL}^* \setminus X}$ , the cutoff is 3.
- (iii) for  $\mathcal{F} = \{\forall i, j : i \neq j. \varphi(i, j)\}_{\varphi(i, j) \in \text{CTL}^* \setminus X}$ , the cutoff is 4.
- (iv) for  $\mathcal{F} = \{\forall i, j, l : \text{neq}(i, j, l), j \in E(i). \varphi(i, j, l)\}_{\varphi(i, j, l) \in \text{CTL}^* \setminus X}$ , the cutoff is 5.

*Proof idea.* We first sketch the proof that 2 is a cutoff for formulas of the form  $\forall i. \varphi(i)$ , and then consider the case of multiple quantified index variables.

Consider formulas of the form  $\forall i. \varphi(i)$ . First, by symmetry of rings, a system instance  $P^{\mathbf{R}(n)}$  in satisfies  $\forall i. \varphi(i)$  if and only if it satisfies  $\varphi(0)$ . Second, consider system instances  $P_0^{\mathbf{R}(n)}$ , which are the same as  $P^{\mathbf{R}(n)}$ , except that the labeling function is restricted to atoms of the form  $p_0$  for  $p \in \text{AP}_{pr}$ . Then there is a stuttering bisimulation between  $P_0^{\mathbf{R}(n)}$  and  $P_0^{\mathbf{R}(2)}$  for all  $n \in \mathbb{N}$ . A stuttering bisimulation implies that both system instances agree on all  $\text{CTL}^* \setminus X$  formulas. The required bisimulation is defined as follows: two global states  $s, s'$ , in  $P_0^{\mathbf{R}(n)}, P_0^{\mathbf{R}(n')}$  are equivalent iff  $s(0) = s'(0)$ , i.e., the processes with index 0 are in the same state, and thus either both have the token or both do not have the token. The key reasons why this is a stuttering bisimulation is that

- the processes with index 0 can mimick each other,
- the behavior of other processes can be ignored, except that they must allow the token to arrive at process 0 again, and
- a process  $j \neq 0$  with the token can, by  $(\dagger)$ , send the token to its successor.

<sup>6</sup>In fact, the results of [Emerson and Namjoshi \[1995, 2003\]](#) hold for a logic that can also talk about local actions  $a_i$  of components. In the same vein, [Khalimov et al. \[2013a\]](#) proposed a local  $X$  operator over state labels as an extension of the logic that preserves the existence of cutoffs. None of the other results in this survey support such local properties, and therefore we have not included them in our definition in Section 2.4.1. We conjecture that most of the results in this chapter could be extended to such a logic.

Moreover, the labels of process 0 do not change during steps of the other processes, i.e., the system “stutters.”

For the case of multiple quantifiers, like  $\forall i, j : i \neq j. \varphi(i, j)$ , only the first quantifier can be instantiated with the fixed value 0, and for the second we need to consider all possible values for  $j$  in  $[n] \setminus \{0\}$ . In fact, we only need to consider the following cases:

- (i)  $j$  is recipient of  $i$ ,
- (ii)  $i$  is recipient of  $j$ , and
- (iii) there is no edge between  $i$  and  $j$ .

We define a monotone function  $h : [n] \rightarrow [m]$ , where  $m$  is the cutoff, such that each of (i), (ii), and (iii) holds for  $(0, j)$  in  $P^{\mathbf{R}(n)}$  if and only if it holds for  $(0, h(j))$  in  $P^{\mathbf{R}(m)}$ . The function  $h$  exists if  $m$  is chosen big enough, for this type of specification  $m = 4$  is sufficient. One can easily show that for every  $j$ , there is a stuttering bisimulation between processes 0,  $h(j)$  in  $P^{\mathbf{R}(m)}$  and processes 0 and  $j$  in  $P^{\mathbf{R}(n)}$ .  $\square$

**Example 4.6 Verification of Milner’s Scheduler.** As mentioned before, Milner’s scheduler is supposed to guarantee that all tasks are activated in a round-robin fashion, and that each task can terminate before being activated again (and only terminates after actually being activated). In Figure 4.2, activation of the task is represented by state label  $A$ , and termination by state label  $C$ . In a uni-directional ring, these requirements can then be expressed in the following parameterized specification:

$$\forall i, j, l : neq(i, j, l), j \in E(i). \text{AG} (A_i \rightarrow (\neg A_l \cup A_j))$$

$$\forall i. \text{AG} ((A_i \rightarrow (A_i \cup \neg A_i \cup C_i)) \wedge (C_i \rightarrow (C_i \cup \neg C_i \cup A_i))).$$

To verify this specification in systems with an arbitrary number of processes, it is sufficient to verify the first property in systems up to size 5 (by Theorem 4.5 (iv)), and the second property in systems up to size 2 (by Theorem 4.5 (i)). Thus, standard LTL model checking in rings of size up to 5 can be used to decide the PMCP for the given parameterized specification and implementation of Milner’s scheduler in uni-directional rings.

**Extensions of Cutoff Results for Rings.** Emerson and Namjoshi [1995, 2003] only provide cutoffs for properties with a small number  $k$  of index variables, but state that their technique is applicable for all  $k$ . Moreover, the original result is based on a strong restriction on process templates, stating that the sequence of actions from  $\{in, out\}$  along every infinite path in  $P$  strictly alternates between  $in$  and  $out$  actions. We note that for their proof to work, restriction (†) above suffices.

## 40 4. TOKEN-PASSING SYSTEMS

Aminof et al. [2014a] investigated ways to compute cutoffs in general.

**Theorem 4.7** [Aminof et al., 2014a]. *Let  $\mathcal{F}_k^*$  be the set of  $k$ -CTL\* $\setminus X$  formulas with only universal or only existential index quantifiers. Then, for any given  $k$ , PMCP( $\mathcal{P}_{simp}^u, \mathbf{R}, \mathcal{F}_k^*$ ) has cutoff  $2k$ .<sup>7</sup>*

*Proof idea.* This result is obtained by applying the general proof technique we will see in the proof of Theorem 4.10 to the special case of uni-directional rings. In this case, we can construct the  $d$ -contractions for every  $k$  explicitly, and note that for every  $k$  and  $d$ , the  $d$ -contraction is a ring of size at most  $2k$ .  $\square$

### 4.2.2 DECIDABILITY FOR SIMPLE TOKEN-PASSING IN GRAPHS

Some of the ideas for simple token-rings have been extended by Clarke et al. [2004] to simple token-passing systems on arbitrary classes of graphs. The main points of difference are: (i) properties are given in prefix  $\text{ILTL} \setminus X$ , and (ii) the decidability result is non-uniform and non-constructive. That is, the proof only shows that for every given  $k$  and parameterized graph  $\mathbf{G}$ , the PMCP for all formulas from  $k\text{-LTL} \setminus X$  is decidable. The proof does not supply an effective way to construct this decision procedure, and also makes no statement about the decidability of the PMCP for the full logic  $\text{ILTL} \setminus X$ .

**Theorem 4.8** [Clarke et al., 2004]. *PMCP( $\mathcal{P}_{simp}^u, \mathbf{G}, k\text{-LTL} \setminus X$ ) is decidable for every parameterized connectivity-graph  $\mathbf{G}$  and natural number  $k$ . Indeed, every  $(\mathcal{P}_{simp}^u, \mathbf{G}, k\text{-LTL} \setminus X)$  has a decomposition.*

*Proof idea.* Recall that a decomposition is a mapping from inputs of the PMCP to a set of finite model checking problems  $\text{MC}_{fin}$  and a formula  $\Phi_{\mathbb{B}}$  that combines the results of model checking (see Section 3.2). In this case,  $\text{MC}_{fin}$  is obtained by considering all so-called  $k$ -topologies of  $\mathbf{G}$ . These are graphs of size up to  $2k$ , obtained by fixing a set of  $k$  vertices in some  $\mathbf{G}(n)$ , and collapsing all other vertices into so-called hub vertices, such that indirect connections between the  $k$  fixed vertices are preserved.<sup>8</sup> One can show that for every graph  $G$ , evaluating a  $k\text{-LTL} \setminus X$ -formula  $\phi$  in  $P^G$  amounts to a Boolean combination of the model-checking results of  $\phi$  on the  $k$ -topologies of  $\mathbf{G}$ . Thus, the  $k$ -topologies, together with the original property, give the first part  $\text{MC}_{fin}$  of our decomposition. Since there are only finitely many  $k$ -topologies, and thus only finitely many Boolean formulas that combine these model checking results, one of them must be the  $\Phi_{\mathbb{B}}$  that is the second part of the decomposition.  $\square$

Looking closely at the decomposition, we note that the Boolean formula  $\Phi_{\mathbb{B}}$  depends on the quantifier structure of  $\phi$  as well as on  $\mathbf{G}$ . If there are no quantifier alternations, then  $\Phi_{\mathbb{B}}$  is just

<sup>7</sup>Khalimov et al. [2013a, Corollary 2] stated that  $2k$  is a cutoff for  $k\text{-LTL} \setminus X$ . Aminof et al. [2014a] noted that this is not the case for formulas with quantifier alternation.

<sup>8</sup>That is, there is a path between vertices  $v$  and  $v'$  that goes through a hub in the  $k$ -topology iff there is a path between  $v$  and  $v'$  that visits none of the  $k$  fixed vertices in the original graph. Note that this is a generalization of the idea by Emerson and Namjoshi [1995, 2003] that one of the main properties that needs to be preserved is whether or not vertices are neighbors.

a big conjunction or disjunction over all model-checking results for the  $k$ -topologies of  $\mathbf{G}$ , and thus we obtain an effective decision procedure whenever we can identify all of these  $k$ -topologies. If  $\phi$  contains quantifier alternations, it is in general not known how to obtain  $\Phi_{\mathbb{B}}$ .<sup>9</sup>

Aminof et al. [2014a] looked at the question whether this proof can be made uniform for arbitrary parameterized graphs  $\mathbf{G}$  and specifications in  $\text{ILTL} \setminus X$ . They show that for certain parameterized connectivity graphs, the (uniform) PMCP is undecidable.

**Theorem 4.9** [Aminof et al., 2014a]. *There exists a parameterized connectivity graph  $\mathbf{G}$  such that  $\text{PMCP}(\mathcal{P}_{simp}^u, \mathbf{G}, \text{ILTL} \setminus X)$  is undecidable.*

*Proof idea.* Define  $P$  to be the process template with two states, one with and one without the token. Every transition must change the state, which, in particular, satisfies condition ( $\dagger$ ) from the definition of TPSs in Section 4.1.2. For  $i, j \in \mathbb{N}$  let  $L_{i,j}$  be the lasso with prefix of length  $i$ , and a loop of length  $j$ . Based on some computable ordering on all deterministic Turing machines (for example, lexicographic), let  $\mathbf{G}$  consist of all graphs of the form  $L_{i,j}$  such that the  $j$ th Turing machine halts on the empty word in at most  $i$  steps. Note that this is a computable set.

For every  $k \in \mathbb{N}$ , let  $\varphi_k$  be the formula (in  $k\text{-LTL} \setminus X$ )

$$\forall i_1, \dots, i_k : \text{neq}(i_1, \dots, i_k). \neg (\mathsf{F}(\text{tok}_{i_1} \cup \text{tok}_{i_2} \cup \dots \cup \text{tok}_{i_k} \cup \text{tok}_{i_1})) ,$$

where  $\text{tok}_i$  is a predicate which is true iff process  $i$  has the token. Thus,  $P^G \models \varphi_k$  if and only if  $G$  does not contain a ring of size  $k$  as a subgraph.

Now reduce the non-halting problem for Turing Machines to the PMCP:

$$\begin{aligned} & \text{the } k\text{th TM does not halt} \\ \Leftrightarrow & \mathbf{G} \text{ does not contain a graph with a ring of size } k \text{ as subgraph} \quad \square \\ \Leftrightarrow & P^G \models \varphi_k \text{ for all } G \in \mathbf{G}. \end{aligned}$$

For token-passing networks with specifications in  $\text{ICTL} \setminus X$ , Clarke et al. [2004] showed that decompositions of the form they define cannot exist in general. However, Aminof et al. [2014a] showed that by further refining the construction of Clarke et al. [2004], a non-uniform decidability result can be obtained even for specifications in  $\text{ICTL}^* \setminus X$ . To this end,  $\text{ICTL}^* \setminus X$  is stratified not only with respect to the number  $k$  of index quantifiers, but also with respect to the number  $d$  of (nested) path quantifiers. In the following, let  $k\text{-CTL}_d^* \setminus X$  be the set of all sentences in  $k\text{-CTL}^* \setminus X$  with nesting depth of path quantifiers at most  $d$ .

**Theorem 4.10** [Aminof et al., 2014a]. *Let  $\mathbf{G}$  be a parameterized connectivity graph. Then for all  $k, d \in \mathbb{N}$ ,  $\text{PMCP}(\mathcal{P}_{simp}^u, \mathbf{G}, k\text{-CTL}_d^* \setminus X)$  is decidable. Indeed, every  $(\mathcal{P}_{simp}^u, \{\mathbf{G}\}, k\text{-CTL}_d^* \setminus X)$  has a decomposition.*

<sup>9</sup>Clarke et al. [2004] claimed in the discussion of their theorem that “given  $k$  and network graph  $\mathbf{G}$ , all  $k$ -indexed LTL $\setminus X$ -specifications have the same reduction,” where a reduction is equivalent to what we call a decomposition. It should be noted that different formulas  $\Phi_{\mathbb{B}}$  may be necessary for specifications with different structures of indexed quantifiers.

## 42 4. TOKEN-PASSING SYSTEMS

*Proof idea.* To show the existence of decompositions, define two graphs  $G, G'$  to be  $(k, d)$ -equivalent if they cannot be distinguished by formulas in  $k\text{-CTL}_d^*\setminus X$  that use  $tok_i$  as the only atomic predicates. The existence of decompositions is proven by two observations: (i) every graph is  $(d, k)$ -equivalent to its  $d$ -contraction (a certain minimal graph with the same observed behavior), and (ii) there are only finitely many different  $d$ -contractions for a given  $k$ . By a similar argument as in the proof of Theorem 4.8, the PMCP for a given parameterized graph  $\mathbf{G}$ , process template  $P$  and specification  $\phi$  can thus be reduced to a Boolean combination of checking whether  $P^{Con} \models \phi$ , for all  $d$ -contractions  $Con$  of  $\mathbf{G}$ .  $\square$

Note that like Theorem 4.8, this result is non-uniform and non-constructive, and therefore does not supply us with an effective decision procedure for the PMCP with specifications from  $\text{ICTL}^*\setminus X$ . However, for specifications without quantifier alternations and for fixed  $\mathbf{G}$  (e.g.,  $\mathbf{G} \in \{\mathbf{R}, \mathbf{B}, \mathbf{C}\}$ ), the proof methods can be used to obtain concrete cutoffs for the PMCP.

We have stated before that the uniform PMCP is undecidable for specifications in  $\text{ILTL}\setminus X$ . For branching-time specifications, undecidability is already obtained with a fixed number of two index quantifiers, as long as the nesting depth of path quantifiers is not bounded.

**Theorem 4.11 [Aminof et al., 2014a].** *There exists a parameterized connectivity graph  $\mathbf{G}$  such that  $\text{PMCP}(\mathcal{P}_{simp}^u, \mathbf{G}, 2\text{-CTL}^*\setminus X)$  is undecidable.*

*Proof idea.* The proof is a variation of the proof of Theorem 4.9. Again, we define a parameterized graph  $\mathbf{G}$  and a sequence of formulas  $\varphi_k$  such that the  $k$ th TM does not halt iff  $\varphi_k$  holds in all system instances. The main difference is that now we can use arbitrary nestings of temporal path quantifiers to distinguish these graphs (similar to those used in Clarke et al. [2004, Theorem 5]), and therefore two index variables are sufficient to define all  $\varphi_k$ .  $\square$

### 4.2.3 UNDECIDABILITY RESULTS FOR MULTI-VALUED TOKENS

The decidability results discussed in Sections 4.2.1 and 4.2.2 consider token-passing systems with simple tokens. Suzuki [1988] and Emerson and Namjoshi [1995, 2003] considered systems where the token can take multiple values.

**Theorem 4.12 [Suzuki, 1988].** *Let 1-Safe be the safety fragment of 1-LTL $\setminus X$ . Then  $\text{PMCP}(\mathcal{P}_T^u, \mathbf{R}, 1\text{-Safe})$  is undecidable for sufficiently large  $T$ .*

*Proof idea.* (based on Emerson and Namjoshi [2003, Section 6]) The main idea is to simulate  $n$  steps of a 2CM in a ring of size  $n$ , where one process in the ring will eventually raise a flag ‘halt’ if and only if the 2CM halts in  $n$  steps. Given a 2CM  $\mathcal{M}$ , we describe how the corresponding finite-state process  $P$  works.

In the first step, the process that starts with the token moves into a special state with the intention that it will simulate the control-state of  $\mathcal{M}$ , and it sets a flag to false. We will call this

process the *controller*. Each of the remaining processes will be used to store a fixed number of bits—these are called *storage* processes. The values in  $\mathsf{T}$  represent different commands that the controller can send to the storage processes.

After the first step, the controller sends around the token with an “initialize” command, which makes all other processes go into storage mode. In this mode, each process stores three bit of information, corresponding to the two counters of  $\mathcal{M}$ , and a “step-counter,” respectively. Each counter is stored as a tally in the ring: for every counter, each process (except the controller) stores one bit in unary encoding, and the number of bits in the ring set to 1 is interpreted as the value of that counter. Thus, a ring of size  $n$  can store counter values up to  $n - 1$ . Initially, all counter-bits are set to 0. The controller implements an increment of a counter  $c$  by circulating a token with the command “increment  $c$ .” A process receiving “increment  $c$ ” will either increment its corresponding counter-bit from 0 to 1 (if possible), and pass a modified token with the “empty” command, or it will not change its internal state and pass the token unchanged (if local increment is not possible). Thus, the value of the token when it returns to the controller indicates whether the command was successful. There are similar commands for decrementing and testing for zero. After the controller implements a step of  $\mathcal{M}$ , it also increments the step-counter.

The controller proceeds in this way until its “increment step-counter” command returns to it unsuccessful (meaning that  $n$  steps of  $\mathcal{M}$  have now been simulated). If after  $n$  steps the machine  $\mathcal{M}$  did not enter a halting state, then the controller sets its flag to true. This completes the description of  $P$ . It is a finite-state process that will move into controller mode if it has the token initially, and into storage mode otherwise. The controller sets its flag to true iff  $\mathcal{M}$  does not halt in  $n$  steps. Since the non-halting problem is undecidable, so is deciding if for all  $n$ , the controller in a ring with process implementation  $P$  sets the flag to true. This proves that the PMCP is undecidable for uni-directional multi-valued token rings.  $\square$

**Binary token.** Emerson and Namjoshi [2003] mentioned a simple way to simulate a multi-valued token with arbitrary set of values  $\mathsf{T}$  using a binary token, i.e., a token with  $\mathsf{T}_{\mathbb{B}} = \{0, 1\}$ . Thus, they obtain another undecidability result.

**Theorem 4.13 [Emerson and Namjoshi, 2003].**  $\text{PMCP}(\mathcal{P}_{\mathbb{B}}^{\mathsf{u}}, \mathbf{R}, \text{1-Safe})$  is undecidable.

**Proof idea.** The general proof idea is the same as in the proof of Theorem 4.12. Additionally, the controller of  $\mathcal{M}$  can use the binary token to transmit arbitrary commands from the set of commands  $\mathsf{T}$  of that proof in a unary encoding: assume commands are numbered, i.e., we have  $\mathsf{T} = [n]$  for some  $n \in \mathbb{N}$ . To encode command  $t \in [n]$ , the controller sends the token with value 1 for  $t$  times, followed by one transmission with value 0. Each storage node internally keeps track of how many 1s it received, say in a variable  $x$ . After receiving the token with value 0, every node knows which command should be executed. Now the controller again sends a 1, and every node checks if it can execute the command, i.e., whether its counter can be in- or decremented, or whether it can witness that the overall counter value is not 0. The first node that can actually

#### 44 4. TOKEN-PASSING SYSTEMS

execute the command (or witness that the overall counter value is not 0) changes the value of the token to 0 (letting the following storage nodes know that they should not execute the command anymore). Regardless of the value of the token in this round, all nodes reset their command variable  $x$  to 0 when they pass the token. As before, the value of the token informs the controller whether the command was successful. This procedure can be repeated whenever a command is passed to the storage nodes in the original construction.  $\square$

**Multi-valued Tokens in General Graphs.** Since rings are a particular form of general graphs,  $\text{PMCP}(\mathcal{P}_{\text{T}_B}^u, \mathbf{G}, \text{1-Safe})$  is in general undecidable. However, uni-directional rings have the characteristic property that, from the perspective of a given process, there is exactly one process from which it always receives the token, and exactly one to which it can send it. This is not the case in graphs with in- or out-degree greater than 1. Since the undecidability proof depends on the fact that the token can be used to pass certain commands to all other processes, it is not immediately obvious that it also works for other parameterized graphs. At least for bi-directional rings, we believe this is possible.

**Conjecture 4.14**  $\text{PMCP}(\mathcal{P}_{\text{T}_B}^u, \mathbf{B}, k\text{-LTL}\setminus X)$  is undecidable.

For general graphs, the problem is mostly open.

**Problem 4.15** For which parameterized connectivity graphs  $\mathbf{G}$  is  $\text{PMCP}(\mathcal{P}_{\text{T}_B}, \mathbf{G}, \mathcal{F})$  decidable, for  $\mathcal{F} \in \{\text{ILTL}\setminus X, \text{ICTL}^*\setminus X\}$ ?

### 4.3 RESULTS FOR DIRECTION-AWARE TOKEN-PASSING SYSTEMS

In the results considered thus far, if a node has multiple recipients, then it cannot choose which one will receive the token, but one of the possible transitions will fire non-deterministically. In this section, we consider systems that are direction-aware, i.e., where processes can choose who will receive the token and/or from whom they want to receive it. We first give a decidability result for a special class of systems that can be used to model a leader election protocol [Emerson and Kahlon, 2004], and then a general undecidability result.

#### 4.3.1 CUTOFFS FOR CHANGE-BOUNDED TOKENS IN BI-DIRECTIONAL RINGS

Let  $\mathbf{B}$  be the parameterized connectivity-graph of bi-directional rings of size  $n$ , i.e.,  $\mathbf{B}(n) = (V, E)$  with  $V = [n]$  and  $E = \{(i, j), (j, i) : j = i + 1 \bmod n\}$ . Let  $Dir_{out} = \{\text{cw}, \text{ccw}\}$ , and  $dir_{out}$  the function that maps edges  $(i, i + 1 \bmod n)$  to cw, and edges  $(i + 1 \bmod n, i)$  to ccw. Let  $Dir_{in} = \{\text{undefined}\}$ , and  $dir_{in}$  be the function that maps all edges to undefined. Let  $\mathbf{B}^{\text{rnd}}$  be the parameterized bi-directional ring with these direction labels. Let  $\mathcal{D}_{\text{T}}^{\text{rnd}}$  be the set of all deterministic process templates in  $\mathcal{P}_{\text{T}}$ .

We call a parameterized (direction-aware) TPS  $(\mathcal{P}_T, \{\mathbf{G}\})$  *change-bounded* if there exists  $b \in \mathbb{N}$  such that, for every run of every system instance  $\mathcal{P}_T^{\mathbf{G}(n)}$ , the value of the token does not change more than  $b$  times.<sup>10</sup>

**Theorem 4.16 [Emerson and Kahlon, 2004].**  $\text{PMCP}(\mathcal{D}_T^{\text{snd}}, \mathbf{B}^{\text{snd}}, \text{1-LTL} \setminus X)$  is decidable if  $(\mathcal{D}_T^{\text{snd}}, \mathbf{B}^{\text{snd}})$  is change-bounded. Indeed, the cutoff is a polynomial depending on the number of states and the transition graph of process template  $P$ .

*Proof idea.* The proof is based on the fact that processes are deterministic, and so the token can only make a finite number of moves until the system either deadlocks or runs into a state that it has seen before, entering an infinite loop. The number of steps until the system reaches such a state is independent of the size of the ring, as long as it has a certain minimal size. This size can be computed by constructing the transition graph for the ring, based on the definition of  $P$ . It gives a cutoff for  $\text{PMCP}(\mathcal{D}_T^{\text{snd}}, \mathbf{B}^{\text{snd}}, \text{1-LTL} \setminus X)$ .  $\square$

Note that this result is not displayed in Figure 4.3, since it considers special restrictions to deterministic process templates and change-bounded tokens.

### 4.3.2 UNDECIDABILITY FOR DIRECTION-AWARE TPSS

Since direction-aware TPSs can degenerate to direction-unaware TPSs, the undecidability results from Section 4.2.3 also hold here, and decidability can only be obtained by adding restrictions.

Note that while Theorem 4.16 gives a cutoff for a system with limited direction-awareness and multi-valued tokens, it is very restricted in the specifications, the processes and the graphs (including direction-awareness) that it can handle. If we consider  $|Dir_{in}| > 1$ , Aminof et al. [2014a] showed that undecidability follows even for systems with a simple token.

Now, consider the case  $Dir_{out} = Dir_{in} = \{\text{cw}, \text{ccw}\}$ , and  $dir_{out} = dir_{in}$  the function that maps edges in clockwise directions to cw, and edges in counterclockwise direction to ccw. Let  $\mathbf{B}^{dir}$  be the parameterized bi-directional ring with these direction labels.

**Theorem 4.17 [Aminof et al., 2014a].**  $\text{PMCP}(\mathcal{P}_{simp}, \mathbf{B}^{dir}, \text{1-LTL} \setminus X)$  is undecidable.

*Proof idea.* The proof goes along the lines of the proofs of Theorems 4.12 and 4.13. The main new idea is that directions can be used to encode the different commands of the controller to the storage nodes, similar to the encoding with a binary token.  $\square$

<sup>10</sup>An example is an integer-valued token with fixed starting value  $b$ , that can only be decreased, and never become smaller than 0.

#### 46 4. TOKEN-PASSING SYSTEMS

Furthermore, undecidability with respect to  $k\text{-LTL} \setminus X$  has been shown even if only one of  $|Dir_{in}| > 1$  or  $|Dir_{out}| > 1$  holds, for sufficiently large  $k$ .

**Theorem 4.18** [Aminof et al., 2014a]. *Assume that  $|Dir_{out}| > 1$ . Then there exist a parameterized directed connectivity graph  $\mathbf{G}$  such that  $\text{PMCP}(\mathcal{P}_{simp}^{\text{snd}}, \mathbf{G}, 9\text{-LTL} \setminus X)$  is undecidable. Similarly, for  $|Dir_{in}| > 1$  and suitable  $\mathbf{G}'$ ,  $\text{PMCP}(\mathcal{P}_{simp}^{\text{rcv}}, \mathbf{G}', 9\text{-LTL} \setminus X)$  is undecidable.*

*Proof idea.* The proof is again a reduction from the non-halting problem of 2CMs and requires a parameterized graph with eight special nodes that are directly connected to the controller, and represent the different commands of the 2CM (increment counter, decrement counter, counter is zero, counter is not zero; each for counters 1 and 2). The other nodes are storage nodes, like in the undecidability proofs before.

Consider the case where  $|Dir_{out}| = 1$ : since all edges have the same outgoing label, processes cannot choose where to send the token. We use a special construction of the parameterized graph and the process template such that every possible path of the token through the graph corresponds to the (partial) execution of one command, represented by the special node that the token visits after leaving the controller. To circumvent the restriction that the controller cannot choose the sending direction, we use a specification that tracks whether the special node that the token is sent to is always the one that the controller intended (represented by the state of the controller). This ensures that we only consider runs of the system that correspond to an execution of the 2CM.

For  $|Dir_{in}| = 1$  we can use a similar construction, except that now the controller can issue a certain command, but the storage processes cannot detect which one is intended, and have to guess which command to execute. The special nodes are in this case visited immediately before the token returns to the controller. Again, we use the specification to ensure that only the intended runs are considered.  $\square$

#### 4.4 DISCUSSION

Even in systems with a relatively restricted communication primitive such as passing a valueless token, the PMCP is undecidable if we do not additionally restrict the graph structure or stratify specifications with respect to the number of index and path quantifiers. Undecidability proofs show that such systems can simulate a 2CM, and thus halting of the 2CM could be decided if we could decide the PMCP.

For certain concrete classes of graphs, such as rings, cutoffs can be found by hand. To prove that  $c$  is a cutoff for a class of systems and specifications in  $\text{ICTL}^* \setminus X$ , it suffices to show that there is a stuttering bisimulation between a system with arbitrarily many components and a system with exactly  $c$  components. For specifications that quantify over multiple processes, this needs to take into account whether processes are direct neighbors or not, and consider all possible cases.<sup>11</sup>

<sup>11</sup>In fact, Aminof et al. [2014b] recently showed that cutoffs can be computed automatically for large classes of graphs that are definable either by graph-grammars or in Monadic Second-Order Logic. Examples include rings, cliques, lines, trees,

If tokens can carry values, decidability is lost even in rings. Similarly, if processes can distinguish directions in the graph, we get undecidability in bi-directional rings. Figure 4.3 (on page 37) and Table 4.1 give an overview of decidability results for token-passing systems with these properties, showing that most of the considered cases are undecidable. To recover decidability, additional restrictions on graphs, processes, or specifications, are necessary.

**Table 4.1:** Decidability results for TPSs and their sources, separated into systems with direction-unaware processes and simple token ( $\mathcal{P}_{simp}^u$ ), systems with direction-unaware processes and multi-valued token ( $\mathcal{P}_T^u, \mathcal{P}_{T\Box}^u$ ), and systems with direction-aware processes

Result	Processes	Graph	Specification	References
a cutoff	$\mathcal{P}_{simp}^u$	<b>R</b>	fragments of $\text{ICTL}^* \setminus X$	[Emerson and Namjoshi 2003] [Aminof et al. 2014a]
decidability for all $k$	$\mathcal{P}_{simp}^u$	$\forall G$	$k\text{-LTL} \setminus X$	[Clarke et al. 2004]
decidability for all $k, d$	$\mathcal{P}_{simp}^u$	$\forall G$	$k\text{-CTL}_d^* \setminus X$	[Aminof et al. 2014a]
undecidability	$\mathcal{P}_{simp}^u$	$\exists G$	$\text{ILTL} \setminus X,$ $2\text{-CTL}^* \setminus X$	[Aminof et al. 2014a]
undecidability	$\mathcal{P}_T^u$	<b>R</b>	1-Safe	[Suzuki 1988]
undecidability	$\mathcal{P}_{T\Box}^u$	<b>R</b>	1-Safe	[Emerson and Namjoshi 2003]
decidability under bounded change	$\mathcal{D}_T^{snd}$	<b>B</b> <sup>snd</sup>	$1\text{-LTL} \setminus X$	[Emerson and Kahlon 2004]
undecidability	$\mathcal{P}_{simp}$	<b>B</b> <sup>dir</sup>	$1\text{-LTL} \setminus X$	[Aminof et al. 2014a]
undecidability for $ Dir_{out}  > 1$ , for $ Dir_{in}  > 1$	$\mathcal{P}_{simp}^{snd},$ $\mathcal{P}_{simp}^{rcv}$	$\exists G$	$9\text{-LTL} \setminus X$	[Aminof et al. 2014a]

For all of the decidability results in this chapter, a restriction to some notion of fairness is essential. While some of the original results are explicitly restricted to systems or runs where every process receives and sends the token infinitely often, we argue here (and in Aminof et al. [2014a]) that a weaker condition is sufficient, namely that from every state of the process there *exists* a path such that it sends or receives the token—but results are not restricted to runs where such paths

series-parallel graphs, but not grids. From a description (i.e., a grammar or a formula) of such a set of graphs, one can build an algorithm that computes a cutoff for a given formula and thus solves the PMCP for indexed  $\text{CTL}^* \setminus X$  over this set of graphs.

## 48 4. TOKEN-PASSING SYSTEMS

are actually taken. There are no decidability results in the literature that consider systems without any fairness assumption.

### 4.4.1 VARIATIONS OF THE MODEL

The basic idea behind structuring concurrent applications using a token is that the current holder of the token is privileged, and thus allowed to perform some special action such as entering its critical section. That is, tokens are a means to resolve conflicts over shared resources. However, systems with a single token are based on the assumption that at each time at most one process may be privileged, or equivalently, there is only one shared resource. This severely limits the degree of concurrency in the system.

There are several schemes to increase the degree of concurrency. For instance, any solution to the celebrated dining philosophers problem allows several (non-neighboring) philosophers to eat at the same time. A generalization of dining philosophers are the drinking philosophers [[Chandy and Misra, 1984](#)] for general conflict graphs, where again one requires that two neighbors are not drinking at the same time, in case both are currently competing for the same resource. In the extreme case, a completely connected conflict graph then boils down to having at most a single neighbor drinking at a time, and thus to a single token scheme. Intuitively, the dining (or drinking) philosophers on general graphs can thus be interpreted as a token scheme with multiple tokens.

The literature contains results on both areas, access to multiple shared resources and multiple tokens. Regarding multiple shared resources, there are several cutoff results. Ensuring fair access is considered by [Emerson and Kahlon \[2002\]](#), FIFO access is considered by [Bouajjani et al. \[2008\]](#), and strict alternation rules as provided by link reversal algorithms [[Barbosa and Gafni, 1989](#)] is considered by [Függer and Widder \[2012\]](#). Regarding multiple tokens, [Emerson and Kahlon \[2004\]](#) contains a decomposition result that allows to reduce reasoning about 2-indexed properties of uni-directional rings with multiple multi-valued tokens, to reasoning in small rings. To obtain decidability, there are several restrictions on token-passing and the processes: (i) in any run, every multi-valued token can only change its value a bounded number of times, (ii) every token has an “owner” process, and transitions that send or receive a token cannot change the state of a process unless the token is owned by it, and (iii) processes have to be deterministic.

Considering the case with multiple tokens, we conjecture that decidability strongly depends on whether or not processes can distinguish different tokens. If this is the case, then the problem is very similar to a multi-valued token and should quickly become undecidable. For indistinguishable tokens, there is more hope to obtain decidability results.

**Problem 4.19** Do any of the decidability results in this chapter still hold for systems with multiple tokens? In particular, do they hold if processes *cannot* distinguish different tokens?

Furthermore, all previous work on TPSs only considers prenex ITL. An interesting question is under which circumstances the given results hold if we allow index quantifiers inside of temporal quantifiers.

**Problem 4.20** Are there fragments of non-prenex ITL that can express interesting properties that are not in  $\text{ICTL}^* \setminus X$  and still have a decidable PMCP for any of the cases considered in this chapter?



## CHAPTER 5

# Rendezvous and Broadcast

This chapter considers systems of processes communicating via pairwise rendezvous, asynchronous rendezvous, broadcast, and combinations of these (as defined in Section 2.2.1). We have already seen in Examples 2.1 and 2.2 how pairwise-rendezvous models standard synchronization between exactly two processes. As discussed by Delzanno et al. [2002], asynchronous rendezvous and broadcast also correspond to standard coordination constructs in different programming languages, such as the `notify` and `notifyAll` constructs of the Java programming language.

The specifications considered in this chapter are either action-based, i.e.,  $\omega\text{Reg}(A)$  and  $\text{Reg}(A)$ , or state-based, i.e.,  $\text{LTL}(C)$  which are LTL formulas whose atoms are indexed by the controller process, and  $\forall U \text{LTL}(U)$  which are LTL formulas that hold for every user process.

Figure 5.1 and Table 5.1 (on page 53) summarize results taken from the literature [Emerson and Kahl, 2003b, Esparza et al., 1999, German and Sistla, 1992], including minor extensions. The figure indicates that safety is always decidable, while liveness is decidable for pairwise rendezvous and undecidable for the other communication primitives.

**Running example.** As the running example of this chapter, we will revisit the simple Client/Server system from Chapter 2, including a number of variants with different communication primitives.

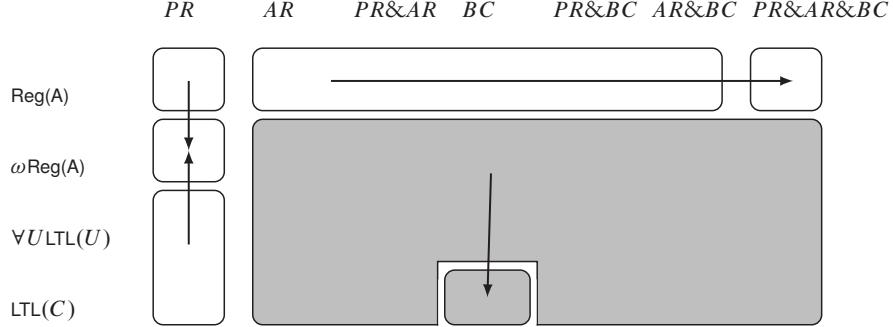
### 5.1 SYSTEM MODEL

**Synchronization constraints.** As in Section 2.2.1, pairwise rendezvous is defined by taking  $\text{card} = \{1\}$ , asynchronous rendezvous is defined by taking  $\text{card} = \{0, 1\}$ , and broadcast is defined by taking  $\text{card} = \mathbb{N}_0$ . Systems with all three primitives, for example, are defined by taking  $\overline{\text{card}} = \{\{1\}, \{0, 1\}, \mathbb{N}_0\}$ .

**Example 5.1 Client/Server (revisited)** Recall the simple client/server system from Examples 2.1 and 2.2. Example 2.2 illustrates pairwise rendezvous where the server can synchronize with a single client at a time.

For asynchronous rendezvous, there are two possibilities if a client is ready to take a transition labeled with  $\text{out}_{\text{enter}}$  or  $\text{out}_{\text{leave}}$ . First, the server and the client take a synchronized transition, as in pairwise rendezvous. The second case may occur when the client is ready to effectuate a transition labeled with  $\text{out}_{\text{enter}}$  or  $\text{out}_{\text{leave}}$ , but from the current state of the server there is no outgoing

## 52 5. RENDEZVOUS AND BROADCAST



**Figure 5.1:** PMCP results for systems with a controller, and clique connectivity-graph. The abbreviations PR, AR, BC stand for pairwise rendezvous, asynchronous rendezvous, broadcasts. White boxes are decidability results, dark boxes are undecidability results. An arrow from box  $A$  to box  $B$  means that the result of  $A$  follows from the result, or variation of the proof, of  $B$ .

transition labeled with  $i n_{\text{enter}}$  or  $i n_{\text{leave}}$ , respectively. In this case, the client may effectuate its transition alone, that is, without the server. One may view this as the client transmitting information to the server in case the server is ready to receive one, otherwise the message is “lost.”

For broadcast, consider the connectivity graph in Figure 5.2(a), and the process templates in Figures 5.2(b) and 5.2(c), where the server has a transition from  $0_S$  labeled with  $\text{enter}!!$ , and all the clients have transitions from  $0_C$  labeled with  $\text{enter}??$ . In this case, the synchronization is initiated by the server, and any client that is ready to take the synchronous step can enter  $1_c$ .

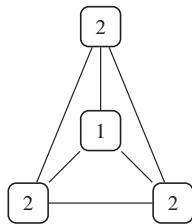
**Parameterized connectivity graph.** A graph  $G = (V, E)$  is a *clique* if for all distinct  $v, w \in V$  it holds that  $(v, w) \in E$ . The *2-ary parameterized clique graph* is the sequence  $n \mapsto \mathbf{C}_\circ(n)$  where  $\mathbf{C}_\circ(n)$  is the clique with  $n + 1$  vertices,  $\text{type}(1) = 1$ , and  $\text{type}(i) = 2$  for all  $i \neq 1$ . The unique process in  $\mathbf{C}_\circ(n)$  of type 1 (it has index 1) is called the *controller*, and all other processes are called *users* and are of type 2.

**Example 5.2 Clique topology for Client/Server system** Figure 5.2(a) shows a clique topology with one controller process (modeling the server), and three user processes (modeling the clients). Note that with the process templates defined for server and clients in Figures 2.1(b) and 2.1(c), the behavior of the composed system does not change, since the process templates do not define synchronous transitions between two clients.

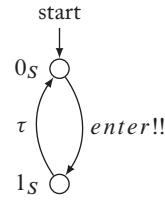
**Runs and Deadlocks.** In this chapter, system runs are defined as *infinite* paths that start in the initial state (cf. Section 2.2.2). The only exception is Theorem 5.7 about the action-based specification language  $\text{Reg}(A)$  which is interpreted over finite paths that start in the initial state.

**Table 5.1:** Decidability results, and their sources, for systems with a controller, using pairwise rendezvous (PR), asynchronous rendezvous (AR), and broadcast (BC)

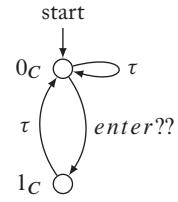
Result	Processes	Graph Specification	References
decidability	PR+BC+AR	$\mathbf{C}_\odot$	$\text{Reg}(A)$ [Esparza et al. 1999] [Emerson and Kahlon 2003b]
decidability in EXPSPACE	PR	$\mathbf{C}_\odot$	$\omega\text{Reg}(C)$ [German and Sistla 1992]
decidability in EXPSPACE	PR	$\mathbf{C}_\odot$	$\forall U \omega\text{Reg}(U)$ [German and Sistla 1992]
decidability in EXPSPACE	PR	$\mathbf{C}_\odot$	$\omega\text{Reg}(A)$ [Emerson and Kahlon 2003b] cf. [German and Sistla 1992]
undecidability	BC	$\mathbf{C}_\odot$	$\omega\text{Reg}(A)$ [Esparza et al. 1999]
undecidability	BC	$\mathbf{C}_\odot$	$\text{LTL}(C)$ [Emerson and Kahlon 2003b]
undecidability	BC	$\mathbf{C}_\odot$	$\forall U \text{LTL}(U)$ cf. [Esparza et al. 1999]
undecidability	AR	$\mathbf{C}_\odot$	$\omega\text{Reg}(A)$ [Emerson and Kahlon 2003b]
undecidability	AR	$\mathbf{C}_\odot$	$\text{LTL}(C)$ [Emerson and Kahlon 2003b]
undecidability	AR	$\mathbf{C}_\odot$	$\forall U \text{LTL}(U)$ cf. [Emerson and Kahlon 2003b]



(a) 2-ary clique graph with 3 user vertices and one controller vertex.



(b) Modified process template  $S'$  for the server.



(c) Modified process template  $C$  for the clients.

**Figure 5.2:** Clique graph and modified process templates for client/server system.

German and Sistla [1992, Theorem 3.19] proved that deadlock detection is decidable for systems using pairwise rendezvous, by a reduction to the reachability problem for VASS. To the

## 54 5. RENDEZVOUS AND BROADCAST

best of our knowledge, this is the only result on parameterized deadlock detection for the systems in this chapter.

**Specifications.** Write  $\forall U \text{LTL}(U)$  for the set of all LTL sentences of the form  $\forall i : \text{type}(i) = 2. \phi(i)$ . The interpretation of such a formula in  $\mathbf{C}_\odot(n)$  is that for every process index  $i$  of type 2 (i.e., for every “user” process) the projection of every infinite path  $\pi$  onto coordinate  $i$  satisfies  $\phi(i)$ .

Write  $\text{LTL}(C)$  for the set of all LTL formulas of the form  $\phi(1)$ . The interpretation of such a formula in  $\mathbf{C}_\odot(n)$  is that the projection of every infinite path  $\pi$  onto the unique controller process satisfies  $\phi(1)$ .

More generally, denote by  $\omega\text{Reg}(C)$  the specification languages of the form  $AL(1)$  where  $L$  is an  $\omega$ -regular language over  $\text{AP}_{\text{pr}} \times \{1\}$  (recall the controller has index 1), and denote by  $\forall U \omega\text{Reg}(U)$  the specification languages of the form  $\forall i : \text{type}(i) = 2. AL(i)$  where  $L$  is an  $\omega$ -regular language over alphabet  $\text{AP}_{\text{pr}} \times \{i\}$  (recall the users have type 2).

The set of action-based specifications, i.e.,  $\text{Reg}(A)$  and  $\omega\text{Reg}(A)$ , are defined in Section 2.4.2.

**Example 5.3 Specifications for Client/Server system.** An interesting safety property for the clients of the system is mutual exclusion, i.e., no two clients should be in state  $1_C$  at the same time. Note that none of the specification types above allows one to directly express this property. However, as noted by German and Sistla [1992], we can indirectly check for it by modifying the process template: add a state  $2_C$  to the client template that can only be entered from  $1_C$  by synchronizing on a new action  $\text{fail}$  with another client process that is also in  $1_C$ . In this way, a process can enter  $2_C$  iff we reach a state where at least two processes are in  $1_C$  at the same time. Then, mutual exclusion is expressed by the formula

$$\forall U \text{G}\neg 2_C(U),$$

where we use states of a process template as atomic propositions. Alternatively, we can express violation of the property as the regular expression over actions

$$\{a, \text{enter}, \text{leave}\}^* \text{ fail}.$$

In addition, we can specify liveness properties, e.g., every client should enter  $1_C$  infinitely often. This is expressed by the formula

$$\forall U \text{GF}1_C(U).$$

In the following two sections, we will see that the mutual exclusion property can be decided for systems with any combination of pairwise rendezvous, asynchronous rendezvous, and broadcast. In contrast, liveness properties like the one above can in general only be decided in systems that are restricted to pairwise rendezvous.

## 5.2 DECIDABILITY RESULTS

All the decidability results in this chapter follow the same idea of building a transition system which is a counter representation of the parameterized system and the parameterized specification. In Proposition 5.4 we show that for communication primitives including the ones dealt with in this chapter (namely broadcast, pairwise rendezvous, asynchronous rendezvous) this transition system is a well-structured transition system. The main point to be checked is monotonicity. Moreover, in the case of pairwise rendezvous, this transition system is a vector addition system with states (VASS).

### 5.2.1 COUNTER REPRESENTATION

The idea of using a counter representation already appears in the work of German and Sistla [1992], and relies quite heavily on the connectivity graph being a clique.

Suppose the process template of the controller has state set  $Q_C$ , the process template of the user processes has state set  $Q_U$ , and the specification is given by an (finite-state or Büchi) automaton  $A$  over the atomic propositions of the controller, and with state set  $Q_A$ . Recall from Vardi [1995] that for every LTL formula  $\varphi$  over atomic propositions AP there is a non-deterministic Büchi automaton over alphabet  $2^{\text{AP}}$ , of size  $2^{O(|\varphi|)}$ , that accepts exactly the infinite strings satisfied by  $\varphi$ .

Informally, the counter representation captures the current state of the controller (an element of  $Q_C$ ), the current state of the automaton (an element of  $Q_A$ ), and the number of user processes in each state of  $Q_U$  (a vector of non-negative integers of dimension  $|Q_U|$ ). Thus, the states of the counter representation are of the form  $(q_c, q_a, \bar{v})$ , and transitions between these states mimic global transitions in system instances.

**Definition of the Counter Representation  $M$ .** Suppose  $Q_U = \{t_1, \dots, t_k\}$ , and automaton  $A$  has initial state  $\iota_A \in Q_A$ , and accepting state set  $F \subseteq Q_A$ . Also, suppose  $s \in Q_C \times (Q_U)^n$  is a global state of a system instance with  $n$  user processes, and  $q_A$  is a state of the automaton  $A$ . Define the *abstracted state*  $\text{abs}(s, q_A)$  as the tuple  $(q_C, q_A, x_1, \dots, x_{|Q_U|})$  in  $Q_C \times Q_A \times \mathbb{N}_0^{|Q_U|}$  where  $q_C$  is  $s(1)$  (the local state of the controller process), and for  $1 \leq i \leq |Q_U|$ ,  $x_i := \#\{j : s(j) = t_i\}$ , i.e., the number of user processes that are in local state  $t_i$  in the global state  $s$ .

The *counter representation* is a transition system  $M = (Q, \Sigma, \Delta)$  defined as follows.

- The *state set*  $Q$  of  $M$  is defined as  $Q_C \times Q_A \times \mathbb{N}_0^{|Q_U|}$ , i.e., the set of all abstracted states  $\text{abs}(s, q_A)$ .
- The set of *action labels*  $\Sigma$  of  $M$  is the set  $\Sigma_{\text{int}} \cup \Sigma_{\text{sync}}$ , i.e., the action labels of the original system.
- The *transitions*  $\Delta$  of  $M$  are of the form  $\text{abs}(s, q_A) \xrightarrow{a} \text{abs}(s', q'_A)$  with  $a \in \Sigma$ , where
  - (i)  $s \xrightarrow{a} s'$  is a global transition of the original system, and

## 56 5. RENDEZVOUS AND BROADCAST

- (ii) if this transition does not involve the controller then  $q'_A = q_A$ , else  $q'_A$  can be reached from  $q_A$  in one step on input  $\{p \in AP_{pr} : p \text{ holds in state } q_C\}$ . Thus the state  $q_A$  evolves according to the label of the state  $q_C$ .

If  $q_A$  is in the accepting set  $F$  then we say that the abstracted transition *hits*  $F$ .

- The set of *initial states* consists of all states of the form  $(\iota_C, \iota_A, \bar{v})$  where  $\iota_C$  is an initial state of the controller template,  $\iota_A$  is the initial state of automaton  $A$ , and  $\bar{v}$  is an arbitrary vector whose support  $S$  (i.e., the set of coordinates with non-zero entries) is non-empty and  $s \in S$  iff  $t_s$  is an initial state of the user process template. Note that the set of initial states is computable and downward closed, and thus we satisfy the hypothesis of Theorem 3.3(2).

Note that the transition system  $M$ , although infinite, is computable, meaning that one can compute the state set and the transition relation.

**Properties of the Counter Representation  $M$ .** We elaborate on the properties of the counter representation that will later be used in decidability proofs.

**Proposition 5.4 If  $\text{card}$  is an interval then  $M$  is a WSTS.** *Let  $\leq$  be the ordering on the states of  $M$  defined by  $(q_C, q_A, \bar{v}) \leq (q'_C, q'_A, \bar{v}')$  if  $q_C = q'_C$ ,  $q_A = q'_A$  and for all  $i$ ,  $v_i \leq v'_i$ . If  $\text{card}$  is an interval, namely of the form  $[a, b]$  or  $[b, \infty)$ , then (i) the ordering  $\leq$  is monotone with respect to the transition system  $M$  and (ii) one can compute, from a basis for an upward closed set  $C$ , a basis for  $\text{Pred}(C)$ .*

**Proof Sketch.** The first item holds under very general computability conditions, namely if  $\text{card}$  is computable and the global transition relation  $\Delta$  is computable (see computability assumption 2.5.1). Indeed: if  $B$  is a basis for  $C$ , then a basis  $\text{Pred}(C)$  consists of all configurations  $\bar{v}$  such that there is some action label  $a$  and a global transition from  $\bar{v}$  to some element of  $B$  on action  $a$ .

For the second item we fix some notation. Define an operation  $+$  on states of  $M$  that have the same first two coordinates, namely,  $(q, r, \bar{v}) + (q, r, \bar{w}) := (q, r, \bar{v} + \bar{w})$  where addition is componentwise. From now on we drop all mention of the first two coordinates. The states of  $U$  are  $\{t_1, \dots, t_k\}$ . For  $i \leq k$ , write  $e_i$  for the vector  $\bar{v}$  such that  $v_i = 1$  and  $v_j = 0$  for all  $j \neq i$ .

For the second item, suppose  $\bar{v} \xrightarrow{a} \bar{w}$  is a transition of  $M$  induced by a local transition  $t_i \xrightarrow{a_{out}} t_j$ . Then we may write  $\bar{v} = e_i + \bar{c}$  and  $\bar{w} = e_j + \bar{d}$  for some vectors  $\bar{c}, \bar{d}$ . Suppose that exactly  $N$  processes synchronize with the initiating process in this transition. Then  $N \in \text{card}$  and either  $N = \max(\text{card})$ ; or  $N < \max(\text{card})$  and  $N + 1 \in \text{card}$  (because  $\text{card}$  is an interval) and, by condition (MAX), there are no more processes in the state represented by  $\bar{v}$  that can synchronize on an  $a$ -action ( $\dagger$ ).

Now, to establish monotonicity, suppose  $\bar{v} \leq \bar{y}$ . We should prove that there is an  $a$ -transition from  $\bar{y}$  to some  $\bar{z} \geq \bar{w}$ . It is enough to consider the case that  $\bar{y} = \bar{v} + e_l$  for some  $l$ .

If  $N = \max(\text{card})$  then we may define  $\bar{z} := \bar{w} + e_l$  (i.e., mimic the transition  $\bar{v} \xrightarrow{a} \bar{w}$ , and the new process in state  $t_l$  does not synchronize).

Now assume  $N < \max(\text{card})$ . There are two cases.

First case: action  $a$  is not enabled from state  $t_l$ , i.e., the local state  $t_l$  has no outgoing transition labeled  $a_{in}$ . In this case we may define  $\bar{z} := \bar{w} + e_l$  (i.e., mimic the transition  $\bar{v} \xrightarrow{a} \bar{w}$ , and the new process in state  $t_l$  does not, in fact cannot, synchronize)

Second case: action  $a$  is enabled from state  $t_l$ , say  $t_l \xrightarrow{a_{in}} t_m$  is a local transition. In this case we may define  $\bar{z} := \bar{w} + e_m$ . Indeed, in the transition from  $\bar{y}$  to  $\bar{z}$  exactly  $N + 1 \in \text{card}$  processes synchronize with the initiating process, and if it were possible that more than  $N + 1$  were available to synchronize, then more than  $N$  would have been available to synchronize in the transition from  $\bar{v}$ , contrary to  $(\dagger)$ .  $\square$

In other words, the combined system in which  $\text{card}$  is an interval and the specification is given by an automaton is a WSTS. Note that  $\text{card}$  is an interval for pairwise rendezvous ( $\{1\}$ ), asynchronous rendezvous ( $\{0, 1\}$ ), and broadcast ( $\mathbb{N}_0$ ). The next lemma allows one to combine WSTS from different communication primitives into a single WSTS.

**Lemma 5.5** *If  $M_i = (Q, \Sigma_i, \Delta_i)$  ( $i = 1, 2$ ) are two transition systems over the same state set  $Q$ , and with disjoint sets of action labels  $\Sigma_i$ , and  $\leq$  is an ordering on  $Q$ , and both  $M_1$  and  $M_2$  are WSTS with respect to  $\leq$ , then the transition system  $M_1 \cup M_2 := (Q, \Sigma_1 \cup \Sigma_2, \Delta_1 \cup \Delta_2)$  is a WSTS with respect to  $\leq$ .*

*Sketch.* First,  $M := M_1 \cup M_2$  is computable since the  $M_i$  are. Second,  $\leq$  is monotonic with respect to  $M$  since the sets of action labels of the  $M_i$ s are disjoint. Third,  $\text{Pred}_M(C) = \text{Pred}_{M_1}(C) \cup \text{Pred}_{M_2}(C)$ . From a basis for  $C$  compute a basis  $B_i$  for  $\text{Pred}_{M_i}(C)$ . Then  $B_1 \cup B_2$ , which is certainly computable, is a basis for  $\text{Pred}_M(C)$ .  $\square$

**Corollary 5.6** *Let  $M$  be the counter representation of the parameterized system  $n \mapsto \text{sys}(\overline{P}, \mathbf{G}(n), \text{card})$  where  $\overline{P}$  consists of the set of all 2-ary system templates,  $\text{card} = \{\{1\}, \{0, 1\}, \mathbb{N}_0\}$  and  $\mathbf{G}$  is the 2-ary clique graph of size  $n$ . Then  $M$  is a WSTS.*

**Aside.** If  $\text{card}$  is not an interval then  $M$  may not be a WSTS under  $\leq$ . This is due to the (MAX) item in the definition of synchronous transition (p. 8). For instance, suppose  $\text{card} = \{1, 3\}$  and the process template has transitions  $q \xrightarrow{a_{out}} q$  and  $s \xrightarrow{a_{in}} s'$ . Let  $c$  be the configuration in which one process is in state  $q$  and two processes are in state  $s$ . Then there is a synchronous transition to configuration  $c'$  in which one process is in state  $q$ , one process is in state  $s$ , and one process is in state  $q'$ . Now, consider configuration  $d$  in which one process is in state  $q$  and three processes are in state  $s$ . Then  $c \leq d$ . However the only synchronous transition from  $d$  results in a configuration  $d'$  in which one process is in state  $q$ , and none in state  $s$  — this is because the (MAX) item forces all

## 58 5. RENDEZVOUS AND BROADCAST

three processes in state  $s$  to receive the broadcast. But it is not the case that  $c' \leq d'$ , and so  $\leq$  is not monotone.

On the other hand, if synchronous transitions are defined without the (MAX) item then  $M$  is a WSTS for every card (since in the proof of Proposition 5.4 we may simply take  $\bar{z} = \bar{w} + e_l$ ).

### 5.2.2 DECIDABILITY FOR ALL THREE PRIMITIVES

We justify the decidability entry in the PR&AR&BC column of Figure 5.1.

**Theorem 5.7** [Esparza et al., 1999, Section 4] [Emerson and Kablon, 2003b, Section 4.1] If synchronization is by pairwise rendezvous, asynchronous rendezvous, and broadcast, then  $\text{PMCP}(\mathcal{P}, \mathbf{C}_\odot, \text{Reg}(A))$  is decidable.

*Proof.* Let  $A$  be an automaton for the complement of the specification  $\phi$ , and build the WSTS  $M$  as above. By Corollary 5.6,  $M$  is a WSTS. Let  $C$  be the finite set  $Q_P \times F_A \times (0, \dots, 0)$ , where  $F_A$  are the final states of the automaton  $A$ . There is a path from  $(\iota_P, \iota_A, 0, \dots, 0)$  to  $\uparrow C$  iff the specification  $\phi$  fails on some system instance of the parameterized system. Now use the fact that coverability is decidable for WSTS (Theorem 3.3).  $\square$

### 5.2.3 DECIDABILITY FOR PAIRWISE RENDEZVOUS

We discuss the decidability results in the PR column of Figure 5.1.

**Theorem 5.8** [German and Sistla, 1992, Section 3.2] If synchronization is by pairwise rendezvous then  $\text{PMCP}(\mathcal{P}, \mathbf{C}_\odot, \omega\text{Reg}(C))$  and  $\text{PMCP}(\mathcal{P}, \mathbf{C}_\odot, \text{LTL}(C))$  are decidable in EXPSPACE.

*Sketch.* Let  $A$  be a Büchi automaton over the atomic propositions of the controller. We build the WSTS  $M$  as above, except that we introduce a unique initial state  $init$ , with transitions to all states of the form  $(\iota_C, \iota_A, (0, \dots, 0))$  where  $\iota_C$  is an initial state of the controller and  $\iota_A$  is the initial state of the automaton, as well as transitions from  $(\iota_C, \iota_A, \bar{v})$  to  $(\iota_C, \iota_A, \bar{w})$  where  $\bar{w}$  is equal to  $\bar{v}$  except that at some coordinate, say  $i$ ,  $w_i = v_i + 1$  and  $t_i$  is an initial state of the user process, and from each of the states  $(\iota_C, \iota_A, \bar{v})$  there are transitions which begin the simulation of the original system. Informally,  $M$  first loads the size of the system and then simulates it. In addition, one has to make sure that global transitions with no effect on the counters are correctly simulated. This is done by splitting each transition into two: first the coordinates of the source-states are decremented by 1 and then the coordinates of the target-states are incremented by 1.

By Corollary 5.6,  $M$  is a WSTS. It is not hard to show that  $M$  is the configuration space of a vector addition system with states (VASS). The reason for this is that transitions of the original system (internal transitions and pairwise-rendezvous) translate to adding constant vectors to states of  $M$ . Actually, one has to make sure that global rendezvous transitions with no net-effect on the counters do not result in extra behaviors. This is done by splitting each VASS transition into two: first the co-ordinates of the source-states are decremented by 1 (recall that a decrement transition

can only be taken in a VASS if the co-ordinates being decremented are non-zero) and then the co-ordinates of the target-states are incremented by 1.

The problem “is there an  $n \in \mathbb{N}$  and an infinite run in the system with  $n$  users that is accepted by automaton  $A$ ” is equivalent to the problem of whether some path of the VASS  $M$  hits  $F$  infinitely often. This is a control state repeated reachability problem, which is decidable in EXPSPACE for VASS (Theorem 3.4).

Note that if the specification is given as an LTL formula  $\phi$  over the controller then the first step is to translate the formula into a non-deterministic Büchi automaton  $L$  of size  $2^{O(|\phi|)}$ . In this case the VASS is  $O(|U|)$ -dimensional and has state set of size  $|Q_C| \times 2^{O(|\phi|)}$ . Thus, by Theorem 3.4 we can solve PMCP in space polynomial in  $|Q_C| \times 2^{O(|\phi|)}$  and exponential in  $|U|$ , thus in EXPSPACE.  $\square$

By simulating a user process in the controller and using the fact that each user has the same set of neighbors, one gets the following.

**Theorem 5.9** [German and Sistla, 1992, Section 3.2] *If synchronization is by pairwise rendezvous then  $\text{PMCP}(\mathcal{P}, \mathbf{C}_\odot, \forall U \omega\text{Reg}(U))$  is decidable in EXPSPACE. In particular, the result holds for  $\forall U \text{LTL}(U)$ .*

A proof of the following theorem is hinted at by Emerson and Kahlon [2003b]. It can be proven in a similar way to Theorem 5.8.

**Theorem 5.10** [Emerson and Kahlon, 2003b, Section 5.2] *If synchronization is by pairwise rendezvous then  $\text{PMCP}(\mathcal{P}, \mathbf{C}_\odot, \omega\text{Reg}(A))$  is decidable in EXPSPACE.*

## 5.3 UNDECIDABILITY RESULTS

We discuss the undecidability results in the BC and AR columns of Figure 5.1. The rest of the undecidability results are then immediate. The basic idea is to reduce the non-halting problem of 2CMs to the PMCP by simulating the states of the 2CM in the controller and distributing the counters over the users.

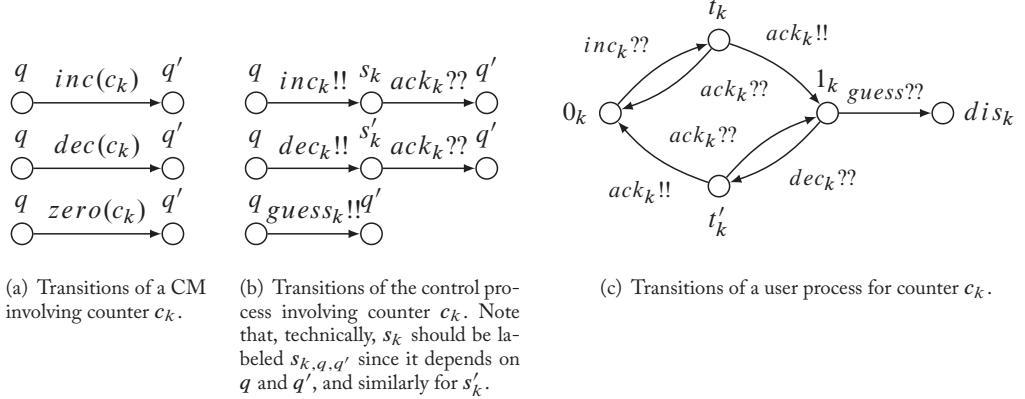
### 5.3.1 UNDECIDABILITY FOR BROADCAST

We discuss the undecidability results in the BC column of Figure 5.1. The proofs of the following results are all similar, and follow Esparza et al. [1999, Section 5].

**Theorem 5.11** *Suppose synchronization is by broadcast. Then:*

1.  $\text{PMCP}(\mathcal{P}, \mathbf{C}_\odot, \text{LTL}(C))$  is undecidable;
2.  $\text{PMCP}(\mathcal{P}, \mathbf{C}_\odot, \omega\text{Reg}(A))$  is undecidable; and
3.  $\text{PMCP}(\mathcal{P}, \mathbf{C}_\odot, \forall U \text{LTL}(U))$  is undecidable.

## 60 5. RENDEZVOUS AND BROADCAST



**Figure 5.3:** Simulating a CM using broadcasts.

**Proof idea.** We explain the first item. The second is similar, and we remark at the end how to deal with the third item. Suppose  $M$  is a 2CM with a halting location  $m$ , and without loss of generality the starting location 1 cannot be reached by a transition from any location. We build a new 2CM  $M'$  that simulates  $M$  but if the halting location  $m$  is reached then both counters are drained to zero and then there is a transition to the initial location. Thus,  $M'$  never stops and has the following property:  $M$  has a computation that enters location  $m$  if and only if  $M'$  has a computation that enters the location  $m$  infinitely often. The protocol in Figure 5.3 simulates  $M'$  using broadcasts. We will see that although such faithful simulations cannot always be ensured (simulating the zero-test is the problem), they can be ensured from some point on.

The controller process stores the current location of  $M'$ , while the users collectively store the values of counters. In a system instance of size  $n + 1$ , there are  $n$  user processes each of which is the product of two copies of the state diagram in Figure 5.3(c) (instantiated with  $k = 1, 2$ ). Thus, the state of a user process is of the form  $(x_1, x_2)$  where the  $x_k$ s are the vertices pictured in Figure 5.3(c).

- If  $x_k = 0_k$  then we say that the process'  $k$ th counter is set to zero;
- if  $x_k = 1_k$ , that its  $k$ th counter is set to one;
- if  $x_k = disk$ , that its  $k$ th counter is disabled; and
- if  $x_k = t_k$  or  $t'_k$ , that its counter is in an intermediate state.

Ideally, the number of user processes with  $x_k = 1$  is equal to the value of counter  $k$ .

In Figure 5.3 we see that increments and decrements are achieved as follows: to increment counter 1 the controller broadcasts this intention ( $inc_k!!$ ) and goes to an intermediate state  $s_1$

while the memory processes with  $k$ th counter set to zero (if any) respond and go to an intermediate state  $t_1$ ; then exactly one memory process broadcasts that it has set its counter 1 bit from zero to one, and all the other processes, including the controller process, leave their intermediate state. It is an invariant of the simulation that if there is a user whose  $k$ th counter is in an intermediate state, say  $t_k$ , then (a) every process whose  $k$ th counter is in an intermediate state is in fact in state  $t_k$ , and (b) the controller process is also in an intermediate state, i.e.,  $s_k$  (see Figure 5.3(b)). In this case the only possible next step is for some user process to broadcast an acknowledgement. Decrements are similar. On the other hand, if no process has a  $k$ th counter set to 0 (i.e., the memory storing the counter is full) and the controller broadcasts  $inc_k!!$  then the simulation hangs. Similarly, if no process has a  $k$ th counter set to 1 and the controller broadcasts  $dec_k!!$  then the simulation hangs. Thus, in an infinite run of a system instance, increments and decrements are faithfully simulated ( $\dagger$ ), i.e., every time the controller tries to increment (or decrement) a counter, it does so successfully.

Zero transitions are more subtle—the controller may broadcast a “guess counter  $k$  is zero” command. If the  $k$ th counter of every user is set to zero, then the simulation proceeds faithfully, i.e., the control of the counter machine proceeds as if the counter is zero and indeed the total value of the counter stored in the users is zero; otherwise, every user whose  $k$ th counter is set to one enters a special state  $disk$  (thus permanently disabling it from being used as storage for counter  $k$ ). In the second scenario the simulation is no longer faithful, i.e., the control of the counter machine proceeds as if the counter is zero but the value of the counter stored by the user processes is not zero. However, we will see that from some point on the simulation is faithful. To this end, let  $N_k$  be the number of processes whose  $k$ th counter is not disabled, and note that the sum  $N_1 + N_2$  is non-increasing ( $\ddagger\ddagger$ ).

Now, if the 2CM  $M'$  enters the halting location  $m$  infinitely often, then the 2CM  $M$  has a (finite) run that enters the halting location  $m$ , and thus there is a run in a large enough system instance in which the control process enters  $m$  infinitely often.

Conversely, suppose that in some system instance there is a run such that the control process enters  $m$  infinitely often. By ( $\dagger$ ), every time the controller tries to increment (or decrement) a counter, it does so successfully. By ( $\ddagger\ddagger$ ), from some point in time  $t$  onwards, the sum  $N_1 + N_2$  is constant. Thus, at all later points  $t' > t$  that the controller broadcasts “guess counter  $k$  is zero,” the  $k$ th counter of every process is equal to zero. Thus, the simulation after time  $t$  is faithful. Let  $t'' \geq t$  be a time in which the controller is in the initial location. Then the run from this time onwards witnesses that  $M'$  enters the halting state  $m$  infinitely often.

In summary, the 2CM  $M$  halts if and only if there is a system instance that satisfies  $\text{EGF}h_1$  where  $h_1$  is the indexed atomic proposition that holds if the controller is in state  $m$ .

For the third item repeat the proof above except that initially the first scheduled process sends a broadcast message declaring itself a controller (it now acts as the controller and all the other processes act as users).  $\square$

## 62 5. RENDEZVOUS AND BROADCAST

### 5.3.2 UNDECIDABILITY FOR ASYNCHRONOUS RENDEZVOUS

We discuss the undecidability results in the AR column of Figure 5.1. The proofs of the following undecidability results are similar and follow Emerson and Kahlon [2003b, Section 4].

**Theorem 5.12** *Suppose synchronization is by asynchronous rendezvous. Then:*

1.  $\text{PMCP}(\mathcal{P}, \mathbf{C}_\odot, \text{LTL}(C))$  is undecidable;
2.  $\text{PMCP}(\mathcal{P}, \mathbf{C}_\odot, \omega\text{Reg}(\mathbf{A}))$  is undecidable; and
3.  $\text{PMCP}(\mathcal{P}, \mathbf{C}_\odot, \forall U \text{ LTL}(U))$  is undecidable.

*Proof Sketch.* In the proof of Theorem 5.11 for  $\text{LTL}(C)$ , replace broadcast by asynchronous rendezvous. The only difference in behavior of the two simulations is that previously, if more than one process received a broadcast message, now, exactly one process receives the asynchronous-rendezvous message. The proof proceeds verbatim, except that one should note that at most one user is in an intermediate state at a time, and the phrase “every user whose  $k$ th counter is set to one enters a special state  $disk$ ,” should change to “at least one user whose  $k$ th counter is set to one enters a special state  $disk$ .”

The case of  $\omega\text{Reg}(\mathbf{A})$  is similar.

To extend this proof to the case  $\forall U \text{ LTL}(U)$  we proceed differently from the broadcast case. The controller’s first move should be to synchronize, once and for all, with a single user, say  $v$ , and then to move into a special state where it is no longer used. On synchronizing the user  $v$  enters a special component of its state set and plays the role of the controller. The specification is of the form “for all users  $U$ , for all runs, if  $U$  initially synchronizes with the controller then it infinitely often satisfies  $h_1$ . ”  $\square$

## 5.4 DISCUSSION

In contrast to the token-passing systems of Chapter 4, the literature has focused on the more symmetric case of clique topologies for systems with communication by broadcast, pairwise rendezvous, or asynchronous rendezvous. One reason for this is that already in uni-directional rings the PMCP for these systems is undecidable, even for safety properties. This is because such systems can simulate a binary-valued token (as in Section 4.1.2 and Section 4.2.3).

However, even in a clique topology there are many undecidability results due to the fact that in a system of size  $n$  one can simulate the run of a 2CM as long as the counters do not exceed  $n$ . Common techniques to do this simulation include:

- using broadcast to elect a unique “controller” process (if needed) that simulates the control of the 2CM, while all the others are “memory” processes;

- using pairwise rendezvous between the controller and user processes to ensure that exactly one process increments/decrements the counter; and
- using broadcast (or asynchronous rendezvous) to reduce the number of active memory processes when the controller incorrectly “guesses” that a counter is zero.

We briefly discuss how the results of this chapter are affected under the assumption that there is no controller (i.e., all processes have the same type and are in a clique). Decidability proofs are not affected. Undecidability proofs in the case of no controller typically proceed as follows: first elect a controller using broadcast, and then proceed as in the case with a controller. Although broadcast is powerful enough to elect a controller, asynchronous rendezvous only seems powerful enough to elect a temporary controller.

**Problem 5.13** Is the PMCP undecidable for systems communicating by asynchronous rendezvous, without a controller and for liveness specifications?

In summary, looking at Figure 5.1, all the entries (in the first three rows) are known to hold also for the case without controller, except for the columns AR and PR&AR and liveness specifications (i.e., the second and third rows).

Decidability results are typically proven using counter representations: since every process in a clique can communicate with every other process, it is enough to store the number of processes in every state. Pairwise rendezvous leads to a vector addition system with states, and thus certain safety and liveness properties are decidable. In contrast, broadcast and asynchronous rendezvous lead to well-structured transition systems, and thus safety is decidable, whereas it turns out that liveness is in general undecidable.

We note that the literature has only a few exact bounds on the complexity of the PMCP. We enumerate them here.

1. For broadcast protocols, [Schmitz and Schnoebelen \[2013\]](#) proved that the complexity of PMCP for coverability specifications is  $\mathbf{F}_\omega$ -complete (the class  $\mathbf{F}_\omega$  of Ackermannian problems is closed under primitive-recursive reductions).
2. For pairwise-rendezvous protocols:
  - (a) [Esparza \[2014\]](#), Section 3.3] reported that the complexity of PMCP for the coverability problem and pairwise rendezvous is EXPSPACE-complete.
  - (b) [Aminof et al. \[2014b\]](#) showed that PMCP is undecidable for pairwise rendezvous and 1-index CTL\* \ X specifications.
  - (c) [Aminof et al. \[2014b\]](#) proved that for topologies that generalize cliques and stars (but not rings) that PMCP for 1-index LTL \ X is EXPSPACE-complete (and PSPACE-complete without a controller), and that the program complexity (i.e., the formula is fixed) is EXPSPACE-complete (and in PTIME without a controller).

## 64 5. RENDEZVOUS AND BROADCAST

### 5.4.1 VARIATIONS OF THE MODEL

One variation are systems without a controller process. Here the connectivity graph is a clique, and all processes have the same type, i.e., all are user processes. The known results are summarized in Figure 5.1 (on page 52).

As a rule of thumb, systems with a unique controller are more difficult to verify than systems with only one process template, or systems with a constant number of process templates, but requiring that there is no bound on the number of processes instantiating each template. For instance, the PMCP for coverability specifications and pairwise-rendezvous synchronization is in PTIME if there is no controller (this follows immediately from German and Sistla [1992, Lemma 4.5]), while it is EXPSPACE-complete if there is a controller [Esparza, 2014, Section 3.3].

The proofs of decidability typically yield *cutoffs* that depend on both the template and the formula. For instance, for pairwise rendezvous with a controller, there is a cutoff (on the number of user processes) that is, roughly, doubly exponential in the size of the user template and the formula [German and Sistla, 1992, p. 687]. Since PMCP of liveness with broadcast is undecidable, there is no way to effectively compute cutoffs. However, the following restricted broadcast system has cutoffs for the specification language which is an extension of the universal fragment of  $\text{CTL}^*\backslash X$  by the epistemic operator  $K$ , see Kouvaros and Lomuscio [2013a]: for every two local states  $q, q'$  and broadcast action label  $a$ , it holds that  $(q, a!! , q')$  is a local transition if and only if  $(q, a?? , q')$  is a local transition. Such a broadcast transition may be called *symmetric broadcast*. Systems that communicate with symmetric broadcast and pairwise rendezvous are, in the presence of a controller, quite expressive, e.g., Kouvaros and Lomuscio [2015] modeled a swarm aggregation algorithm. In the restricted case of a single symmetric broadcast action and pairwise rendezvous (or, more generally,  $k$ -wise rendezvous for some  $k$ ), both safety and liveness are decidable [Aminof et al., 2015].

Ad hoc networks (Chapter 7) are concurrent systems in which the connectivity graphs are *general graphs* (not necessarily cliques), and the communication primitive is broadcast. Although the PMCP for general graphs and safety specifications is undecidable, one regains decidability by restricting the class of graphs (e.g., to graphs of bounded diameter).

The decidability results in the PR column of Figure 5.1 still hold if one only considers runs satisfying the following *fairness* condition: a run of a system instance is *fair* (following German and Sistla [1992, p. 680]) if every process that is enabled infinitely often is scheduled infinitely often. Here, a process is *enabled* in a global state if it can make an internal transition or synchronize with another process. However, the complexity is at least as hard as reachability for VASS.

The *Token-passing systems* that we see in Chapter 4 can be defined in terms of pairwise rendezvous, and there various fairness notions are used to get cutoff results. A similar “no-blocking” idea is used to get cutoffs for broadcast systems in the presence of a controller on cliques; see Kouvaros and Lomuscio [2013b, Definition 4.3].

## CHAPTER 6

# Guarded Protocols

In this chapter we extend the computational model of Section 2.2 with a new means of coordination. Until now processes coordinated with synchronized transitions and we have used transition labels to express which transitions should be taken together. In this chapter, we introduce guards, which are conditions on the global state and determine whether a specific local transition may be fired. We review results on systems with guards that contain quantifiers over all processes except the one evaluating the guard. Hence, the current state of other processes may restrict the control flow of a process. In contrast to synchronized transitions that, intuitively, link transitions of different processes, guards link transitions to the state of other processes.

Most of the models we discuss here contain two types of process templates, a single coordinator  $C$ , and user processes  $U$ . The systems are parameterized in the number of processes of type  $U$ .

The (un)decidability results we review here depend on the form of the guards as well as on the logic fragments used to formalize the specifications. Figure 6.1 and Table 6.1 give an overview of the results that we discuss in this chapter. In addition, in Section 6.7 we review restrictions of guards which ensure decidability, and are used to model cache-coherence protocols [Emerson and Kahlon, 2003a,c].

The undecidability results are proven by reduction to the non-halting problem of two-counter machines (cf. Section 3.1). The decidability proofs are based on the cutoff results by Emerson and Kahlon [2000], who construct runs in a small cutoff system from runs of bigger systems and vice versa.

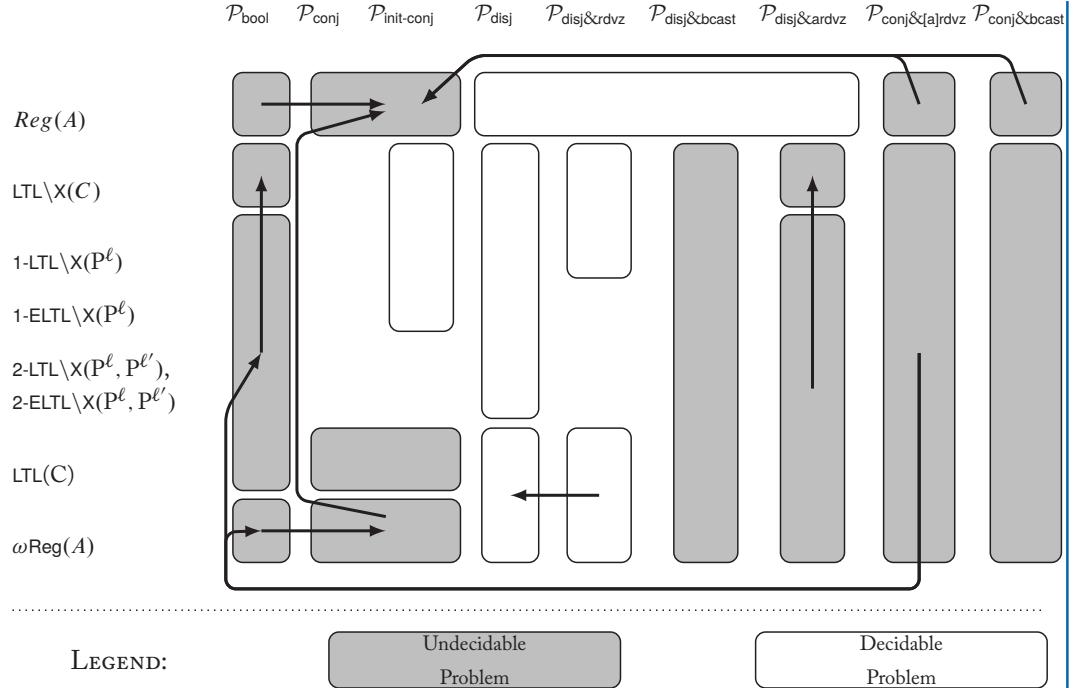
### 6.1 MOTIVATING EXAMPLE

Consider a multi-threaded program, composed of  $n$  threads that are concurrently accessing or modifying a shared doubly linked list. Figure 6.2 shows an example of such a list. As is typical, we assume that a single update of a pointer in the list is atomic, that is, a thread reads an old pointer value, if another thread is currently writing to that pointer. However, our list is a non-atomic data structure, that is, while one process is updating several pointers in the list, the other processes may access the list. Figure 6.3 shows the list in an inconsistent state, when one thread started to delete the second element. To avoid such scenarios, we need a protocol that ensures that no other thread accesses the list in an inconsistent state. Such a protocol is usually called “multiple readers/single writer protocol.”

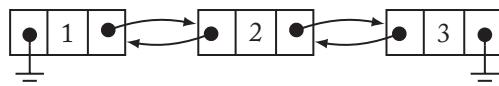
## 66 6. GUARDED PROTOCOLS

**Table 6.1:** Decidability results for Guarded Protocols and their sources, separated into systems with boolean guards ( $\mathcal{P}_{\text{bool}}$ ), conjunctive guards ( $\mathcal{P}_{\text{conj}}$ ), init-conjunctive guards ( $\mathcal{P}_{\text{init-conj}}$ ), and disjunctive guards ( $\mathcal{P}_{\text{disj}}$ )

Result	Protocol	Graph	Specification	References
undecidability	$\mathcal{P}_{\text{bool}}$	clique ( <b>C</b> )	$\text{LTL}(C)$	[Emerson and Namjoshi 1996] (proof idea)
undecidability	$\mathcal{P}_{\text{init-conj}}$	clique ( <b>C</b> )	$\text{LTL}(C)$ , $\text{Reg}(A)$ , $\omega\text{Reg}(A)$	[Emerson and Kahlon 2003b]
a cutoff for $\ell \in \{1, 2\}$	$\mathcal{P}_{\text{init-conj}}[2]$	clique ( <b>C</b> )	$1\text{-LTL}\backslash X(P^\ell)$ , $1\text{-ELTL}\backslash X(P^\ell)$ , $\text{LTL}\backslash X(C)$	[Emerson and Kahlon 2000]
a trivial cutoff for deadlock- free instances	$\mathcal{P}_{\text{init-conj}}$	clique ( <b>C</b> )	as above	[Emerson and Kahlon 2000]
a trivial cutoff for finite-path properties	$\mathcal{P}_{\text{init-conj}}$	clique ( <b>C</b> )	as above	[Emerson and Kahlon 2000]
a cutoff $\ell \in \{1, 2\}$	$\mathcal{P}_{\text{disj}}[2]$	clique ( <b>C</b> )	$1\text{-LTL}\backslash X(P^\ell)$ , $1\text{-ELTL}\backslash X(P^\ell)$ , $\text{LTL}\backslash X(C)$	[Emerson and Kahlon 2000]
a cutoff	$\mathcal{P}_{\text{disj}}$	clique ( <b>C</b> )	$2\text{-LTL}\backslash X(P^\ell, P^{\ell'})$ , $2\text{-ELTL}\backslash X(P^\ell, P^{\ell'})$ , $\text{LTL}\backslash X(C)$	[Emerson and Kahlon 2000]
decidability	$\mathcal{P}_{\text{disj}}$	clique ( <b>C</b> )	$\text{Reg}(A)$ , $\omega\text{Reg}(A)$	[Emerson and Kahlon 2003b]
decidability	$\mathcal{P}_{\text{disj}\&\text{crdz}}$	clique ( <b>C</b> )	$\text{Reg}(A)$ , $\omega\text{Reg}(A)$ , $\text{LTL}(C)$ , $1\text{-LTL}\backslash X(P^\ell)$	[Emerson and Kahlon 2003b]
decidability	$\mathcal{P}_{\text{disj}\&\text{ardvz}}$	clique ( <b>C</b> )	$\text{Reg}(A)$	[Emerson and Kahlon 2003b]
undecidability	$\mathcal{P}_{\text{disj}\&\text{ardvz}}$	clique ( <b>C</b> )	$\omega\text{Reg}(A)$	[Emerson and Kahlon 2003b]
decidability	$\mathcal{P}_{\text{disj}\&\text{cbcast}}$	clique ( <b>C</b> )	$\text{Reg}(A)$	[Emerson and Namjoshi 1998], [Emerson and Kahlon 2003b]
undecidability	$\mathcal{P}_{\text{disj}\&\text{cbcast}}$	clique ( <b>C</b> )	$\omega\text{Reg}(A)$	[Esparza et al. 1999], [Emerson and Kahlon 2003b]
undecidability	$\mathcal{P}_{\text{conj}\&\text{disj}}$	clique ( <b>C</b> )	all above	[Emerson and Kahlon 2003b],
undecidability	$\mathcal{P}_{\text{conj}\&\text{qfa}\text{rdz}}$	clique ( <b>C</b> )		[Esparza et al. 1999]
undecidability	$\mathcal{P}_{\text{conj}\&\text{cbcast}}$	clique ( <b>C</b> )		



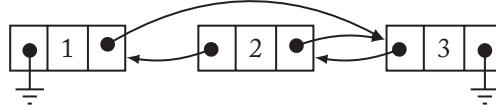
**Figure 6.1:** Decidability results for guarded protocols. An arrow from box  $A$  to box  $B$  means that the result of  $A$  follows from the result, or variation of the proof, of  $B$ . Blank entries are those not covered in the literature. The guarded protocols are defined over process templates  $P^1, \dots, P^d$ , and  $1 \leq \ell, \ell' \leq d$ . In case of  $\text{LTL}\backslash X(C)$  and  $\text{LTL}(C)$ , we assume that template  $C = P^1$ , and there is only one process of type  $C$ .



**Figure 6.2:** A doubly linked list shared by  $n$  threads.

To prevent several threads from simultaneously modifying the list or accessing the list that is being modified by another thread, we introduce a multiple readers/single writer protocol that is shown in Figure 6.4. The guards  $\phi_1, \dots, \phi_4$  are defined in Equations 6.1–6.4. The formal syntax and semantics of guards follows in Section 6.2.

## 68 6. GUARDED PROTOCOLS



**Figure 6.3:** An inconsistent state of the list when one thread is deleting the second element.

$$\phi_1 \equiv [\forall \text{ other } j] \text{ neutral}_j \vee \text{try-write}_j \vee \text{try-read}_j \quad (6.1)$$

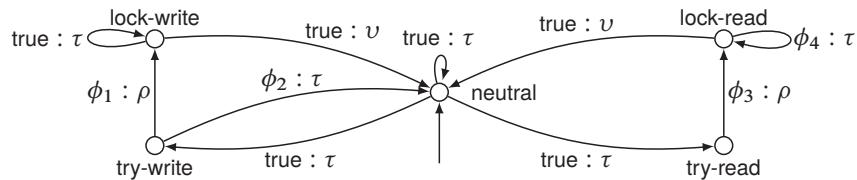
$$\phi_2 \equiv [\exists \text{ other } j] \text{ lock-write}_j \quad (6.2)$$

$$\phi_3 \equiv [\forall \text{ other } j] \text{ neutral}_j \vee \text{try-read}_j \vee \text{lock-read}_j \quad (6.3)$$

$$\phi_4 \equiv [\forall \text{ other } j] \text{ neutral}_j \vee \text{try-read}_j \vee \text{lock-read}_j \quad (6.4)$$

When a thread does not have to access or modify the list, it remains in the neutral state. When a thread has to modify the list, it changes its state to try-write. The thread can lock the list for exclusive access by changing its state to lock-write, provided that no other thread resides in the state lock-write, or in the state lock-read. The thread has an option to go back to the state neutral, if another thread has already entered the state lock-write. When the thread enters the state lock-write, it can iterate over the list and modify its elements; as soon as it finishes with the modifications, the thread goes back to the neutral state and thus allows other threads to access the list.

When a thread is going to iterate over the elements of the list without modifying it, the thread changes its state to try-read. The thread can lock the list for shared access by changing its state to lock-read, provided that there is no other thread in state try-write or lock-write, that is, no other thread has exclusively locked the list, or is trying to do so. Multiple processes can reside in the state lock-read and therefore can concurrently read the contents of the list. A thread must leave the state lock-read, as soon as it has finished reading its contents, or another thread has entered the state try-write.



**Figure 6.4:** A process template modeling a multiple readers/single writer protocol. The circles represent local states labeled with state names, e.g., neutral. The edges represent transitions labeled with guards and actions, e.g.,  $\phi_1 : \rho$  for a guard  $\phi_1$  and an action  $\rho$ .

The protocol enforces the readers to have a lower priority than the writer. Moreover, the readers are preempted by a thread that is going to modify the list. When writes happen rarely,

multiple readers can efficiently access the list. On the other hand, when the writes occur frequently, the readers might starve, without ever accessing the list.

Note that, in order to lock the list, a thread has to make sure that there are no other threads in certain states. Although the exclusive access to the list can be enforced with a token-passing protocol (see Section 4), it is not easy to model the locking mechanism by multiple readers with token passing. The protocol can be modeled using broadcasts (see Section 5) by introducing auxiliary states and enforcing the threads to “listen” to the broadcasts by the threads locking and unlocking the list. This modeling, however, is closer to hardware rather than to multi-threaded programs, because the threads have to perform steps synchronously. Guarded protocols is a natural computational model for our example.

## 6.2 SYSTEM MODEL

To formalize guarded protocols, we slightly specialize the notion of process template, as we require specific atomic propositions to define guards. From this specialized process template, we obtain a system template in the usual way. Then, for a given system template we can define guards. In this way, we collect all ingredients to define guarded system instances: we associate to each *local* transition a guard, and restrict the *global* transition relation in a way that only admits global transitions that are built from local transitions whose guards evaluate to true in the current global state.

**Identity-labeled process templates.** We specialize the definition of a process template from Section 2.2 in that we identify atomic propositions with the local states, and restrict the label of a local state to be the singleton containing the state itself. That is, given  $d$  disjoint sets of local states  $Q^1, \dots, Q^d$ , we specialize  $AP_{pr}$  to be  $Q^1 \cup \dots \cup Q^d$ . An *identity-labeled process template*  $P^\ell$  is a process template  $P^\ell = (Q^\ell, Q_0^\ell, \Sigma_{pr}, \delta^\ell, \lambda^\ell)$ , where for all  $q \in Q^\ell$  it holds that  $\lambda^\ell(q) = \{q\}$ . Then, a tuple  $(P^1, \dots, P^d)$  of identity-labeled process templates is an *identity-labeled  $d$ -ary system template*.

**Example 6.1** In our example in Figure 6.4, there is one process template  $P^1$  with the set of local states  $Q^1 = \{\text{neutral}, \text{try-read}, \text{try-write}, \text{lock-read}, \text{lock-write}\}$ . The set of atomic propositions  $AP_{pr}$  equals to the set of local states  $Q^1$ . The set of actions is  $\Sigma_{pr} = \Sigma_{int} = \{\rho, \nu, \tau\}$ . Every local state is labeled with itself, e.g.,  $\lambda^1(\text{neutral}) = \{\text{neutral}\}$ .

**Guards.** We start by restricting the syntax of formulas that will be used in guards. In what follows, we will write  $\phi(j)$  to denote a boolean formula over  $AP_{pr} \times \{j\}$ . In other words,  $\phi(j)$  is a boolean formula over atomic propositions  $AP_{pr}$  of a process, indexed with a free index variable  $j$ . When  $j$  is instantiated with a constant  $w \in V(G)$ , the formula  $\phi(w)$  becomes a boolean formula over  $AP_{sys}$ . Given  $w \in V(G)$ , the formula  $\phi(w)$  is evaluated inductively at a

## 70 6. GUARDED PROTOCOLS

state  $s = (q_1, \dots, q_{|V|}) \in S$  of a system instance  $\overline{P}^G$  as follows: for  $p \in AP_{pr}$ , it holds that  $(\overline{P}^G, s) \models (p, w)$  if and only if  $(p, w) \in \Lambda(s)$ ; The boolean connectives are interpreted as usual.

We define disjunctive, conjunctive, and boolean guards as follows.

- Let  $\phi(j)$  be a disjunction over  $AP_{pr} \times \{j\}$ . Then,
  - $[\forall \text{ other } j] \phi(j)$  is a *conjunctive guard*, and
  - $[\exists \text{ other } j] \phi(j)$  is a *disjunctive guard*.
- Conjunctive and disjunctive guards are *boolean guards*. If  $f$  and  $g$  are boolean guards, then  $\neg f$ ,  $f \vee g$ , and  $f \wedge g$  are boolean guards as well.

To simplify notation, we write  $p_j$  to denote a proposition  $(p, j) \in AP_{pr} \times \{j\}$ .

**Example 6.2** In the protocol in Figure 6.4, the guard  $\phi_1 \equiv [\forall \text{ other } j] \text{ neutral}_j \vee \text{try-write}_j \vee \text{try-read}_j$  is an example of a conjunctive guard, whereas the guard  $\phi_2 \equiv [\exists \text{ other } j] \text{ lock-write}_j$  is an example of a disjunctive guard.

One can see from the definition that the quantifiers  $[\forall \text{ other } j]$  and  $[\exists \text{ other } j]$  are never nested, but the boolean connectives can introduce subformulas with several quantifiers.

Intuitively, if one fixes a process  $v$ , then  $[\forall \text{ other } j] \phi(j)$  means that  $\phi$  should evaluate to true on *all processes* other than  $v$ , and  $[\exists \text{ other } j] \phi(j)$  means that  $\phi$  should evaluate to true on *at least one* process other than  $v$ . Formally, given a system instance  $\overline{P}^G$ , its global state  $s$ , a process  $v \in V(G)$ , and a guard  $f$ , we evaluate  $f$  in  $s$  with respect to  $v$  (in symbols,  $(\overline{P}^G, s) \models_v f$ ) as follows.

- If  $f$  has the form  $[\forall \text{ other } j] \phi(j)$ , then  $(\overline{P}^G, s) \models_v f$  if and only if for *every* process  $w \in V(G) \setminus \{v\}$  it holds  $(\overline{P}^G, s) \models \phi(w)$ , where  $\phi(w)$  is obtained from  $\phi(j)$  by substituting all instances of  $j$  with  $w$ .
- Similarly, if  $f$  has the form  $[\exists \text{ other } j] \phi(j)$ , then  $(\overline{P}^G, s) \models_v f$  if and only if there *exists* a process  $w \in V(G) \setminus \{v\}$  with the property  $(\overline{P}^G, s) \models \phi(w)$ .
- If  $f$  is constructed using boolean connectives  $\neg$ ,  $\vee$ , and  $\wedge$ , then  $(\overline{P}^G, s) \models_v f$  is defined as usual via evaluation of the subformulas of  $f$ .

Consider a boolean formula  $\text{all\_states}(i) = \bigvee_{q \in Q^1 \cup \dots \cup Q^d} (q, i)$ . Observe that the formula evaluates to true on every local state  $q \in Q^1 \cup \dots \cup Q^d$  of a process  $i$ . Using  $\text{all\_states}(i)$ , one can construct the conjunctive guard “[ $\forall \text{ other } i$ ]  $\text{all\_states}(i)$ ” and the disjunctive guard “[ $\exists \text{ other } i$ ]  $\text{all\_states}(i)$ ” that both evaluate to true on *every* global state. We abbreviate either of these guards with true: the form  $[\forall \text{ other } i] \text{ all\_states}(i)$  is used when the class of protocols is restricted to the protocols with conjunctive guards; the form  $[\exists \text{ other } i] \text{ all\_states}(i)$  is used when the class of protocols is restricted to the protocols with disjunctive guards.

**Guarded protocols.** A guarded protocol is an identity-labeled  $d$ -ary system template equipped with an assignment of a guard to each transition. Formally, a *guarded protocol* is a pair  $((P^1, \dots, P^d), \text{gd})$ , where  $(P^1, \dots, P^d)$  is an identity-labeled  $d$ -ary system template with  $P^\ell = (Q^\ell, Q_0^\ell, \Sigma_{pr}, \delta^\ell, \lambda^\ell)$ , and  $\text{gd}$  maps every transition from  $\bigcup_{1 \leq \ell \leq d} \delta^\ell$  to a guard.

**Example 6.3** The protocol in Figure 6.4 has 10 transitions, each labeled with a guard. For instance, the transition  $t = (\text{try-read}, \rho, \text{lock-read})$  is labeled with the guard  $\phi_3$ , or, formally,  $\text{gd}(t) = \phi_3$ .

When we consider guarded protocols with synchronization actions, a standard assumption is that receive actions are always enabled. More formally, for every  $\text{in}_a \in AP_{pr}$  and every  $t = (q, \text{in}_a, q') \in \delta^\ell$ , we require that  $\text{gd}(t) = \text{true}$ . For instance, if  $\text{in}_a$  is a broadcast receive  $a??$ , then every transition with action label  $a??$  is guarded by true.

**Connectivity graphs.** In this chapter we restrict  $d$ -ary connectivity graphs to cliques. Let  $\mathbf{C} : \mathbb{N}^d \mapsto \mathbf{C}(\mathbf{n})$  be a sequence of graphs that associates a  $d$ -ary connectivity graph with a size vector  $\mathbf{n} = (n_1, \dots, n_d) \in \mathbb{N}^d$ . We require that each  $\mathbf{C}(\mathbf{n})$  is a clique of size  $n$ , i.e., if  $\mathbf{C}(\mathbf{n}) = (V, E, \text{type})$ , then  $E = V \times V$  and for all  $\ell \in [d]$ , it holds  $|\{v \in V : \text{type}(v) = \ell\}| = n_\ell$ . We also introduce notation  $\mathbf{I}(\mathbf{n})$  for the set of indices  $\{i : 1 \leq i \leq n_1 + \dots + n_d\}$ .

**Example 6.4** The protocol in Figure 6.4 has one process template, that is,  $d = 1$ . Then,  $\mathbf{C}(\mathbf{n}) : \mathbb{N} \rightarrow \mathbb{N}$  is a function that for each  $n \geq 1$ , returns a clique of size  $n$ , which corresponds to a system of  $n$  processes created from the template  $P^1$ .

**Guarded system instances.** We construct a guarded system instance  $\overline{P}^{\mathbf{C}(\mathbf{n})}$  of a guarded protocol  $((P^1, \dots, P^d), \text{gd})$  by restricting the global transition relation. In particular, we restrict internal and synchronous transitions—defined in Section 2.2—to guarded internal transitions and guarded synchronous transitions.

A *guarded internal transition* of  $\overline{P}^{\mathbf{C}(\mathbf{n})}$  is a triple  $(s, a, s') \in S \times \Sigma_{int} \times S$  for which there exists a process index  $v \in V$  fulfilling the conditions (INT-STEP), (INT-FRAME), and

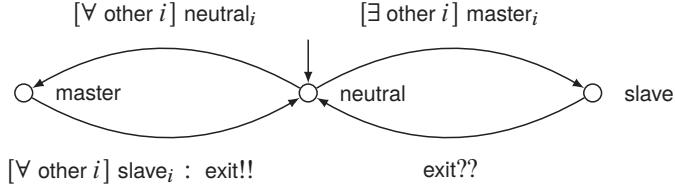
(INT-GUARD) The guard of the process transition  $t_v = (s(v), a, s'(v))$  is satisfied in the global state  $s$ , that is,  $(\overline{P}^{\mathbf{C}(\mathbf{n})}, s) \models_v \text{gd}(t_v)$ .

A *guarded synchronous transition* is  $(s, a, s') \in S \times \Sigma_{sync} \times S$  for which there exists a process index  $v \in V$  and a set  $\mathcal{I} \subseteq \{w \in V : E(v, w)\}$  of recipients of  $v$  in  $\mathbf{C}(\mathbf{n})$  satisfying (CARD), (STEP), ( $\mathcal{I}$ -STEP), (FRAME), (MAX), and the following condition:

(GUARD) The guard of the process transition  $t_v = (s(v), out_a, s'(v))$  of process  $v$  is satisfied in the global state  $s$ , that is,  $(\overline{P}^{\mathbf{C}(\mathbf{n})}, s) \models_v \text{gd}(t_v)$ . (Recall that input actions are labeled with true.)

## 72 6. GUARDED PROTOCOLS

Finally, the global transition relation  $\Delta$  of  $\overline{P}^{\mathbf{C}(n)}$  consists only of guarded internal and synchronous transitions.



**Figure 6.5:** A barrier synchronization protocol. The first process enters the barrier as a master, and all other processes enter the barrier later as slaves. Once all processes gather in the states slave and master, the master broadcasts the action  $\text{exit}!!$  and all processes go to the state neutral synchronously. The guards of unlabeled edges are equal to true, and the actions of unlabeled edges are equal to  $\tau$ .

**Example 6.5** Consider a system instance  $P_{RW}^{\mathbf{C}(3)}$  of three processes following the template  $P_{RW}$  that is shown in Figure 6.4. Let  $s$  and  $t$  be the states (neutral, try-read, try-read) and (neutral, lock-read, try-read), respectively. Then, the transition  $(s, \tau, t)$  is a guarded internal transition of the system instance  $(P_{RW})^{\mathbf{C}(3)}$ . Indeed, the condition (INT-GUARD) is satisfied, as the guard  $\phi_3$  holds true in the global state  $s$  and therefore process 3 can perform its internal transition (try-read,  $\tau$ , lock-read).

**Example 6.6** Consider the process template  $P_B$  shown in Figure 6.5. The processes following the template  $P_B$  implement a barrier synchronization protocol: the processes enter the barrier and, as soon as all processes gather at the barrier, leave it synchronously. Let  $s$  and  $t$  be the states (master, slave, slave) and (neutral, neutral, neutral), respectively. Then, the transition  $(s, \tau, t)$  is a guarded transition of the system instance  $P_B^{\mathbf{C}(3)}$ . Indeed, the condition (GUARD) is satisfied, as the guard  $[\forall \text{other } i] \text{ slave}_i$  holds true for process 1 in the global state  $s$ , and therefore process 1 performs the transition (master,  $x!!$ , neutral). Moreover, according to the conditions (CARD) and (MAX), processes 2 and 3 must perform the transition (slave,  $x??$ , neutral).

**Deadlocks.** In this chapter, we require that the transition relation of a system instance is total. If there is a deadlock in a system, then the deadlocked run is extended to an infinite one with the deadlock state repeated at the end (cf. discussion in Section 2.2.2).

In contrast to most other systems considered in the literature, the parameterized deadlock detection problem for guarded protocols has been studied extensively, and decidability results for PMCP often extend to parameterized deadlock detection. This is the case for Theorems 6.22, 6.25, and 6.30 in this chapter.

### 6.2.1 CLASSES OF GUARDED PROTOCOLS

There are several important classes of guarded protocols in the literature. The following classes restrict the form of guards, and assume that only *internal transitions* occur, that is,  $\Sigma_{sync} = \emptyset$ :

$\mathcal{P}_{\text{bool}}$ : guarded protocols with boolean guards [Emerson and Namjoshi, 1996];

$\mathcal{P}_{\text{disj}}$ : guarded protocols using only disjunctive guards [ $\exists$  other  $j$ ]  $\phi(j)$  [Emerson and Kahlon, 2000];

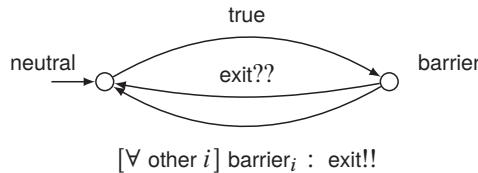
$\mathcal{P}_{\text{conj}}$ : guarded protocols using only conjunctive guards [ $\forall$  other  $j$ ]  $\phi(j)$  [Emerson and Kahlon, 2003b];

$\mathcal{P}_{\text{init-conj}}$ : the subclass of  $\mathcal{P}_{\text{conj}}$  with the following restrictions [Emerson and Kahlon, 2000]:

- each process template  $P^\ell$  has  $init^\ell$  as the only initial state, i.e.,  $Q_0^\ell = \{init^\ell\}$ ; and
- every guard is satisfiable by the initial states, i.e., the guard can be written in the form  $[\forall \text{ other } j] (init^1 \vee \dots \vee init^d \vee \psi(j))$ .

Although both  $\mathcal{P}_{\text{conj}}$  and  $\mathcal{P}_{\text{init-conj}}$  are usually called protocols with conjunctive guards, we differentiate  $\mathcal{P}_{\text{init-conj}}$  from  $\mathcal{P}_{\text{conj}}$ , as they differ in decidability. In particular, the former have remarkable decidability results.

**Example 6.7** Consider the guarded protocol  $(P, gd)$  shown in Figure 6.4. As it contains both conjunctive and disjunctive guards, the protocol belongs to the class  $\mathcal{P}_{\text{bool}}$ . Let  $(P', gd')$  be the protocol obtained from  $(P, gd)$  by removing the transition that is labeled with  $\phi_2$ . Then, the resulting protocol belongs to the class  $\mathcal{P}_{\text{conj}}$ . Moreover, as the guards  $\phi_1, \phi_3$ , and  $\phi_4$  contain the initial state neutral, the guarded protocol  $(P', gd')$  belongs to the class  $\mathcal{P}_{\text{init-conj}}$ .



**Figure 6.6:** A simplified version of the barrier synchronization protocol shown in Figure 6.5. Every process may enter the state barrier. Once all processes gather in the state barrier, one of them broadcasts the action  $exit!!$  and all processes go to the state neutral synchronously. The guards of unlabeled edges are equal to true, and the actions of unlabeled edges are equal to  $\tau$ .

Further, when synchronous transitions are allowed, we consider the classes of the form  $\mathcal{P}_{g\&z}$ . Each such a class contains guards of type  $g \in \{\text{conj}, \text{disj}\}$ , internal guarded transitions, and guarded transitions with the synchronization primitives of type  $z \in \{\text{bcast}, \text{rdv}, \text{ardv}\}$ , where  $\text{conj}$

## 74 6. GUARDED PROTOCOLS

and  $\text{disj}$  stand for conjunctive and disjunctive guards, while  $\text{bcast}$ ,  $\text{rdv}$ , and  $\text{ardv}$  denote broadcast, pairwise rendezvous, and asynchronous rendezvous, respectively (Section 2.2.1). For instance,  $\mathcal{P}_{\text{disj}\&\text{bcast}}$  and other combinations of disjunctive guards with synchronization primitives have been investigated by Emerson and Kahlon [2003a,b,c]. We revisit the results on combinations of guards with synchronization primitives in Section 6.6.

**Example 6.8** The guarded protocol shown in Figure 6.5 belongs to the class  $\mathcal{P}_{\text{bool}\&\text{bcast}}$ . Figure 6.6 shows a simplified version of the barrier protocol that belongs to the class  $\mathcal{P}_{\text{conj}\&\text{bcast}}$ .

In what follows, by  $\mathcal{P}[\mathbf{d}]$  we denote guarded protocols from class  $\mathcal{P}$  with exactly  $\mathbf{d}$  process templates.

One can find two syntactically different forms of disjunctive guards in the literature. The first paper on guarded systems [Emerson and Namjoshi, 1996]<sup>1</sup> introduces disjunctive guards as expressions  $[\exists \text{ other } j] \phi(j)$ , where  $\phi(j)$  is a boolean formula over  $\text{AP}_{\text{pr}} \times \{j\}$ . Recall that in this book we consider  $\phi(j)$  only in restricted form, namely,  $\phi(j)$  is a disjunction over propositions from  $\text{AP}_{\text{pr}} \times \{j\}$ ; cf. Emerson and Kahlon [2000, 2003a,b,c]. In fact, both definitions of guarded systems have the same expressive power. Indeed, every global state of a system instance is labeled with exactly one local state of every process, and thus it is easy to prove the following observation.

**Observation 6.9** Consider a guarded  $\mathbf{d}$ -ary system template  $(P^1, \dots, P^d)$ . Let  $\phi(j)$  be a boolean formula over  $\text{AP}_{\text{pr}} \times \{j\}$ . Then there exists a formula  $\psi(j) = p_1 \vee \dots \vee p_k$  with  $\{p_1, \dots, p_k\} \subseteq \text{AP}_{\text{pr}} \times \{j\}$  with the following property:

For every clique  $\mathbf{C}(n)$ , for every global state  $s$  of system instance  $\overline{P}^{\mathbf{C}(n)}$ , and for every process  $v \in V(\mathbf{C}(n))$ , it holds  $\overline{P}^{\mathbf{C}(n)}, s \models \phi(v)$  if and only if  $\overline{P}^{\mathbf{C}(n)}, s \models \psi(v)$  is true.

### 6.2.2 SPECIFICATIONS

Indexed formulas on guarded protocols restrict the scope of indexed variables to a single process template. Recall that the notation  $\forall i : \text{type}(i) = \ell. \phi(i)$  denotes that  $i$  ranges only over the indices of processes instantiated from template  $P^\ell$ .

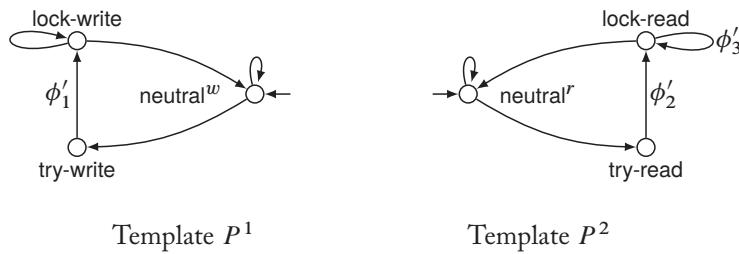
In what follows, we consider several classes of specifications met in the literature.

- Classes  $1\text{-LTL}\backslash X(P^\ell)$  and  $1\text{-ELTL}\backslash X(P^\ell)$  contain *one-index* formulas over the atomic propositions of processes having the same fixed process template  $P^\ell$ . If  $\phi(i)$  is an  $\{\text{A}, \text{E}\}$ -free  $1\text{-CTL}^*\backslash X$  path formula over  $Q^\ell \times \{i\}$ , then “ $\forall i : \text{type}(i) = \ell. \text{A}\phi(i)$ ” and “ $\forall i : \text{type}(i) = \ell. \text{E}\phi(i)$ ” are  $1\text{-LTL}\backslash X(P^\ell)$  and  $1\text{-ELTL}\backslash X(P^\ell)$  formulas, respectively.<sup>2</sup>

<sup>1</sup>We do not include the results by Emerson and Namjoshi [1996] in this book, as they consider synchronous systems, that is, the systems where all processes make a step at once.

<sup>2</sup>In the nonparameterized case, one can reduce the problem of checking an ELTL-formula  $\text{E}\neg\psi$  on a Kripke structure  $M$  to  $M \not\models \text{A}\psi$ . This, however, does not work in the parameterized case: a straightforward negation of PMCP for  $1\text{-LTL}\backslash X(P^\ell)$ , that is,  $\neg(\forall n \geq (1, \dots, 1). \overline{P}^n \models \forall i : \text{type}(i) = \ell. \text{A}\phi(i))$  is not equivalent to PMCP for  $1\text{-ELTL}\backslash X(P^\ell)$ , that is,  $\forall n \geq (1, \dots, 1). \overline{P}^n \models \forall i : \text{type}(i) = \ell. \text{E}\neg\phi(i)$ .

- Classes  $\text{LTL}(C)$  and  $\text{LTL} \setminus X(C)$  contain LTL and  $\text{LTL} \setminus X$  formulas, respectively, over the states of the sole controller process of type 1. This only process is created from a process template  $P^1$  and is usually called the *controller*, while the other processes are created from a process template  $P^2$  and are usually called the *users*. In what follows, we will use  $C$  to denote the controller process template  $P^1$  and  $U$  to denote the user process template  $P^2$ . One can see  $\text{LTL} \setminus X(C)$  as a subclass of  $1\text{-LTL} \setminus X(C)$ , where the process type is compared to 1, that is, a specification is of the form  $\forall i: \text{type}(i) = 1. \varphi(i)$ , and there is only one process of type 1. The properties from these classes are also discussed in Section 5.
- Classes  $2\text{-LTL} \setminus X(P^\ell, P'^\ell)$  and  $2\text{-ELTL} \setminus X(P^\ell, P'^\ell)$  are similar to  $1\text{-LTL} \setminus X(P^\ell)$  and  $1\text{-LTL} \setminus X(P^\ell)$ , but they contain *two-index* formulas over the states of processes created from process templates  $P^\ell$  and  $P'^\ell$ . That is, the formulas have the form
  - $\forall i: \text{type}(i) = \ell. (\forall j: \text{type}(j) = \ell'. A\phi(i, j))$  or
  - $\forall i: \text{type}(i) = \ell. (\forall j: \text{type}(j) = \ell'. E\phi(i, j))$ .
- Classes  $\text{Reg}(A)$  and  $\omega\text{Reg}(A)$  are regular and  $\omega$ -regular languages over actions  $\Sigma_{int}$  as defined in Section 2.4.2.



**Figure 6.7:** Another modeling of multiple readers/single writer protocol using two process templates: template  $P^1$  models a writer in the system; template  $P^2$  models a reader in the system. The guards  $\phi'_1$ ,  $\phi'_2$ , and  $\phi'_3$  are defined in Equations 6.5–6.7. The guards of unlabeled edges are equal to true, and the actions of unlabeled edges are equal to  $\tau$ .

**Example 6.10** Consider the guarded protocol shown in Figure 6.4.

The specification  $\forall i: \text{type}(i) = 1. \text{AG}(\text{try-write}_i \rightarrow \text{F(lock-write}_i \vee \text{neutral}_i))$  states that in every execution, every process that is trying to obtain the write lock will eventually do so, or will fall back to the state *neutral*. This specification belongs to the class  $1\text{-LTL} \setminus X(P^1)$ .

The specification  $\forall i: \text{type}(i) = 1. \text{EG}(\text{try-write}_i \rightarrow \text{F lock-write}_i)$  states that there is at least one execution, where every process that is trying to obtain the write lock will do so. This specification belongs to the class  $1\text{-ELTL} \setminus X(P^1)$ . The specification  $(\rho \tau^* \nu)^*$  states that locking action  $\rho$

## 76 6. GUARDED PROTOCOLS

and unlocking action  $v$  alternate; the specification belongs to the class  $\text{Reg}(A)$ . Similarly, the specification  $(\rho \tau^* v)^\omega$  states that locking and unlocking actions alternate infinitely often; it belongs to the class  $\omega\text{Reg}(A)$ .

**Example 6.11** Consider the guarded protocol shown in Figure 6.7. The guards  $\phi'_1$ ,  $\phi'_2$ , and  $\phi'_3$  are defined as follows:

$$\phi'_1 \equiv [\forall \text{ other } j] \text{neutral}_j^r \vee \text{neutral}_j^w \vee \text{try-read}_j \vee \text{try-write}_j \quad (6.5)$$

$$\phi'_2 \equiv [\forall \text{ other } j] \text{neutral}_j^r \vee \text{neutral}_j^w \vee \text{try-read}_j \vee \text{lock-read}_j \quad (6.6)$$

$$\phi'_3 \equiv [\forall \text{ other } j] \text{neutral}_j^r \vee \text{neutral}_j^w \vee \text{try-read}_j \vee \text{lock-read}_j \quad (6.7)$$

The specification  $\forall i: \text{type}(i) = 1. (\forall j: \text{type}(j) = 2. \text{AG}(\neg \text{lock-write}_i \vee \neg \text{lock-read}_j))$  states that in every execution, no reader and writer can lock the critical section simultaneously. This specification belongs to the class  $2\text{-LTL}\backslash X(P^1, P^2)$ .

The specification  $\forall i: \text{type}(i) = 1. (\forall j: \text{type}(j) = 2. \text{EG}((\text{try-write}_i \wedge \text{lock-read}_j) \rightarrow \text{lock-write}_i))$  states that there is an execution, in which a writer eventually obtains the lock, when a reader holds a lock. This specification belongs to the class  $2\text{-ELTL}\backslash X(P^1, P^2)$ .

**Example 6.12** Consider the guarded protocol shown in Figure 6.7. Denote with  $C$  process template  $P^1$  and with  $U$  process template  $P^2$ . Further, assume that every system instance contains exactly one instance of  $P^1$ .

The specification  $G(\text{try-write} \rightarrow F \text{ lock-write})$  states that in every execution, the controller—the only writer in the system—eventually obtains the write lock, whenever it tries to enter the critical section. This specification belongs to the classes  $LTL\backslash X(C)$  and  $LTL(C)$ .

The specification  $G(\text{try-write} \rightarrow X (\text{lock-write} \vee X \text{ lock-write}))$  states that in every execution the controller obtains the write lock in at most two system steps, after it has entered the state  $\text{try-write}$  (this specification is violated on a system instance with at least three readers). This specification belongs to the class  $LTL(C)$ , but not to the class  $LTL\backslash X(C)$ .

## 6.3 UNDECIDABILITY: BOOLEAN AND CONJUNCTIVE GUARDS

We start by showing that PMCP with boolean guarded protocols is undecidable in Theorem 6.13. For conjunctive guarded protocols, we review results from Emerson and Kahlon [2003b, Propositions 3.1, 3.2, 3.3] that show that PMCP is undecidable for  $\text{Reg}(A)$ ,  $\omega\text{Reg}(A)$ , and  $LTL(C)$ . We will see that the same proofs apply to the class  $\mathcal{P}_{\text{init-conj}}$ .

**Theorem 6.13**  $\text{PMCP}(\mathcal{P}_{\text{bool}}, C, LTL(C))$  is undecidable. The non-halting problem for two-counter machines can be reduced to  $\text{PMCP}(\mathcal{P}_{\text{bool}}, C, \{G \neg halt_C\})$  with  $d = 2$ , where  $halt_C$  is a local state of the controller  $C$ .

*Proof.* This proof is based on the idea mentioned by Emerson and Namjoshi [1996] in the conclusions.

Recall the notation from Section 3.1. Let  $\mathcal{M}$  be a 2CM with locations  $[m]$ , the initial state 1, the halting state  $m$ , and a set of commands  $\Delta \subseteq [m] \times \mathcal{A} \times [m]$ . We denote the two counters of  $\mathcal{M}$  by  $A$  and  $B$ . Given such a machine, we construct two process templates: Template  $C$  for the coordinator emulating the control flow of  $\Delta$ ; Template  $U$  for the user processes. Each user process is either storing a single unary digit of  $A$  or  $B$ , or staying in the standby state  $init_U$ . Given  $N \geq 1$ , the system  $(C, U)^{\mathbf{C}(2N+1)}$  simulates at least  $N$  steps of  $\mathcal{M}$ .

The construction ensures that whenever the controller of type  $C$  begins to execute a command  $a$ , exactly one user process of type  $U$  performs a unary increment or decrement as prescribed by  $a$ . This is ensured by the controller and the user processes using conjunctive and disjunctive guards as follows.

- Using either a conjunctive or a disjunctive guard, the controller tests the counters for zero by observing the local states of the user processes.
- The controller tests with a disjunctive guard, whether at least one user process heard the command by  $C$  and started to execute it.
- A user process tests with a conjunctive guard that no other user process has started to execute the command issued by the controller.

We construct template  $C$  such that it has three kinds of states.

- The initial state  $init_C$ .
- A state  $i \in [m]$  for each location  $i$  of the counter machine.
- A state  $\langle i, a, i' \rangle$  representing a transition  $(i, a, i') \in \Delta$ . We need this state to record that the machine is in the middle of executing a command  $a \in \{inc(A), dec(A), inc(B), dec(B)\}$ . When  $C$  moves from  $\langle i, a, i' \rangle$ , the system has finished to simulate the command  $a$  at location  $i$  and is continuing with the command at location  $i'$ .

Template  $U$  has the following states.

- The initial state  $init_U$ , where the process contributes neither to  $A$ , nor to  $B$ .
- Storage states  $A_1$  and  $B_1$  reflecting that the process contributes a unary 1 to  $A$  and  $B$ , respectively.
- Temporary states  $A_{01}$  and  $B_{01}$  meaning that the process is in the middle of adding a unary 1 to  $A$  or  $B$ , respectively.
- Temporary states  $A_{10}$  and  $B_{10}$  meaning that the process is in the middle of subtracting a unary 1 from  $A$  or  $B$ , respectively.

## 78 6. GUARDED PROTOCOLS

**Table 6.2:** The guards of the transitions that simulate the commands over counter  $A$ . The sets  $\Delta_A^+$  and  $\Delta_A^-$  are defined as follows:  $\Delta_A^+ = \{(i, a, i') \in \Delta \mid a = inc(A)\}$  and  $\Delta_A^- = \{(i, a, i') \in \Delta \mid a = dec(A)\}$ . To obtain the case for  $B$ , swap  $A$  and  $B$

C's transition	Guard
$(init_C, \tau, 1)$	$[\forall \text{ other } j] init_U$
$(i, inc(A), \langle i, inc(A), i' \rangle)$	$[\forall \text{ other } j] init_U \vee A_1 \vee B_1$
$(i, dec(A), \langle i, dec(A), i' \rangle)$	$[\forall \text{ other } j] init_U \vee A_1 \vee B_1$
$(\langle i, inc(A), i' \rangle, end(A), i')$	$[\exists \text{ other } j] A_{01}$
$(\langle i, dec(A), i' \rangle, end(A), i')$	$[\exists \text{ other } j] A_{10}$
$(i, zero(A), i')$	$[\forall \text{ other } j] init_U \vee B_1$
U's transition	Guard
$(init_U, inc(A), A_{01})$	$[\forall \text{ other } j] init_U \vee A_1 \vee B_1 \vee \bigvee_{\substack{t \in \Delta_A^+ \\ m}} \langle t \rangle$
$(A_{01}, \tau, A_1)$	$[\forall \text{ other } j] init_U \vee A_1 \vee B_1 \vee \bigvee_{k=1}^m k$
$(A_1, dec(A), A_{10})$	$[\forall \text{ other } j] init_U \vee A_1 \vee B_1 \vee \bigvee_{\substack{t \in \Delta_A^- \\ m}} \langle t \rangle$
$(A_{10}, \tau, init_U)$	$[\forall \text{ other } j] init_U \vee A_1 \vee B_1 \vee \bigvee_{k=1}^m k$

Figure 6.8 depicts the transitions of template  $C$ , and Figure 6.9 depicts the transitions of template  $U$ . Table 6.2 assigns the guards to the transitions of  $C$  and  $U$ .

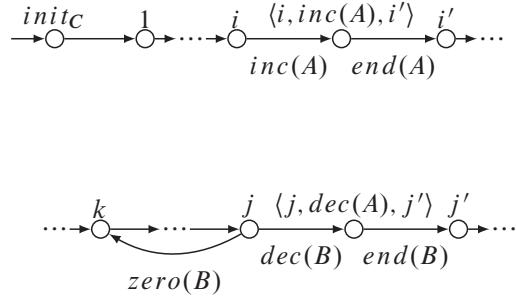
When a user process observes at least one process at state  $\langle i, a, i' \rangle$ —this can be only the controller—it starts to execute the command  $a$  by moving to one of temporary states  $\{A_{01}, A_{10}, B_{01}, B_{10}\}$ . After that the controller waits until one of  $U$ 's leaves its intermediate state. To prevent several user processes from executing the same command, the transition guards ensure that no other user process has started to execute the command. For instance, two user processes cannot both move to  $A_{01}$  after seeing  $\langle i, inc(A), i' \rangle$ , because as soon as the first of them moved to  $A_{01}$ , the guard of the second process becomes false, as the first process violates  $[\forall \text{ other } j] init_U \vee A_1 \vee B_1 \vee \bigvee_{(i, inc(A), i') \in \Delta} \langle i, inc(A), i' \rangle$ . Note that we are exploiting the interleaving semantics in this argument.

Given  $N \geq 1$ , the system instance of  $2N$  processes of type  $U$  and one process of type  $C$  simulates at least  $N$  steps of the two counter machine  $\mathcal{M}$ . By identifying proposition  $halt_C$  with the state  $m$  of the controller, we specify the non-halting property as  $G \neg m$  over the state of  $C$ . As we have encoded the undecidable non-halting problem of an arbitrary two-counter machine  $\mathcal{M}$  as  $\text{PMCP}(\{C, U\}, \mathbf{C}, (G \neg m))$ , we have proven the claim.  $\square$

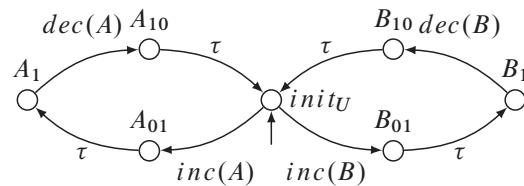
From Theorem 6.13, we know that  $\text{PMCP}(\mathcal{P}_{\text{bool}}, \mathbf{C}, \text{LTL}(C))$  is undecidable. So, we immediately have undecidability for the following specification classes.

- As the LTL-formula in the proof does not use the next-time operator  $\text{X}$ , the proof applies to  $\text{LTL} \setminus \text{X}(\mathcal{C})$  without modification.
- As  $\text{LTL} \setminus \text{X}(\mathcal{C})$  is subclass of 2-LTL  $\setminus \text{X}(\mathcal{P}^\ell, \mathcal{P}^{\ell'})$ , the proof applies to 2-LTL  $\setminus \text{X}(\mathcal{P}^\ell, \mathcal{P}^{\ell'})$  and 2-ELTL  $\setminus \text{X}(\mathcal{P}^\ell, \mathcal{P}^{\ell'})$  as well.
- One can encode a non-deterministic choice whether a process takes the role of the controller or the user. Using conjunctive guards one assures that the first process to take a step becomes the controller, and all other processes become user processes. Then Theorem 6.13 applies to 1-LTL  $\setminus \text{X}(\mathcal{P}^\ell)$  and 1-ELTL  $\setminus \text{X}(\mathcal{P}^\ell)$ .

**Corollary 6.14**  $\text{PMCP}(\mathcal{P}_{\text{bool}}, \mathbf{C}, \mathcal{F})$  is undecidable for every specification class  $\mathcal{F}$  from the list:  $\text{LTL} \setminus \text{X}(\mathcal{C})$ ,  $\text{LTL}(\mathcal{C})$ , 1-LTL  $\setminus \text{X}(\mathcal{P}^\ell)$ , 1-ELTL  $\setminus \text{X}(\mathcal{P}^\ell)$ , 2-LTL  $\setminus \text{X}(\mathcal{P}^\ell, \mathcal{P}^{\ell'})$ , 2-ELTL  $\setminus \text{X}(\mathcal{P}^\ell, \mathcal{P}^{\ell'})$ .



**Figure 6.8:** Template  $C$  simulating the control flow of a 2CM over counters  $A$  and  $B$ . The edges are labeled with actions; if an edge is not labeled, then the action is  $\tau$ .



**Figure 6.9:** Template  $U$  of user processes that together simulate two integer counters  $A$  and  $B$ . An instance of  $U$  contributes a unary 1 to  $A$  (resp.  $B$ ), when in  $A_1$  (resp. in  $B_1$ ). The edges are labeled with actions; if an edge is not labeled, then the action is  $\tau$ .

We now turn our attention to conjunctive guards. The case of conjunctive guards is undecidable for  $\text{LTL}(\mathcal{C})$ , as shown by Emerson and Kahlon [2003b, Propositions 3.1–3.2]. We re-use the

## 80 6. GUARDED PROTOCOLS

**Table 6.3:** The guards of the transitions simulating the 2CM's commands labeled with actions  $a \in \{inc(A), dec(A)\}$  and  $zero(A)$ . To obtain the case for  $B$ , swap  $A$  and  $B$

C's transition	Guard
$(init_C, \tau, 1)$	$[\forall \text{ other } j] init_U$
$(i, a, \langle i, a, i' \rangle)$	$[\forall \text{ other } j] init_U \vee A_1 \vee B_1$
$(\langle i, a, i' \rangle, end(A), i')$	$true$
$(i, zero(A), i')$	$[\forall \text{ other } j] init_U \vee B_1$
U's transition	Guard
$(init_U, inc(A), A_{01})$	$[\forall \text{ other } j] init_U \vee A_1 \vee B_1 \vee \bigvee_{t \in \Delta_A^+} \langle t \rangle$
$(A_1, dec(A), A_{10})$	$[\forall \text{ other } j] init_U \vee A_1 \vee B_1 \vee \bigvee_{t \in \Delta_A^-} \langle t \rangle$
$(A_{01}, \tau, A_1)$	$[\forall \text{ other } j] init_U \vee A_1 \vee B_1 \vee \bigvee_{k=1}^m k$
$(A_{10}, \tau, init_U)$	$[\forall \text{ other } j] init_U \vee A_1 \vee B_1 \vee \bigvee_{k=1}^m k$

construction of  $C$  and  $U$  from the proof of Theorem 6.13. As we cannot use disjunctive guards anymore, we replace several guards with  $true$  as reflected in Table 6.3. Rather, we encode the interlocking behavior in the specification using the next-time operator  $\mathbf{X}$ .

**Theorem 6.15**  $\text{PMCP}(\mathcal{P}_{\text{conj}}, \mathbf{C}, \text{LTL}(C))$  is undecidable. The non-halting problem for two-counter machines can be reduced to  $\text{PMCP}(\mathcal{P}_{\text{conj}}, \mathbf{C}, \{\mathbf{G}\neg\mathbf{halt}_C\})$  with  $d = 2$ , where  $\mathbf{halt}_C$  is a local state of the controller  $C$ .

**Proof.** We change the commands from Table 6.2 of the proof of Theorem 6.13, as shown in Table 6.3. The guards of the transitions  $(\langle i, inc(A), i' \rangle, end(A), i')$  and  $(\langle i, dec(A), i' \rangle, end(A), i')$  are replaced with  $true$ . With the new guards the reachability of location  $m$  in a family  $\{(C, U)^{\mathbf{C}(1,n)}\}_{n \geq 1}$  does not simulate the two-counter machine anymore, as the process  $C$  may invoke  $inc(A)$  and then immediately proceed to the next control location without waiting for a user process performing the operation. Indeed, the guard is  $true$ , so  $C$  can jump out of  $\langle i, inc(A), i' \rangle$  earlier than it could with the disjunctive guards in Table 6.2. To deal with that issue, we only consider restricted runs. This can be done by constructing a specification that uses the  $\mathbf{X}$  operator, and is thus represented with  $\text{LTL}(C)$  instead of  $\text{LTL} \setminus \mathbf{X}(C)$ . Formally, given an action  $a \in \{inc(A), dec(A), dec(B), dec(B)\}$  of the counter machine, we introduce an LTL formula  $yield_a$ :

$$yield_a = \mathbf{G} \left( \bigwedge_{(i,a,i') \in \Delta} ((\neg \langle i, a, i' \rangle \wedge \mathbf{X} \langle i, a, i' \rangle) \rightarrow \mathbf{XX} \langle i, a, i' \rangle) \right).$$

Observe that in runs in which  $yield_{inc(A)}$  holds,  $C$  cannot make the next step immediately after going to  $\langle i, inc(A), i' \rangle$ . As a consequence, the formula can be used to filter out only the executions, where a process of type  $U$  makes a step right after  $C$  moved to  $\langle i, inc(A), i' \rangle$ .

Finally, the  $yield_a$  formulas are combined with the non-halting property. Thus, we arrive at the parameterized model checking problem, where a system instance with  $2N$  user processes is simulating at least  $N$  steps of the two counter machine  $\mathcal{M}$ :

$$\forall n \geq 1. (C, U)^{\mathbf{C}(1,n)} \models (yield_{inc(A)} \wedge yield_{inc(B)} \wedge yield_{dec(A)} \wedge yield_{dec(B)}) \rightarrow G(\neg \text{halt}_C).$$

As this problem is an instance of  $\text{PMCP}(\mathcal{P}_{\text{conj}}, \mathbf{C}, \text{LTL}(C))$ , we conclude that the latter is undecidable.

Note carefully that the constructed specification is not a safety property, as its negated normal form is using temporal operators  $G$  and  $F$ .  $\square$

In fact, the guarded protocol constructed in the proof of Theorem 6.15 uses only init-conjunctive guards, that is, all the guards in Table 6.3 contain  $init_U$ . Thus, we immediately arrive at the conclusion that PMCP is undecidable for  $\mathcal{P}_{\text{init-conj}}$ , which is a subclass of  $\mathcal{P}_{\text{conj}}$ :

**Corollary 6.16** *The problem  $\text{PMCP}(\mathcal{P}_{\text{init-conj}}, \mathbf{C}, \text{LTL}(C))$  is undecidable.*

As noted by Emerson and Kahlon [2003b, Proposition 3.3], the proof of Theorem 6.15 can be also applied to regular and  $\omega$ -regular action-based specifications.

**Theorem 6.17**  *$\text{PMCP}(\mathcal{P}_{\text{conj}}, \mathbf{C}, \text{Reg}(A))$  and  $\text{PMCP}(\mathcal{P}_{\text{conj}}, \mathbf{C}, \omega\text{Reg}(A))$  are undecidable.*

*Proof.* We re-use the proof from Theorem 6.15, except that the  $yield$  constraints have to be expressed as a regular language over actions, rather than an LTL formula. We show how to express  $yield_{inc(A)}$  and  $yield_{dec(A)}$ ; the formulas for  $B$  are obtained similarly.

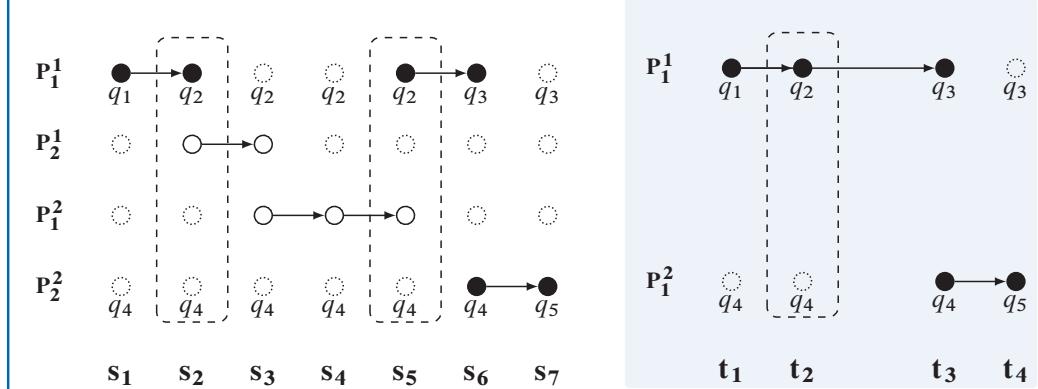
Let letter  $i$  denote an action  $inc(A)$  or  $dec(A)$  of  $C$ , letter  $e$  denote the action  $end(A)$ , and expression  $X$  be the regular expression that recognizes the complement  $\Sigma_{pr} \setminus \{i, e\}$  of  $i$  and  $e$ . Then  $yield_i$  can be written as the following regular expression:

$$yield_i = (i \cdot e \mid X)^*.$$

As the parameterized model checking problem is undecidable for regular properties, so it is for  $\omega$ -regular properties.  $\square$

## 6.4 DECIDABILITY: INIT-CONJUNCTIVE AND DISJUNCTIVE GUARDS

In contrast to the classes of boolean and conjunctive guards of the previous section, the PMCP over indexed  $\text{LTL} \setminus X$  formulas and  $\text{LTL} \setminus X(C)$  is decidable for the classes of init-conjunctive and disjunctive guards [Emerson and Kahlon, 2000]. In this section, we present the results from a



**Figure 6.10:** Illustration of path reduction. Processes 1 and 4 of  $(P^1, P^2)^{(2,2)}$  are projected onto Processes 1 and 2 of  $(P^1, P^2)^{(1,1)}$  using  $pr_{idx} : \{1 \mapsto 1, 4 \mapsto 2\}$ . Thus, the run  $s_1 \dots s_7$  becomes the sequence  $t_1 \dots t_4$  by collapsing  $s_2 \dots s_5$  to  $t_2$  and hiding the local states of processes 2 and 3 on the left. Observe that  $t_1 = pr_{st}(s_1)$  and  $t_4 = pr_{st}(s_7)$ .

slightly different angle than the original work by Emerson and Kahlon [2000]: we give a general proof schema that structures cutoff proofs for the disjunctive and init-conjunctive guards; and we explain why both types of guards preserve their satisfiability when the system has “enough” processes. We have to pay attention to deadlocked computations in all cases. Here, we give the proofs of the cutoff results as instances of two general proof schemas: namely, *bounding* shows how to reduce a run in a “large” system to a run in “small” cutoff system; *monotonicity* constructs an equivalent run (a term we make precise later) of a large system from a run of a small system. Together, bounding and monotonicity prove equivalence of large systems to the cutoff system with respect to the formulas from 1-LTL\X, 1-ELTL\X( $P^\ell$ ), and LTL\X( $C$ ).

#### 6.4.1 PRELIMINARIES

We start with the ingredients of the schemas: the counting function, path reduction, and symmetry argument.

**Counting function.** Given a system instance  $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$  with the set of global states  $S$ , we define the *counting function*  $\kappa : S \times (Q^1 \cup Q^2) \rightarrow \mathbb{N}_0$ . For every global state  $s \in S$  and a local state  $q \in Q^1 \cup Q^2$ , the number of times  $q$  occurs in  $s$  is given by  $\kappa(s, q)$ . When it is clear from context, we apply the same symbol  $\kappa$  to different system instances.

**Path reduction.** Intuitively, given a path (a sequence of global states) in a large system, we project the global states onto a subset of the processes. Steps in the original path performed by the pro-

cesses that are not in the subset result in stuttering in the projection. In path reduction we remove stuttering that is due to this effect. See Figure 6.10 for an illustration.

Formally, consider two system instances  $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$  and  $(P^1, P^2)^{\mathbf{C}(m_1, m_2)}$  with  $n_1 \geq m_1$  and  $n_2 \geq m_2$ . An *index projection* is a bijective partial function  $pr_{idx}$  from  $V(\mathbf{C}(n_1, n_2))$  to  $V(\mathbf{C}(m_1, m_2))$ . For  $v \in V(\mathbf{C}(n_1, n_2))$ , we write  $pr_{idx}(v) = \perp$  to denote that  $pr_{idx}$  is undefined on  $v$ . Using  $pr_{idx}$ , we define a state projection  $pr_{st}$ . Namely, given a state  $s$  of  $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$ , state  $t = pr_{st}(s)$  of  $(P^1, P^2)^{\mathbf{C}(m_1, m_2)}$  is defined as follows: For every  $i \in \mathbf{I}((n_1, n_2))$ , if  $pr_{idx}(i) \neq \perp$ , then the local state  $t(pr_{idx}(i))$  is defined as  $s(i)$ .

Given the functions  $pr_{idx}$  and  $pr_{st}$ , we define a function  $pr_{path}$  that maps a run  $\pi = s_1 s_2 \dots$  of  $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$  to a sequence of states of  $(P^1, P^2)^{\mathbf{C}(m_1, m_2)}$ . Note that it is part of the framework to prove that the latter sequence is actually a run. To this end, we partition  $\pi$  into (finitely or infinitely many) paths  $\Pi_1, \Pi_2, \dots$  with the following properties for every  $k \geq 1$ .

- The last state of  $\Pi_k$  and the first state of  $\Pi_{k+1}$  form a transition of a process whose index  $i$  satisfies  $i \in \mathbf{I}((n_1, n_2))$  and  $pr_{idx}(i) \neq \perp$ .
- Every pair of successive states in  $\Pi_k$  forms a transition of a process whose index  $i$  satisfies  $i \in \mathbf{I}((n_1, n_2))$  and  $pr_{idx}(i) = \perp$ .

Let  $s_{j(1)} s_{j(2)} \dots$  be the sequence of the first states of  $\Pi_1, \Pi_2, \dots$ . Then  $pr_{path}(\pi)$  is defined as follows.

- If the sequence  $\Pi_1 \Pi_2 \dots$  is infinite, then  $pr_{path}(\pi) = pr_{st}(s_{j(1)}) pr_{st}(s_{j(2)}) \dots$
- If the sequence  $\Pi_1 \Pi_2 \dots$  has length  $k$ , then  $pr_{path}(\pi) = pr_{st}(s_{j(1)}) pr_{st}(s_{j(2)}) \dots pr_{st}(s_{j(k)}) (pr_{st}(s_{j(k)}))^{\omega}$ .

Given a path  $\pi$ ,  $pr_{path}(\pi)$  is a sequence, not necessarily a run. As  $pr_{st}$  and  $pr_{path}$  remove processes, the sequence  $pr_{path}(\pi)$  is not immediately a path of  $(P^1, P^2)^{\mathbf{C}(m_1, m_2)}$ . The partitioning of  $pr_{path}(\pi)$  implies that  $pr_{path}(\pi)$  is *stuttering equivalent* to  $\pi$ ; a state  $s_i$  of  $pr_{path}(\pi)$  forms its own partition that is equivalent to partition  $\Pi_i$ . Similarly to [Baier and Katoen, 2008, Theorem 7.92], one can prove the following.

**Observation 6.18** Let  $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$  and  $(P^1, P^2)^{\mathbf{C}(m_1, m_2)}$  be two system instances of a guarded protocol  $(P^1, P^2)$  such that  $n_2 \geq m_2$  and  $n_1 \geq m_1$ . Let  $pr_{idx}$  be an index projection from  $\mathbf{C}(n_1, n_2)$  to  $\mathbf{C}(m_1, m_2)$  and  $c \in V(\mathbf{C}(n_1, n_2))$  a process index.

Consider an LTL\X-formula  $\varphi$  over atomic propositions  $Q^{\text{type}(c)} \times \{c\}$  and an LTL\X-formula  $\varphi'$  that is the result of substitution of  $c$  with  $pr_{idx}(c)$ . For each run  $\pi$  of  $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$ , the following holds:

$$(P^1, P^2)^{\mathbf{C}(n_1, n_2)}, \pi \models \varphi \text{ if and only if } (P^1, P^2)^{\mathbf{C}(m_1, m_2)}, pr_{path}(\pi) \models \varphi'.$$

Of course, we are interested only in an index projection  $pr_{idx}$  that actually generates runs, that is, it turns every run  $\pi$  of  $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$  into a run  $pr_{path}(\pi)$  of  $(P^1, P^2)^{\mathbf{C}(m_1, m_2)}$ . We call such a projection a *proper index projection*.

## 84 6. GUARDED PROTOCOLS

An index projection removes processes, and thus it cannot increase the number of processes in a local state. This intuition is formulated in the following proposition, which is easy to prove.

**Proposition 6.19** *Let  $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$  be a system instance with the set of global states  $S$  and  $pr_{idx}$  an index projection. For every global state  $s \in S$  and every local state  $q \in Q^1 \cup Q^2$ , it holds that  $\kappa(s, q) \geq \kappa(pr_{st}(s), q)$ .*

**Symmetry argument.** Similarly to parameterized token rings (cf. [Emerson and Namjoshi \[1995\]](#)), it was noticed by [Emerson and Kahlon \[2000\]](#) that model checking of a guarded protocol with a fixed number of processes against a formula from  $1\text{-ELTL}\setminus X(P^\ell)$  or  $1\text{-LTL}\setminus X(P^\ell)$  can be reduced to model checking against an  $\text{ELTL}\setminus X$  formula over the states of a fixed process:

### Observation 6.20

For a system instance  $(P^1, P^2)^{\mathbf{C}(m_1, m_2)}$  of a guarded protocol, process type  $\ell \in \{1, 2\}$ , index  $c$  of a process of type  $\ell$ , and an indexed  $\{A, E, X\}$ -free path formula  $\varphi(i)$ , the following holds.

1.  $(P^1, P^2)^{\mathbf{C}(m_1, m_2)} \models \forall i : \text{type}(i) = \ell. E\varphi(i)$  if and only if  $(P^1, P^2)^{\mathbf{C}(m_1, m_2)} \models E\varphi(c)$ .
2.  $(P^1, P^2)^{\mathbf{C}(m_1, m_2)} \models \forall i : \text{type}(i) = \ell. A\varphi(i)$  if and only if  $(P^1, P^2)^{\mathbf{C}(m_1, m_2)} \not\models E\neg\varphi(c)$ .

When having guarded protocols with one controller  $C$  and many user processes  $U$ , Observation 6.20(2) leads to the following corollary.

### Corollary 6.21

For a system instance  $(C, U)^{\mathbf{C}(1, m_2)}$  of a guarded protocol  $(C, U)$ , and an  $\{A, E, X\}$ -free path formula  $\varphi$  over the states of controller  $C$ , the following holds:  $(C, U)^{\mathbf{C}(1, m_2)} \models A\varphi$  if and only if  $(C, U)^{\mathbf{C}(1, m_2)} \not\models E\neg\varphi$ .

### 6.4.2 PROOF SCHEMAS

Based on counting function, path reduction, and symmetry argument, we can define the two proof schemas that are used to prove cutoff results for disjunctive and init-conjunctive guards.

#### Bounding schema

- i. Estimate the cutoff size  $(c_1, c_2)$  based on the number of local states in  $P^1$  and  $P^2$  and the types of guards (disjunctive, init-conjunctive). Typically, one gives an arithmetic expression for  $c_1$  and  $c_2$  over  $|Q^1|$  and  $|Q^2|$ , respectively. Fix system size  $(n_1, n_2) \geq (c_1, c_2)$  and an arbitrary run  $\pi = s_1 s_2 \dots$  of  $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$ , which we use in the rest of the proof. In case of disjunctive guards, we require an intermediate step and  $\pi$  actually is a representative run with a special structure.
- ii. Based on point (i), construct an index projection  $pr_{idx}$  from  $\mathbf{C}(n_1, n_2)$  to  $\mathbf{C}(c_1, c_2)$ . This leads to the mapping  $pr_{path}$ , as discussed in the path reduction.

- iii. Show that the guards of  $\pi$  are not locked in  $pr_{path}(\pi)$ . From this follows that the pairs of consecutive states in  $pr_{path}(\pi)$  form a transition, and thus that the sequence  $pr_{path}(\pi)$  is a path of  $(P^1, P^2)^{\mathbf{C}(c_1, c_2)}$ . If  $\pi$  is a deadlocked run, then prove that  $pr_{path}(\pi)$  is also a deadlocked run.

This actually constitutes the core of the proof. One shows that if a guard evaluates to true in a state  $s$  of  $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$ , then it is also true in state  $pr_{st}(s)$  of  $(P^1, P^2)^{\mathbf{C}(c_1, c_2)}$ . Typically, if one underestimates  $(c_1, c_2)$  at point (i), then the proof breaks at this point.

- iv. Apply Observation 6.18 to conclude that for every index  $k$  with  $pr_{idx}(k) \neq \perp$  it holds that  $(P^1, P^2)^{\mathbf{C}(n_1, n_2)} \models E\varphi(k)$  implies  $(P^1, P^2)^{\mathbf{C}(c_1, c_2)} \models E\varphi(pr_{idx}(k))$ .

#### Monotonicity schema

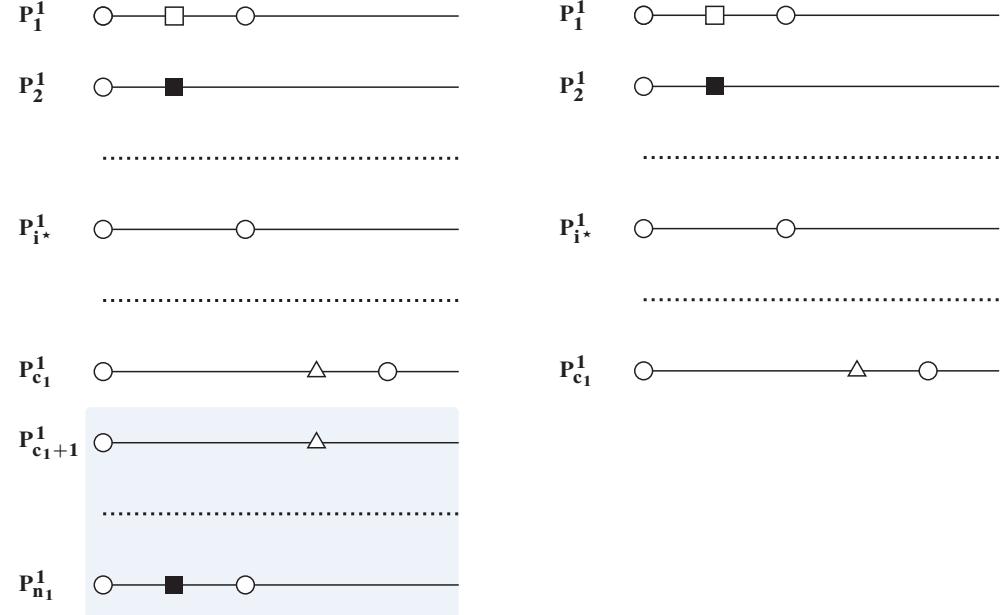
- i. Fix system sizes  $(m_1, m_2)$  and  $(m_1 + d_1, m_2 + d_2)$  with  $d_1 + d_2 = 1$  and  $d_1, d_2 \in \{0, 1\}$ . Pick an arbitrary run  $\pi = t_1 t_2 \dots$  of  $(P^1, P^2)^{\mathbf{C}(m_1, m_2)}$ . The goal of the next steps is to construct a corresponding run  $\sigma$  of system instance  $(P^1, P^2)^{\mathbf{C}(m_1 + d_1, m_2 + d_2)}$ .
- ii. Construct a sequence  $\sigma = s_1 s_2 \dots$  of global states of  $(P^1, P^2)^{\mathbf{C}(m_1 + d_1, m_2 + d_2)}$  by mapping every pair of states  $(t_i, t_{i+1})$  of  $\pi$  on a sequence of states in  $(P^1, P^2)^{\mathbf{C}(m_1 + d_1, m_2 + d_2)}$ . Typically, the pair of states is replaced by one or two transitions that mimic  $(t_i, t_{i+1})$ .
- iii. Show that the guards of the transitions in  $\sigma$  evaluate to true. Thus, the pairs of consecutive states in  $\sigma$  form a transition, and the sequence  $\sigma$  is a run of  $(P^1, P^2)^{\mathbf{C}(m_1 + d_1, m_2 + d_2)}$ . If  $\pi$  is a deadlocked run, then prove that  $\sigma$  is also a deadlocked run.
- iv. Construct an index mapping  $pr_{idx}$  from  $\mathbf{C}(m_1 + d_1, m_2 + d_2)$  to  $\mathbf{C}(m_1, m_2)$  as follows:  $pr_{idx}(m_1 + d_1 + m_2 + d_2) = \perp$ ; for all  $i \leq m_1 + m_2$ ,  $pr_{idx}(i) = i$ . Conclude that  $pr_{path}(\sigma) = \pi$ .
- v. From  $pr_{path}(\sigma) = \pi$ , conclude that for every index  $k \leq m_1 + m_2$  it holds that  $(P^1, P^2)^{\mathbf{C}(m_1, m_2)} \models E\varphi(pr_{idx}(k))$  implies  $(P^1, P^2)^{\mathbf{C}(m_1 + d_1, m_2 + d_2)} \models E\varphi(k)$ .

In the following sections, we instantiate the bounding and monotonicity proof schemas for init-conjunctive and disjunctive guards. Figures 6.11, 6.12, 6.13, 6.14, and 6.15 illustrate these proofs.

**Cutoff argument.** Assume that the bounding schema works for a cutoff size  $(c_1, c_2)$ . By applying inductively the monotonicity schema and Observation 6.20, we conclude that the following holds for every  $n_1 \geq c_1$  and  $n_2 \geq c_2$  and every formula  $\psi$  from 1-LTL\X and 1-ELTL\X:

$$(P^1, P^2)^{\mathbf{C}(n_1, n_2)} \models \psi \text{ if and only if } (P^1, P^2)^{\mathbf{C}(c_1, c_2)} \models \psi.$$

## 86 6. GUARDED PROTOCOLS



**Figure 6.11:** Illustration of the bounding schema for the init-conjunctive guards. For simplicity, we left out the processes instantiated from the template  $P^2$ . Local states of the processes are depicted with  $\circ$ ,  $\square$ ,  $\blacksquare$ , and  $\triangle$ . The processes in the shaded area are hidden by the index projection function  $pr_{idx}$ , and thus only the processes  $P^1_1, \dots, P^1_{c_1}$  are left after the bounding schema has been applied. The process  $P^1_{i^*}$  makes infinitely many transitions, unless the original execution is deadlocked. (If the process  $P^1_{i^*}$  has an index above  $c_1$ , we first apply the symmetry argument and swap it with an arbitrary process having an index below  $c_1$ , e.g., process  $P^1_2$ .)

### 6.4.3 INIT-CONJUNCTIVE GUARDS

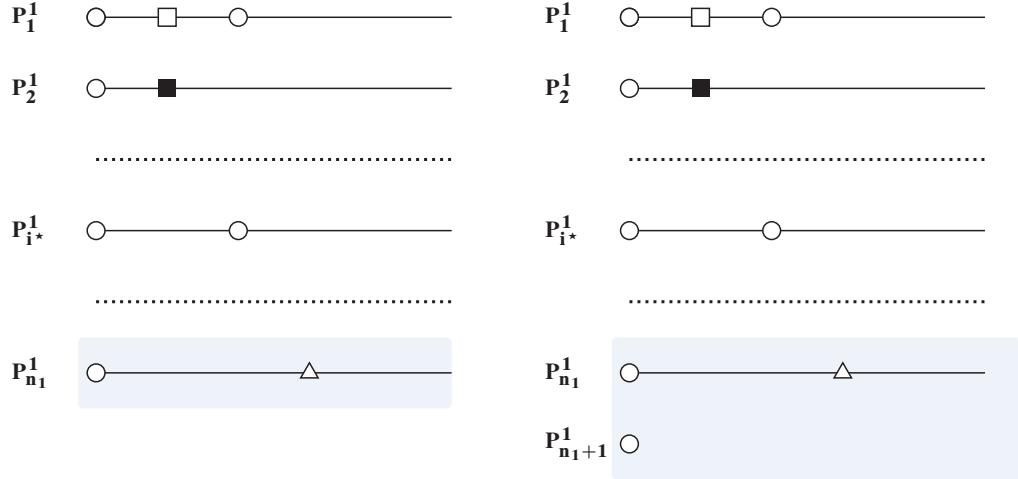
Here we instantiate the bounding and monotonicity schemas to arrive at the following theorem for init-conjunctive guarded protocols [Emerson and Kahlon, 2000, Theorem 7]).

#### Theorem 6.22

For number  $\ell \in \{1, 2\}$  and a specification class  $\mathcal{F}$  from  $1\text{-LTL}\backslash X(P^\ell)$ ,  $1\text{-ELTL}\backslash X(P^\ell)$ , or  $LTL\backslash X(C)$ , the problem  $PMCP(\mathcal{P}_{init\text{-conj}}[2], C, \mathcal{F})$  is decidable.

Moreover, for every init-conjunctive guarded protocol  $(P^1, P^2)$ , there exists a cutoff of size  $(c_1, c_2)$ , that is for every formula  $\varphi \in \mathcal{F}$ , the following two statements are equivalent.

1. For every clique  $C(n_1, n_2)$  it holds  $(P^1, P^2)^{C(n_1, n_2)} \models \varphi$ .
2. For every clique  $C(m_1, m_2)$  with  $m_1 \leq c_1$  and  $m_2 \leq c_2$ , it holds  $(P^1, P^2)^{C(m_1, m_2)} \models \varphi$ .



**Figure 6.12:** Illustration of the monononicity schema for init-conjunctive guards. For simplicity, we left out the processes instantiated from the template  $P^2$ . Local states of the processes are depicted with  $\circlearrowleft$ ,  $\square$ ,  $\blacksquare$ , and  $\triangle$ . The additional process  $P_{n_1+1}^1$  never leaves the initial state  $\circlearrowleft$ .

The cutoff size is defined by  $c_\ell = 2|Q^\ell| + 1$  and  $c_{3-\ell} = 2|Q^{3-\ell}|$ .

For the rest of the section, we fix a guarded protocol  $(P^1, P^2)$  with init-conjunctive guards and a  $\{A, E, X\}$ -free formula  $\varphi(1)$  over the states of process 1. As in Theorem 6.22, we fix cutoff size to be  $c_1 = 2|Q^1| + 1$  and  $c_2 = 2|Q^2|$ .

For bounding schema we consider two cases: (1) a run is free of deadlocks; and (2) a run has a deadlock.

(1) *Bounding schema for deadlock-free runs:*

- i. Let  $\pi = s_1 s_2 \dots$  be a deadlock-free run of  $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$ . This implies that there is a process that performs infinitely many steps in  $\pi$ . Let  $i^*$  be the index of such a process.
- ii. Construct an index projection  $pr_{idx}$  from  $\mathbf{C}(n_1, n_2)$  to  $\mathbf{C}(c_1, c_2)$  that has the following properties: (a)  $pr_{idx}(1) = 1$ ; (b)  $pr_{idx}(i^*) \neq \perp$ . Note carefully that by the definition of index projection, exactly  $c_1 + c_2$  indices of  $\mathbf{C}(n_1, n_2)$  are mapped on  $\mathbf{C}(c_1, c_2)$ .
- iii. We have to show that for every  $i \geq 1$ , the guard of the transition from  $s_i$  to  $s_{i+1}$  is evaluated to true in  $pr_{st}(s_i)$ . Consider the transition from  $s_i$  to  $s_{i+1}$  made by process  $p$ . Let its guard  $g_i$  be  $[\forall \text{ other } p] q_1 \vee \dots \vee q_k$ . As  $s_i \models_p g_i$ , all processes are in a state from  $\{q_1, \dots, q_k\}$ , and for every  $q \in Q^1 \cup Q^2 \setminus \{q_1, \dots, q_k\}$  it holds that  $\kappa(s_i, q) = 0$ . Hence, from Proposition 6.19, it immediately follows that  $\kappa(pr_{st}(s_i), q) = 0$ . From the latter we conclude that  $pr_{st}(s_i) \models_{pr_{idx}(p)} g_i$ . Thus,  $pr_{path}(\pi)$  constitutes a run of  $(P^1, P^2)^{\mathbf{C}(c_1, c_2)}$ .

## 88 6. GUARDED PROTOCOLS

- iv. Apply Observation 6.18 to conclude that for every index  $k$  with  $pr_{idx}(k) \neq \perp$  it holds that  $(P^1, P^2)^{\mathbf{C}(n_1, n_2)} \models E\varphi(k)$  implies  $(P^1, P^2)^{\mathbf{C}(c_1, c_2)} \models E\varphi(pr_{idx}(k))$ .

(2) *Bounding schema for deadlocked runs:*

- i. Let  $\pi = s_1 s_2 \dots s_d (s_d)^\omega$  be a deadlocked run of  $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$ .
- ii. We construct an index projection  $pr_{idx}$  from  $\mathbf{C}(n_1, n_2)$  to  $\mathbf{C}(c_1, c_2)$  with the following properties:
  - (a)  $pr_{idx}(1) = 1$ ; and
  - (b) for every  $q \in Q^1 \cup Q^2$ ,  $\kappa(pr_{st}(s_d), q) \geq \min(\kappa(s_d, q), 2)$ .

Such a mapping exists, as  $c_1 + c_2 = 2 \cdot (|Q^1| + |Q^2|) + 1$ .

- iii. As in the deadlock-free case, for every  $i \geq 1$  and every  $q \in Q^1 \cup Q^2$ ,  $\kappa(s_i, q) \geq \kappa(pr_{path}(s_i), q)$ . It follows that for every  $i : 1 \leq i < d$  and every process index  $p : 1 \leq p \leq n_1 + n_2$ ,  $s_i \models_p g_i \Rightarrow s_i \models_{pr_{idx}(p)} g_i$ .

It remains to prove that  $pr_{st}(s_d)$  is a deadlock state, that is, for every guard  $g$  and some process index  $p$ , it holds  $s_d \not\models_p g \Rightarrow pr_{st}(s_d) \not\models_{pr_{idx}(p)} g$ . Let us fix  $g$  to be  $[\forall \text{ other } p] q_1 \vee \dots \vee q_k$  and a process index  $p$ . As  $s_d, p \not\models g$ , there is a state  $q \in Q^1 \cup Q^2 \setminus \{q_1, \dots, q_k\}$  with the following properties:

- (a) if  $s_d(p) = q$ , then  $\kappa(s_d, q) \geq 2$ ; and
- (b) if  $s_d(p) \neq q$ , then  $\kappa(s_d, q) \geq 1$ .

From point (ii) we have  $\kappa(pr_{st}(s_d), q) \geq \min(\kappa(s_d, q), 2)$ . Hence, in case (a), we have  $\kappa(pr_{st}(s_d), q) \geq 2$ . In case (b), we have  $\kappa(pr_{st}(s_d), q) \geq 1$ . In both cases we conclude that  $pr_{st}(s_d) \not\models_{pr_{idx}(p)} g$ . Thus,  $pr_{st}(s_d)$  is a deadlock state, as required.

- iv. Apply Observation 6.18 to conclude that for every index  $k$  with  $pr_{idx}(k) \neq \perp$  it holds that  $(P^1, P^2)^{\mathbf{C}(n_1, n_2)} \models E\varphi(k)$  implies  $(P^1, P^2)^{\mathbf{C}(c_1, c_2)} \models E\varphi(pr_{idx}(k))$ .

*Monotonicity schema:*

- i. Fix system sizes  $(m_1, m_2)$  and  $(m_1 + d_1, m_2 + d_2)$  for  $\ell \in \{1, 2\}$  and  $d_1 = \ell \cdot (2 - \ell)$  and  $d_2 = \ell \cdot (\ell - 1)$ . Pick an arbitrary run  $\pi = t_1 t_2 \dots$  of  $(P^1, P^2)^{\mathbf{C}(m_1, m_2)}$ .
- ii. Map every pair  $(t_i, t_{i+1})$  on a pair  $(s_i, s_{i+1})$ , where for all  $j : 1 \leq j \leq m_1 + m_2$ ,  $s_i(j) = t_i(j)$ ,  $s_{i+1}(j) = t_{i+1}(j)$  and  $s_i(m_1 + m_2 + d_1 + d_2) = s_{i+1}(m_1 + m_2 + d_1 + d_2) = init^\ell$ .
- iii. We have to show that for every guard  $g$ , every process index  $p$ , and every  $i \geq 1$ , if  $t_i \models_p g$ , then  $s_i \models_{pr_{idx}(p)} g$ . This is immediate, as, by the definition, every init-conjunctive guard contains  $init^\ell$ , which is the only state the added process resides in.

- iv. Construct an index mapping  $pr_{idx}$  from  $\mathbf{C}(m_1 + d_1, m_2 + d_2)$  to  $\mathbf{C}(m_1, m_2)$  as follows:  $pr_{idx}(m_1 + d_1 + m_2 + d_2) = \perp$ ; for all  $i \leq m_1 + m_2$ ,  $pr_{idx}(i) = i$ . Conclude that  $pr_{path}(\sigma) = \pi$ .
- v. From  $pr_{path}(\sigma) = \pi$ , conclude that for every index  $k \leq m_1 + m_2$  it holds that  $(P^1, P^2)^{\mathbf{C}(m_1, m_2)} \models E\varphi(pr_{idx}(k))$  implies  $(P^1, P^2)^{\mathbf{C}(m_1+d_1, m_2+d_2)} \models E\varphi(k)$ .

Figure 6.11 illustrates the bounding schema, and Figure 6.12 illustrates the monotonicity schema.

By applying the bounding and monotonicity schemas, we finish the proof of Theorem 6.22.

**Further Results.** As one can see from the proof of Theorem 6.22, if the system instances are deadlock-free, there is a trivial cutoff.

**Theorem 6.23 [Emerson and Kahlon, 2000].** Consider a guarded protocol  $(P^1, \dots, P^d)$  with conjunctive guards and a formula  $\varphi$  from  $1\text{-LTL} \setminus X(P^\ell)$ ,  $1\text{-ELTL} \setminus X(P^\ell)$ , or  $LTL \setminus X(C)$  for some  $\ell \in [d]$ . If every instance  $(P^1, \dots, P^d)^{\mathbf{C}(n)}$  does not have deadlocked runs, then there is a cutoff system of size  $(1, \dots, 1, 2, 1, \dots, 1)$ , i.e., the following two statements are equivalent.

1. For each  $(n_1, \dots, n_d) \geq (1, \dots, 1)$ , it holds  $(P^1, \dots, P^d)^{\mathbf{C}(n_1, \dots, n_d)} \models \varphi$ .
2. The cutoff instance of size  $(c_1, \dots, c_d)$ , where  $c_\ell = 2$  and  $\forall k \neq \ell. c_k = 1$ , satisfies the formula:  $(P^1, \dots, P^d)^{(c_1, \dots, c_d)} \models \varphi$ .

A similar argument works for properties that require only finite paths, e.g., reachability and safety.

**Theorem 6.24 [Emerson and Kahlon, 2000].** Consider a  $d$ -ary conjunctive system template  $(P^1, \dots, P^d)$  and a formula  $\varphi$  from  $1\text{-LTL} \setminus X(P^\ell)$ ,  $1\text{-ELTL} \setminus X(P^\ell)$ , or  $LTL \setminus X(C)$  for some  $\ell \in [d]$ . If  $\varphi$  can be verified on finite computations, then the system of size  $(1, \dots, 1)$  is a cutoff: for each  $(n_1, \dots, n_d) \geq (1, \dots, 1)$ , it holds  $(P^1, \dots, P^d)^{\mathbf{C}(n_1, \dots, n_d)} \models \varphi$  if and only if  $(P^1, \dots, P^d)^{\mathbf{C}(1, \dots, 1)} \models \varphi$ .

#### 6.4.4 DISJUNCTIVE GUARDS

Emerson and Kahlon [2000] obtained the following cutoff result for systems with disjunctive guards.

**Theorem 6.25** For number  $\ell \in \{1, 2\}$  and a specification class  $\mathcal{F}$  from  $1\text{-LTL} \setminus X(P^\ell)$ ,  $1\text{-ELTL} \setminus X(P^\ell)$ , or  $LTL \setminus X(C)$ , the problem  $\text{PMCP}(\mathcal{P}_{\text{disj}}[2], \mathbf{C}, \mathcal{F})$  is decidable.

Moreover, for every disjunctive guarded protocol  $(P^1, P^2)$ , there exists a cutoff of size  $(c_1, c_2)$ , that is for every formula  $\varphi \in \mathcal{F}$ , the following two statements are equivalent.

1. For every clique  $\mathbf{C}(n_1, n_2)$  it holds  $(P^1, P^2)^{\mathbf{C}(n_1, n_2)} \models \varphi$ .

## 90 6. GUARDED PROTOCOLS

2. For every clique  $\mathbf{C}(m_1, m_2)$  with  $m_1 \leq c_1$  and  $m_2 \leq c_2$ , it holds  $\overline{P}^{\mathbf{C}(m_1, m_2)} \models \varphi$ .

The cutoff size is defined by  $c_\ell = |Q^\ell| + 2$  and  $c_{3-\ell} = |Q^{3-\ell}| + 1$ .

To prove Theorem 6.25, we instantiate the bounding and monotonicity schemas. The most important part of the proof, however, is established in the “Unlocking” lemma that follows after the technical definition of a local computation of a process in a run of a system instance. In the following definition, we project a run on one process and remove stuttering introduced by the other processes.

**Definition 6.26** Let  $\pi = s_1, \dots, s_k$  be a run of a guarded system instance  $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$  and  $p$  be the index of a process of type  $\ell \in \{1, 2\}$ . Further, let  $i(1), \dots, i(m)$  be a sequence of numbers in  $[k]$  with the following properties:

- $i(1) = 1$ ;
- for all  $j : 1 \leq j < m$ ,  $i(j) < i(j+1)$  and  $(s_{i(j)}(p), s_{i(j+1)}(p)) \in \delta^\ell$ ; and
- for all  $j : 1 \leq j < m$ , for all  $h : 1 \leq h < i(j)$ , it holds  $(s_{i(j)+h}(p), s_{i(j)+h+1}(p)) \notin \delta^\ell$ .

We call the sequence  $s_{i(1)}(p), \dots, s_{i(m)}(p)$  a *local computation* of  $p$  in  $s_1, \dots, s_k$  and denote it as  $(s_1, \dots, s_k) \downarrow p$ .

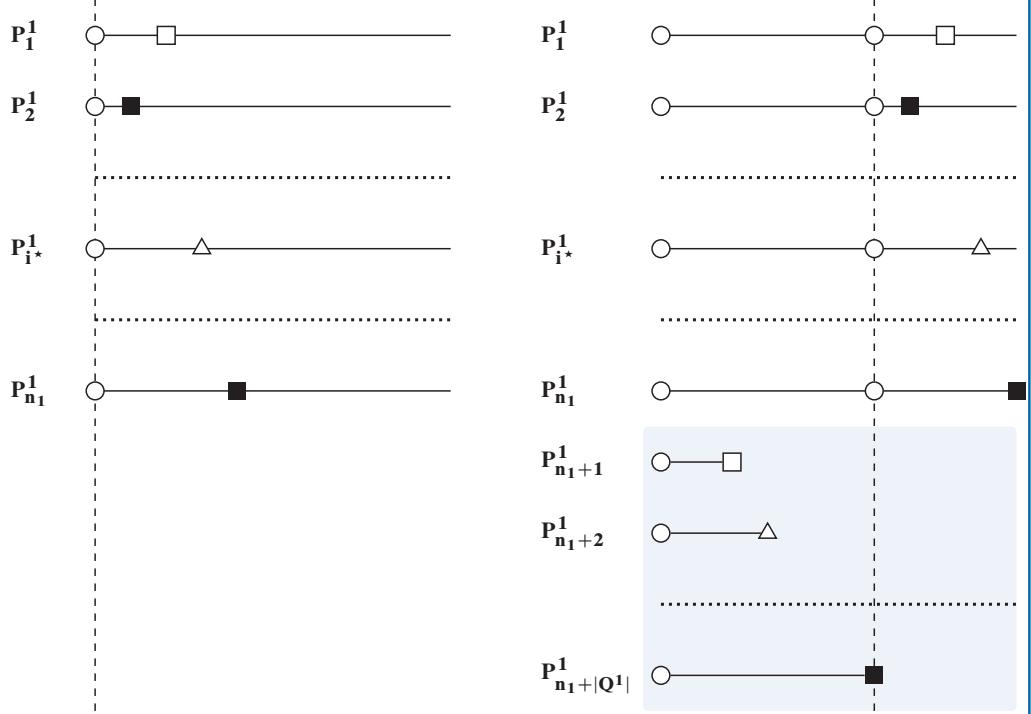
**Definition 6.27** Let  $\pi$  be run  $s_1 s_2 \dots$  of a guarded instance  $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$ . We define the set of local reachable states as  $Reach(\pi) = \{q \mid \exists i \geq 1, \exists p : 1 \leq p \leq n_1 + n_2, s_i(p) = q\}$ .

In the following lemma, we assign a unique process to each local state from  $Reach(\pi)$ . That local state is called a *goal state*.

**Lemma 6.28 Unlocking** Let  $\pi$  be a run  $s_1 s_2 \dots$  of a guarded system instance  $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$  that has only disjunctive guards. There exists a run  $\pi'$  of  $(P^1, P^2)^{\mathbf{C}(n_1 + |Q^1|, n_2 + |Q^2|)}$  organized as follows:  $\pi'$  has a prefix, where each of the at most  $|Q^1| + |Q^2|$  processes runs into its unique goal state from  $Reach(\pi)$  and remains there; the prefix is followed by transitions of  $n_1 + n_2$  processes as in  $\pi$ . The run  $\pi'$  is stuttering equivalent to  $\pi$  w.r.t. to  $n_1 + n_2$  processes.

**Proof.** We construct  $\pi'$  and then exploit that the transitions have only disjunctive guards to show that  $\pi'$  is a run. To this end, for each  $q \in Q^1 \cup Q^2$ , we define the distance of  $q$  from the initial state in  $\pi$  as  $dist(q) = \min\{i \mid \exists p : 1 \leq p \leq n_1 + n_2, s_i(p) = q\}$ , if  $q \in Reach(\pi)$ , and  $\infty$  otherwise.

Let  $q_1, \dots, q_k$  be the sequence of all local states from  $Reach(\pi)$ , ordered with respect to  $dist$ . Due to interleaving semantics, only one process takes a step at a time. Hence, for every  $q, q' \in Reach(\pi)$ ,  $q \neq q'$ , it holds  $dist(q) \neq dist(q')$ . Due to this fact, we can define a function  $first : Reach(\pi) \rightarrow [n_1 + n_2]$  that maps a reachable state  $q$  to the index of the process that performed the transition  $dist(q)$ . As all processes are initially either in  $init^1$  or  $init^2$ , we define  $first(init^1) = n_1$



**Figure 6.13:** Illustration of the unlocking lemma (Lemma 6.28) for the disjunctive guards. For simplicity, we left out the processes instantiated from the template  $P^2$ . Local states of the processes are depicted with  $\circ$ ,  $\square$ ,  $\blacksquare$ , and  $\triangle$ . The shaded area on the right highlights the processes added by the unlocking lemma. Each of these processes halts after reaching its designated state.

and  $\text{first}(\text{init}^2) = n_2$ . Note that  $\text{first}$  does not have to be an injection, that is, it maps several local states to the index of the same process, if the process is the first one to visit these states.

We construct a set of local computations  $\{\rho_i \mid 1 \leq i \leq k\}$  with  $\rho_i = (s_1, \dots, s_{\text{dist}(q_i)}) \downarrow \text{first}(q_i)$  for each  $i : 1 \leq i \leq k$ . Now we define a set of paths  $\{y_1^i, \dots, y_{|\rho_i|}^i \mid 1 \leq i \leq k\}$  that belong to the system instance  $(P^1, P^2)^{\mathbf{C}(n_1 + |Q^1|, n_2 + |Q^2|)}$ . For each  $i : 1 \leq i \leq k$ , each  $j : 1 \leq j \leq |\rho_i|$ , and each process index  $p : 1 \leq p \leq n_1 + |Q^1| + n_2 + |Q^2|$  we define a local state  $y_j^i(p)$  of process  $p$  as follows:

$$y_j^i(p) = \begin{cases} \rho_i[j] & \text{if } p = n_1 + n_2 + i, 0 \leq i \leq |\text{Reach}(\pi)| \\ \rho_i[|\rho_i|] & \text{if } \text{dist}(q_{i'}) < \text{dist}(q_i), p = n_1 + n_2 + i', 0 \leq i' \leq |\text{Reach}(\pi)| \\ \text{init}^{\text{type}(p)} & \text{otherwise.} \end{cases}$$

## 92 6. GUARDED PROTOCOLS

Then we extend the global states  $s_1, s_2, \dots$  with the halting states of the new  $|Q^1| + |Q^2|$  processes, that is, for each  $j \geq 1$ :

$$s'_j(p) = \begin{cases} s_j(p) & \text{if } p \leq n_1 + n_2 \\ \rho_i[|\rho_i|] & \text{if } p = n_1 + n_2 + i, 0 \leq i \leq |\text{Reach}(\pi)| \\ \text{init}^{\text{type}(p)} & \text{otherwise.} \end{cases}$$

Finally, we construct a sequence  $\pi'$  as a  $(y_1^1, \dots, y_{|\rho_1|}^1), \dots, (y_1^k, \dots, y_{|\rho_k|}^k), s'_1, s'_2 \dots$ . Now we show that  $\pi'$  is actually a run.

First, we show that the suffix  $s'_1, s'_2 \dots$  forms a path of  $(P^1, P^2)^{\mathbf{C}(n_1 + |Q^1|, n_2 + |Q^2|)}$ . As the transitions have only disjunctive guards, and the states of the processes with indices below  $n_1 + n_2$  remain as in  $\pi$ , the processes with the indices above  $n_1 + n_2$  do not disable the guards of the transitions made by the processes with the indices below  $n_1 + n_2$ . On the contrary, all the guards that are required to fire the transitions of  $\pi$  become enabled after executing the sequence  $(y_1^1, \dots, y_{|\rho_1|}^1), \dots, (y_1^k, \dots, y_{|\rho_k|}^k)$ . Formally, let  $g \equiv [\exists \text{ other } j] q_1 \vee \dots \vee q_m$  be a disjunctive guard,  $s_i$  a state on  $\pi$ , and  $p \leq n_1 + n_2$  a process index such that  $s_i \models_p g$ . Then there is a process index  $p' \leq n_1 + n_2$  and a local state  $q \in \{q_1, \dots, q_m\}$  with  $s_i(p') = q$ . As  $q \in \text{Reach}(\pi)$ , there is a process index  $p'' > n_1 + n_2$  with  $y_{n_k}^k(p'') = q$ , moreover, for all  $i \geq 1$ ,  $s'_i(p'') = q$ . Hence, for all  $i \geq 1$ ,  $s'_i \models_p g$ .

Second, we prove that the prefix  $(y_1^1, \dots, y_{|\rho_1|}^1), \dots, (y_1^k, \dots, y_{|\rho_k|}^k)$  is a run of  $(P^1, P^2)^{\mathbf{C}(n_1 + |Q^1|, n_2 + |Q^2|)}$ . As we define each  $\rho_i$  as a local computation, the sequent states are related with  $\delta^1$  or  $\delta^2$ . The guards in  $y_1^i, \dots, y_{|\rho_i|}^i$  are enabled, as we ordered  $q_1, \dots, q_k$  by  $\text{dist}$ , and  $\pi$  witnessed the shortest local computations  $\rho_1, \dots, \rho_k$ , leading to the local states in  $\text{Reach}(\pi)$ .

The run  $(y_1^1, \dots, y_{|\rho_1|}^1), \dots, (y_1^k, \dots, y_{|\rho_k|}^k)$  does not change the local states of the processes with the indices below  $n_1 + n_2$ . Thus,  $\pi'$  is stuttering equivalent to  $\pi$  w.r.t. the indices below  $n_1 + n_2$ .

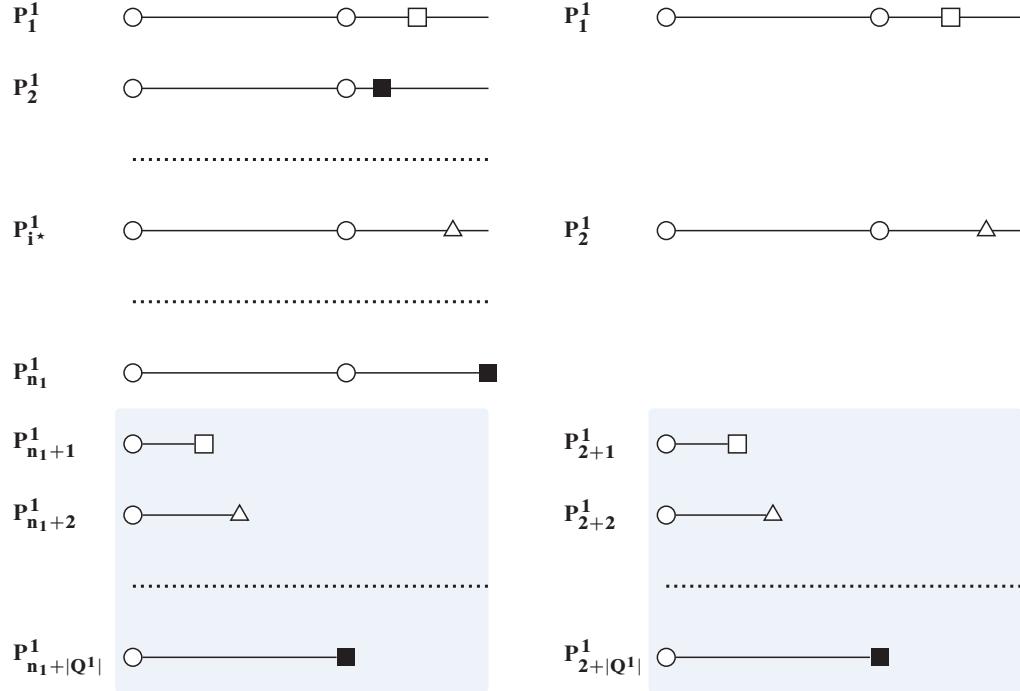
Hence,  $\pi'$  is the required run of  $(P^1, P^2)^{\mathbf{C}(n_1 + |Q^1|, n_2 + |Q^2|)}$ . □

Figure 6.13 illustrates application of the unlocking lemma.

We are now in the position to prove Theorem 6.25 according to the proof schema.

*Bounding schema:*

- i. Let  $\pi = s_1 s_2 \dots$  be a run of  $(P^1, P^2)^{\mathbf{C}(n_1, n_2)}$ . If  $\pi$  is a deadlock-free run, then let  $i^*$  be the index of a process that fires infinitely many transitions in  $\pi$ , otherwise  $i^*$  is an arbitrary index different from 1. Construct a stuttering equivalent run  $\pi'$  of  $(P^1, P^2)^{\mathbf{C}(n_1 + |Q^1|, n_2 + |Q^2|)}$  as stated in Lemma 6.28.
- ii. Construct an index projection  $pr_{idx}$  from  $\mathbf{C}(n_1 + |Q^1|, n_2 + |Q^2|)$  to  $\mathbf{C}(c_1, c_2)$  as follows:

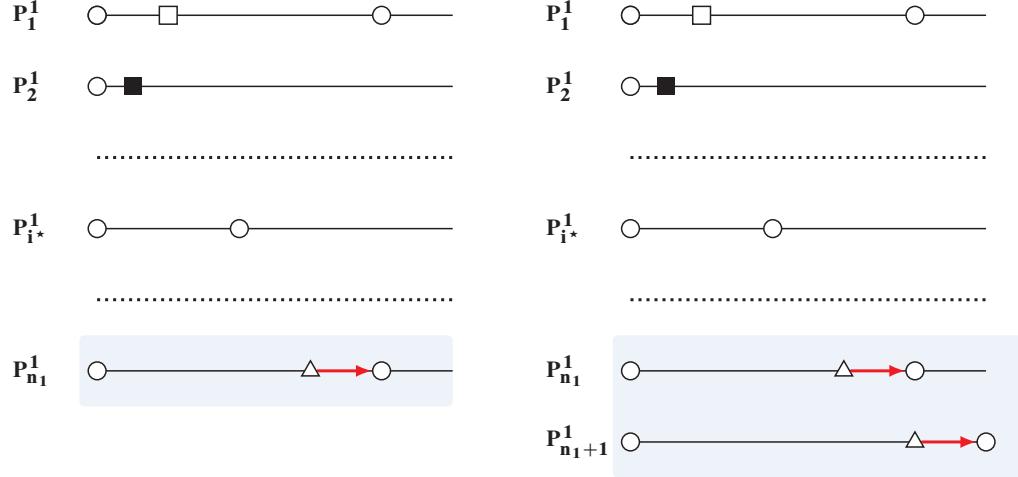


**Figure 6.14:** Illustration of the bounding schema applied together with the unlocking lemma (Lemma 6.28). For simplicity, we left out the processes instantiated from the template  $P^2$ . Local states of the processes are depicted with  $\circ$ ,  $\square$ ,  $\blacksquare$ , and  $\triangle$ . Process  $P_1^1$  is kept as is, because its behavior is observed by the specification. Process  $P_{i^*}^1$  is mapped to process  $P_2^1$ , as it makes infinitely many transitions (if the execution is deadlock-free). Every process  $P_{n_1+i}^1$  is mapped to process  $P_{2+i}^1$  for  $1 \leq i \leq |Q^1|$ .

$$pr_{idx}(i) = \begin{cases} 1 & \text{if } i = 1 \\ 2 & \text{if } i = i^* \\ 2 + (i - (n_1 + n_2)) & \text{if } n_1 + n_2 < i \leq n_1 + n_2 + |Q_1| + |Q_2| \\ \perp & \text{otherwise.} \end{cases}$$

- iii. As we keep the processes with the indices above  $n_1 + n_2$ , and they fire transitions before the processes with the indices below  $n_1 + n_2$ , their guards remain enabled. The guards of the transitions by the processes with the indices 1 and  $i^*$  are enabled by the local states of the

94 6. GUARDED PROTOCOLS



**Figure 6.15:** Illustration of the monononicity schema for disjunctive guards. For simplicity, we left out the processes instantiated from the template  $P^2$ . Local states of the processes are depicted with  $\circ$ ,  $\square$ ,  $\blacksquare$ , and  $\triangle$ . The additional process  $P^1_{n_1+1}$  mimics every transition made by the process  $P^1_{n_1}$ .

processes with the indices above  $n_1 + n_2$ , as constructed in Lemma 6.28. Hence,  $pr_{path}(\pi')$  is a run of  $(P^1, P^2)^{\mathbf{C}(c_1, c_2)}$ .

- iv. From stuttering equivalence of  $\pi$  and  $\pi'$  and Observation 6.18, we conclude that for every index  $k$  with  $pr_{idx}(k) \neq \perp$  it holds that  $(P^1, P^2)^{\mathbf{C}(n_1, n_2)} \models E\varphi(k)$  implies  $(P^1, P^2)^{\mathbf{C}(c_1, c_2)} \models E\varphi(pr_{idx}(k))$ .

Figure 6.14 illustrates application of the bounding schema together with the unlocking lemma.

**Remark 6.29** In the proof of Lemma 6.28, we move specific processes to cover all states in  $Reach$ . Thus, if the run  $\pi$  has a deadlock state  $s_d$ , it might happen that  $s'_d$  is not a deadlock state. To repair this, similar to Unlocking lemma, one can introduce a suffix that moves the processes with indices above  $n_1 + n_2$  out of  $Reach(\pi) \setminus \{q \mid \exists p. s_d(p) = q\}$  states.

*Monotonicity schema:*

- i. Fix system sizes  $(m_1, m_2)$  and  $(m_1 + d_1, m_2 + d_2)$  with  $d_1, d_2 \in \{0, 1\}$  and  $d_1 + d_2 = 1$ . Pick an arbitrary run  $\pi = t_1 t_2 \dots$  of  $(P^1, P^2)^{\mathbf{C}(m_1, m_2)}$ .
- ii. By  $N$  we abbreviate the index  $m_1 + m_2$ . Then  $N + 1 = m_1 + d_1 + m_2 + d_2$ . Map every pair  $t_i, t_{i+1}$  as follows.

Case 1. If  $(t_i, t_{i+1})$  is a transition by process with index  $N$ , then we map it on a triple  $s_i, s'_i, s_{i+1}$ , where  $\forall p : 1 \leq p < m_1 + m_2$ .  $s_i(p) = t_i(p)$ ,  $s'_i(p) = t_i(p)$ ,  $s_{i+1}(p) = t_{i+1}(p)$  and  $s_i(N) = t_i(N)$ ,  $s'_i(N) = s_{i+1}(N) = t_{i+1}(N)$  and  $s_i(N+1) = s'_i(N+1) = t_i(N)$ ,  $s_{i+1}(N+1) = t_{i+1}(N+1)$ . Informally, the process with index  $N+1$  copies the transition of the process with index  $N$ .

Case 2. If  $(t_i, t_{i+1})$  is a transition by process  $p \neq N$ , then we map it on a pair  $s_i, s_{i+1}$ , where  $\forall p : 1 \leq p \geq m_1 + m_2$ .  $s_i(p) = t_i(p)$ ,  $s_{i+1}(p) = t_{i+1}(p)$  and  $s_i(N+1) = t_i(N)$ ,  $s_{i+1}(N+1) = t_{i+1}(N)$ .

- iii. In case (ii.1), we have to show that for every guard  $g$  and every process index  $p < m_1 + m_2$ , if  $t_i \models_p g$  holds then  $s_i \models_p g$  and  $s'_i \models_p g$  also hold. Let  $g$  be  $[\exists \text{ other } q_1 \vee \dots \vee q_k]$ . From  $t_i \models_p g$  we conclude that there is a process  $p' \leq m_1 + m_2$  and  $p' \neq p$  with  $t_i(p') \in \{q_1, \dots, q_k\}$ . From (ii),  $s_i(p') = s'_i(p') = t_i(p')$ . Hence,  $s_i \models_p g$  and  $s'_i \models_p g$ . In case (ii.2), we immediately obtain that guards are preserved in every state  $s_i$ .
- iv. Apply Observation 6.18 to conclude that for every index  $k$  with  $pr_{idx}(k) \neq \perp$  it holds that  $(P^1, P^2)^{\mathbf{C}(m_1, m_2)} \models E\varphi(pr_{idx}(k))$  implies  $(P^1, P^2)^{\mathbf{C}(m_1+d_1, m_2+d_2)} \models E\varphi(k)$ .

Figure 6.15 illustrates application of the monotonicity schema.

The unlocking lemma, the bounding schema, and the monotonicity schema applied together finish the proof of Theorem 6.25.

**Further Results.** Emerson and Kahlon [2000] also extended the cutoff result on two index specifications:

**Theorem 6.30** *Let  $\ell$  and  $\ell'$  be process types and  $\mathcal{F}$  be one of 2-LTL\X( $P^\ell, P^{\ell'}$ ) and 2-ELTL\X( $P^\ell, P^{\ell'}$ ). The problem PMCP( $\mathcal{P}_{\text{disj}}$ ,  $\mathbf{C}$ ,  $\mathcal{F}$ ) is decidable. Moreover, there is a cutoff of size  $(c_1, \dots, c_d)$ , where  $c_\ell = |Q^\ell| + 1$  and  $c_{\ell'} = |Q^{\ell'}| + 1$  and for all other  $k \in [d] \setminus \{\ell, \ell'\}$ ,  $c_k = |Q^k|$ .*

The estimates of cutoffs in Theorems 6.25 and 6.30 require model checking of quite large systems containing at least  $|Q^\ell|$  processes of each type  $\ell$ . Emerson and Kahlon [2000] gave an efficient solution for universal-path-quantified formulas. We do not give details here.

Emerson and Kahlon [2003b] noted that the PMC problem for protocols with disjunctive guards is decidable for regular and  $\omega$ -regular properties over actions.

**Theorem 6.31** *PMCP( $\mathcal{P}_{\text{disj}}$ ,  $\mathbf{C}$ ,  $\mathcal{F}$ ) is decidable for  $\mathcal{F} = \text{Reg}(A)$  and  $\mathcal{F} = \omega\text{Reg}(A)$ .*

**Proof Sketch.** The intuition is that the behavior of all the systems instantiated from a disjunctive system template  $(P^1, \dots, P^d)$  can be encoded as a Petri net  $\mathcal{P}$ . The language of  $\mathcal{P}$  is checked for intersection with the specification, i.e., the language of either a Büchi automaton, or a finite automaton over actions. This problem is known to be decidable [Esparza and Nielsen, 1994, Theorem 8].

## 96 6. GUARDED PROTOCOLS

The constructed Petri net  $\mathcal{P}$  has one place per local state from  $Q^1 \cup \dots \cup Q^d$ ; the number of tokens in a place  $q$  encodes the number of processes in local state  $q$ . If a process makes a transition from one local state to another, then the number of tokens in the source place and the target places are decremented and incremented, respectively. A disjunctive guard  $[\exists \text{ other } i] q_1 \vee \dots \vee q_k$  is then encoded as  $k$  transitions—one per local state  $q_i$ —fetching a token from  $q_i$  and then returning it back to  $q_i$ .  $\square$

## 6.5 DISJUNCTIVE GUARDS VS. RENDEZVOUS

In this section we consider the relation between systems with disjunctive guards and pairwise rendezvous. We review results by [Emerson and Kahlon \[2003b\]](#) who showed that the systems with disjunctive guards and the systems with pairwise rendezvous are reducible to each other in the sense that the reduction preserves properties from  $\text{Reg}(A)$  and  $\omega\text{Reg}(A)$  (formally defined below).

For  $\text{LTL} \setminus X(C)$  properties, we show that the given reductions (designed for  $\text{Reg}(A)$  and  $\omega\text{Reg}(A)$ ) work only in one direction: the provided reduction from disjunctive guards to pairwise rendezvous preserves  $\text{LTL} \setminus X(C)$  properties. However, for the other direction, the provided reduction from pairwise rendezvous to disjunctive guards is not conservative with respect to  $\text{LTL} \setminus X(C)$ .

First, we introduce the notions of  $\xi$ -reducibility ([Emerson and Kahlon \[2003b\]](#) denoted it via  $\prec_\phi$ ) and a special case of stuttering trace equivalence. Let us fix AP and two LTSs  $M_i = (Q_i, Q_i^0, \Sigma_i, \delta_i, \lambda_i)$  for  $i = 1, 2$ . Furthermore, let  $\xi : \Sigma_1 \rightarrow \Sigma_2^+$  be an action mapping; as common we extend  $\xi$  to a language  $L \subseteq \Sigma_1^* \cup \Sigma_1^\omega$  as  $\xi(L) = \{\xi(w) : w \in L\}$ .

We say that  $M_1$  is  $\xi$ -reducible to  $M_2$  if for every regular language  $L \subseteq \Sigma_1^*$  and every  $\omega$ -regular language  $L_\omega \subseteq \Sigma_1^\omega$  the following holds.

- Properties of *finite* action-labeled runs of  $M_1$  and  $M_2$  are indistinguishable by  $L$  and  $\xi(L)$ , i.e.,  $\mathcal{L}(M_1) \cap L = \emptyset$  iff  $\mathcal{L}(M_2) \cap \xi(L) = \emptyset$ .
- Properties of *infinite* action-labeled runs of  $M_1$  and  $M_2$  are indistinguishable by  $L_\omega$  and  $\xi(L_\omega)$ , i.e.,  $\xi(\mathcal{L}_\omega(M_1)) \cap L_\omega = \emptyset$  iff  $\mathcal{L}_\omega(M_2) \cap \xi(L_\omega) = \emptyset$ .

Thus, if  $M_1$  is  $\xi$ -reducible to  $M_2$ , then one can reduce model checking of an action-based property of  $M_1$ , which is using one kind of synchronization primitives, to model checking of the corresponding property of  $M_2$ , which is using another set of synchronization primitives. Note carefully that  $\xi$ -reducibility does not require the languages of  $M_1$  and  $M_2$  to be equivalent (up to  $\xi$ ), hence, the language of  $M_2$  may contain action runs that are not in  $\xi(\Sigma_1)$ .

Furthermore, for a (finite or infinite state-labeled) path  $\pi : i \mapsto q_i$  we call the sequence  $tr(\pi) : i \mapsto \lambda(q_i)$  the *trace* of  $\pi$ . Let  $stutter_k$  be the function defined as  $stutter_k(w_1 w_2 \dots) = (w_1)^k (w_2)^k \dots$  for any word  $w_1 w_2 \dots \in AP^* \cup AP^\omega$ , i.e., every letter of the word is repeated  $k$  times. We say that  $M_1$  is  $k$ -equivalent to  $M_2$ , if the traces of the systems coincide up to proposition stuttering, i.e.,  $\{stutter_k(tr(\pi)) : \pi \text{ is a run of } M_1\} = \{tr(\pi') : \pi' \text{ is a run of } M_2\}$ .

One can see that  $k$ -equivalence implies stutter trace equivalence. Thus, using the well-known fact about preservation of  $\text{LTL} \setminus X$  properties by stutter trace equivalent systems (cf. Baier and Katoen [2008, Theorem 7.92]), we obtain: *For  $k$ -equivalent LTSs  $M_1$  and  $M_2$  and for  $\text{LTL} \setminus X$ -formula  $\psi$ , the following holds:  $M_1 \models \psi$  iff  $M_2 \models \psi$ .*

Now we prove an interesting fact: Every system with disjunctive guards is  $\xi$ -reducible and 2-equivalent to a system with pairwise rendezvous (shown by Emerson and Kahlon [2003b, Section 5]):

**Theorem 6.32** *For every guarded system template  $(P^1, \dots, P^d)$  there exists a system template  $(\tilde{P}^1, \dots, \tilde{P}^d)$  using pairwise rendezvous such that for every clique  $C(n)$ , the system instance  $(P^1, \dots, P^d)^{C(n)}$  is  $\xi$ -reducible and 2-equivalent to the system instance  $(\tilde{P}^1, \dots, \tilde{P}^d)^{C(n)}$ .*

**Proof.** Let every process template  $P^\ell$  be  $(Q^\ell, Q_0^\ell, \Sigma^\ell, \delta^\ell, \lambda^\ell)$ . Following Emerson and Kahlon [2003b], we assume that the set of actions  $\Sigma_{pr}$  is a disjoint union  $\delta^1 \cup \dots \cup \delta^d$ , and every transition is labeled with a unique action from  $\Sigma_{pr}$ . For every  $\ell \in [d]$ , we construct a process template  $\tilde{P}^\ell = (\tilde{Q}^\ell, \tilde{Q}_0^\ell, \tilde{\Sigma}^\ell, \tilde{\delta}^\ell, \tilde{\lambda}^\ell)$  as follows.

- Set of states  $\tilde{Q}^\ell$  is the disjoint union of the original states  $Q^\ell$  and of a set of auxiliary states defined by  $\{\text{syn}(t) : t \in \delta^\ell\} \cup \{\text{cosyn}(c, t) : c \in Q^1 \cup \dots \cup Q^d \text{ and } t \in \delta^\ell\}$ .
- Initial states stay the same:  $\tilde{Q}_0^\ell = Q_0^\ell$ .
- State labels  $\tilde{\delta}^\ell$  are assigned as follows: Original states  $q \in Q^\ell$  are labeled with  $\{q\}$ ; Synchronization states  $\text{syn}((q, a, q')) \in Q^\ell$  are labeled with their predecessor  $\{q\}$ ; Co-synchronization states  $\text{cosyn}(c, t) \in Q^\ell$  are labeled with their predecessor  $\{c\}$ .
- $\tilde{\Sigma}^\ell = \{req(t)! : t \in \delta^\ell\} \cup \{ack(t)? : t \in \delta^\ell\} \cup \{ack(t)! : t \in \bigcup_{k \in [d]} \delta^k\} \cup \{req(t)? : t \in \bigcup_{k \in [d]} \delta^k\}$ .
- For every transition  $t = (q, a, q')$  from  $\delta^\ell$  one adds to  $\tilde{\delta}^\ell$  two transitions  $(q, req(t)!, \text{syn}(t))$  and  $(\text{syn}(t), ack(t)?, q')$ . Furthermore, for every proposition  $c_i$  of the disjunctive guard  $gd(t) = [\exists \text{ other } j] c_1 \vee \dots \vee c_k$ , one adds two transitions  $(c_i, req(t)?, \text{cosyn}(t))$  and  $(\text{cosyn}(t), ack(t)!, c_i)$  to the corresponding transition relation  $\tilde{\delta}^\ell$  of the process template containing the state  $c_i$ .

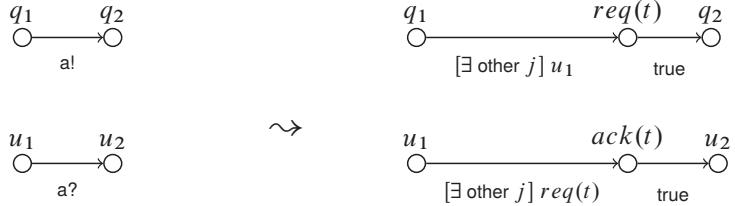
It is easy to see that every transition  $t = (q, a, q')$  of  $P^\ell$  is translated to a sequence of transitions  $q \xrightarrow{\text{req}(t)!} \text{syn}(t) \xrightarrow{\text{ack}(t)?} q'$  made by a process of type  $\ell$ ; this sequence is synchronized with a sequence  $c_i \xrightarrow{\text{req}(t)?} \text{cosyn}(c_i, t) \xrightarrow{\text{ack}(t)!} c_i$  executed by a process residing in state  $c_i$ . The intermediate state  $\text{cosyn}(c_i, t)$  stores its predecessor  $c_i$  to return back to it later.

By construction, the intermediate states  $\text{syn}(t)$  and  $\text{cosyn}(c_i, t)$  preserve the labels of their predecessors. Thus, every letter in a trace of  $(P^1, \dots, P^d)^{C(n)}$  is doubled in  $(\tilde{P}^1, \dots, \tilde{P}^d)^{C(n)}$ .

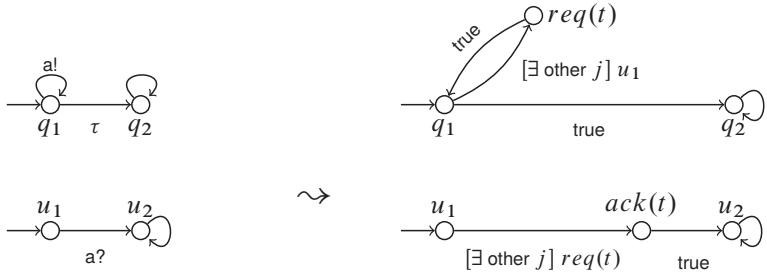
## 98 6. GUARDED PROTOCOLS

Define a mapping  $\xi(t)$  as  $req(t)! \cdot ack(t)?$ . Now it is easy to see that every system instance  $(P^1, \dots, P^d)^{\mathbf{C}(n)}$  is  $\xi$ -reducible and 2-equivalent to the system instance  $(\tilde{P}^1, \dots, \tilde{P}^d)^{\mathbf{C}(n)}$ .  $\square$

**Emerson and Kahlon [2003b, Section 5]** showed a reduction of systems with pairwise rendezvous to systems with disjunctive guards, while preserving  $\xi$ -reducibility.



**Figure 6.16:** Reducing *with caveat* pairwise rendezvous (left) to transitions with disjunctive guards (right).



**Figure 6.17:** An example showing that the reduction on Figure 6.16 does not preserve formulas from  $LTL \setminus X(C)$ .

**Theorem 6.33** *For each system template  $(P^1, \dots, P^d)$  with pairwise rendezvous there is a guarded protocol  $(\tilde{P}^1, \dots, \tilde{P}^d)$  with disjunctive guards having the following property: For every clique  $\mathbf{C}(n)$ , system instance  $(P^1, \dots, P^d)^{\mathbf{C}(n)}$  is  $\xi$ -reducible to system instance  $(\tilde{P}^1, \dots, \tilde{P}^d)^{\mathbf{C}(n)}$ .*

**Proof idea.** The proof similar to the proof of Theorem 6.32. Figure 6.16 shows how a pair of synchronized transitions  $t = (q_1, a!, q_2)$  and  $(u_1, a?, u_2)$  is translated to four transitions with disjunctive guards. The intermediate states  $req(t)$  and  $ack(t)$  keep the labels of their predecessors, i.e., of  $q_1$  and  $u_1$ . Given process templates, such a translation is done for every pair of transitions carrying synchronization actions of the same type.

Let  $b, c, d$ , and  $e$  be the actions assigned to the transitions  $q_1 \rightarrow req(t)$ ,  $req(t) \rightarrow q_2$ ,  $u_1 \rightarrow ack(t)$ ,  $ack(t) \rightarrow u_2$ , respectively. Then the mapping  $\xi$  of  $a!$  is defined as  $\xi(a!) = bcde$ .

Thus, a finite (or Büchi) automaton checking a specification has to recognize the sequence  $bcd e$  instead of letter  $a!$ , which corresponds to handshake of two processes issuing actions  $a!$  and  $a?$ .  $\square$

In contrast to action-based properties, state-based properties are not preserved by the reduction in the proof of Theorem 6.33.

**Observation 6.34** The reduction provided in the proof of Theorem 6.33 does not preserve  $LTL \setminus X(C)$  properties.

*Proof.* Figure 6.17 shows a system template  $(C, U)$  with rendezvous (left-hand side) and its reduction to a guarded protocol  $(C', U')$  with disjunctive guards (righthand-side). Fix an arbitrary  $n \in \mathbb{N}$ . For the system with rendezvous, it holds that  $(C, U)^{C(1,n)} \models F q_2$ , as  $n$  processes of type  $U$ , allow  $C$  to fire  $(q_1, a!, q_1)$  finitely many times, that is, every user process moves from  $u_1$  to  $u_2$  once. In the system with disjunctive guards,  $C'$  can fire  $(q_1, [\exists \text{ other } j] u_1, \text{req}(t))$  infinitely often, without a single process of type  $U'$  moving. Hence,  $(C', U')^{C(1,n)} \not\models F q_2$ .  $\square$

## 6.6 VARIATIONS ON THE MODEL: GUARDS AND SYNCHRONIZATION PRIMITIVES

The protocols we considered so far had guarded internal transitions only. Emerson and Kahlon [2003a,b,c] investigated the cases of guarded systems with broadcast actions, pairwise rendezvous, and asynchronous rendezvous. These combined systems allow one to model, e.g., cache coherence protocols.

Most of the results for the combined models are obtained using reduction techniques by Emerson and Kahlon [2003b]. The results are summarized in Figure 6.1. Here we briefly discuss the ideas behind the results. Most of the proofs are based on the undecidability results for (rendezvous and broadcast) systems without guards, and the reduction provided in Section 6.5.

**Disjunctive guards and rendezvous  $\mathcal{P}_{\text{disj\&rdvz}}$ .** Decidability of the PMCP for  $\mathcal{P}_{\text{disj\&rdvz}}$  (disjunctive guards and pairwise rendezvous) depends on the specification class. Decidability for  $\text{Reg}(A)$  is given by Emerson and Kahlon [2003b]; the result relies on backward reachability in well-structured systems, which is decidable [Abdulla et al., 1996]. Further, as the systems with disjunctive guards are reducible to systems with pairwise rendezvous (see Theorem 6.32) one can re-use the decidability proof for rendezvous by German and Sistla [1992] (cf. Section 5). Namely, the PMCP is also decidable for the classes of  $LTL \setminus X(C)$ ,  $LTL(C)$ ,  $1-LTL \setminus X(P^\ell)$ , and  $\omega\text{Reg}(A)$  properties.

**Disjunctive guards and broadcast  $\mathcal{P}_{\text{disj\&bcast}}$ .** In the case of  $\mathcal{P}_{\text{disj\&bcast}}$  (disjunctive guards and broadcasts), none of the specification classes mentioned in this section are decidable except  $\text{Reg}(A)$ . As disjunctive guards are reducible to pairwise rendezvous, and systems with pairwise rendezvous and broadcasts are reducible to systems with broadcasts (see Section 5 and Esparza et al. [1999]), we can reduce systems in  $\mathcal{P}_{\text{disj\&bcast}}$  to systems with broadcast. As Esparza et al. [1999] showed,

## 100 6. GUARDED PROTOCOLS

the PMCP is undecidable for all classes except  $\text{Reg}(A)$ ; in the case of  $\text{Reg}(A)$ , the construction of a coverability graph by [Emerson and Namjoshi \[1998\]](#) applies.

**Disjunctive guards and asynchronous rendezvous**  $\mathcal{P}_{\text{disj}\&\text{ardvz}}$ . The PMCP for  $\mathcal{P}_{\text{disj}\&\text{ardvz}}$  (disjunctive guards and asynchronous rendezvous) is also undecidable, except for the case of  $\text{Reg}(A)$ , which is similar to the result for  $\mathcal{P}_{\text{disj}\&\text{rdvz}}$ . As shown by [Emerson and Kahlon \[2003b\]](#), the PMCP of  $\text{LTL}\backslash X(C)$  properties is undecidable for systems with both pairwise rendezvous and asynchronous rendezvous. Due to this and reducibility of disjunctive guards to pairwise rendezvous, it immediately follows that the PMCP is undecidable for all specification classes that include  $\text{LTL}\backslash X(C)$ .

**Conjunctive guards and synchronization.** Disjunctive guards are reducible to pairwise rendezvous (see [Theorem 6.32](#)). Thus, PMCP for  $\mathcal{P}_{\text{conj+disj}}$  (in other words, PMCP for boolean guards,  $\mathcal{P}_{\text{bool}}$ ) can be reduced to PMCP for  $\mathcal{P}_{\text{conj+rdvz}}$  (conjunctive guards and pairwise rendezvous). As PMCP is undecidable for boolean guards and every class of specifications (see [Theorem 6.13](#)), PMCP for  $\mathcal{P}_{\text{conj+rdvz}}$  and every class of specifications is undecidable as well.

Furthermore, as shown by [Emerson and Kahlon \[2003b\]](#), pairwise rendezvous is reducible to asynchronous rendezvous. As we have just shown that PMCP for  $\mathcal{P}_{\text{conj+rdvz}}$  is undecidable for every class of specifications, it follows that PMCP for  $\mathcal{P}_{\text{conj+ardvz}}$  (conjunctive guards and asynchronous rendezvous) is also undecidable for every class of specifications. Similarly, PMCP is undecidable for  $\mathcal{P}_{\text{conj+bcast}}$  (conjunctive guards and broadcasts). Again, this is due to reducibility of disjunctive guards to pairwise rendezvous and of pairwise rendezvous to broadcast [[Esparza et al., 1999](#)]. Together with the undecidability results for boolean guards, this shows undecidability for all considered classes of specifications.

## 6.7 DISCUSSION

In this chapter, we introduced the results on guarded protocols in the unified framework. This allows us to present different computational models from the literature as instances of our definitions. We re-structure the proofs of the cutoff theorems for disjunctive and init-conjunctive guards, which highlight central arguments of these important results. [Emerson and Kahlon \[2003b\]](#) claimed in their discussions that there is a decidability result for conjunctive guards. However, the reference they give is based on init-conjunctive guards [[Emerson and Kahlon, 2000](#)]. As the proofs use the fact that an init-conjunctive guard cannot be disabled by a process at the initial state, it is not straightforward to generalize these proofs to conjunctive guards. Hence, to the best of our understanding, decidability for system with *conjunctive guards* and specifications from  $1\text{-LTL}\backslash X(P^\ell)$ ,  $1\text{-ELTL}\backslash X(P^\ell)$ , or  $\text{LTL}\backslash X(C)$  are open problems.

We clarified that the reduction of disjunctive guards to rendezvous presented in [Emerson and Kahlon \[2003b\]](#) is limited to action-based specifications. In particular, we showed that disjunctive guards are not powerful enough to capture rendezvous with respect to  $\text{LTL}(C)$  properties.

Guarded protocols are interesting not only from a theoretical point of view, but are also useful in modeling cache coherence protocols—an important concept for hardware designs. The

following papers discuss applications of guarded protocols in modeling and verification of cache coherence protocols. There are two notable papers by [Emerson and Kahlon \[2003a,c\]](#) showing how guarded protocols with broadcasts can be applied to modeling of cache coherence protocols. In the mentioned papers, the authors consider restrictions of guarded systems with one process template  $U$ .

- The conjunctive guards have the form  $[\forall \text{ other } i] init$ .
- The disjunctive guards have the form  $[\exists \text{ other } i] \neg init$ .
- Apart from guards, processes use broadcasts.
- The transitions of process templates have various structural restrictions, e.g., from every state the process is able to go to  $init$ .

[Emerson and Kahlon \[2003a,c\]](#) modeled eight cache coherence protocols with the framework of guarded systems. As the PMCP for guarded systems with broadcast is undecidable, the authors had to develop special abstraction and cutoff techniques for the protocols with the restrictions mentioned above. As the results seem to be considerably tailored to the special structure of cache coherence protocols, we do not give details in this survey. An interested reader finds a detailed exposition in the original papers.

Some lower bounds are known for disjunctively guarded systems: PMCP is undecidable for 1-index  $\text{CTL}^*\backslash X$  specifications [Aminof et al. \[2014b\]](#); for systems with or without a controller the complexity of the PMCP is PSPACE-complete (the upper bound is from [Emerson and Kahlon \[2000\]](#) and the lower bound is from [Aminof et al. \[2014b\]](#)); for systems with a controller the program complexity (i.e., the formula is fixed and not part of the input) is coNP-complete [Aminof et al. \[2014b\]](#), whereas for systems without a controller the program complexity is in PTIME (the upper bound is from [Emerson and Kahlon \[2000\]](#) and the lower bound is from [Aminof et al. \[2014b\]](#)).

From a theoretical point of view, we observe from the discussions in this chapter that there are the following open problems.

**Problem 6.35** Is the PMCP with conjunctive guards and the following specifications decidable:  $1\text{-LTL}\backslash X(P^\ell)$ ,  $2\text{-LTL}\backslash X(P^\ell, P^{\ell'})$ ,  $LTL(C)$ ? Is there a cutoff, similar to Theorem 6.22?

**Problem 6.36** Is the PMCP with disjunctive guards and the following specifications decidable: prenex  $ILTL\backslash X$ , prenex  $ICTL^*\backslash X$ ,  $ILTL\backslash X$ ?



## CHAPTER 7

# Ad Hoc Networks

In ad hoc networks, processes communicate via broadcasts. Recall from Section 2.2.1 that in transitions synchronized by broadcast, all immediate neighbors of a sender simultaneously take a transition with the sender, and the sender is not blocked if there are no recipients that are ready to synchronize. This chapter summarizes the results by [Delzanno et al. \[2010, 2011, 2012b\]](#) and [Abdulla et al. \[2013a\]](#).

The literature studies ad hoc networks arranged in different classes of connectivity graphs that try to model typical topologies met in practice (see Section 7.4), and also networks in which broadcast messages can be lost and thus not received by some of the sender's neighbors. In case of (non-lossy) ad hoc networks (AHN), parameterized model checking problems, even for safety properties, are undecidable for systems with general connectivity graphs, but become decidable for certain classes of graphs. In case of lossy ad hoc networks (LAHN), the additional non-determinism of whether or not a message is lost for a receiving process makes all parameterized model checking problems considered in this chapter decidable. However, it is well known that important problems cannot be solved in the presence of lossy links, e.g., the two generals paradox [\[Gray, 1978\]](#). A related model is that of *mobile* ad hoc networks, in which the structure of the communication graph can change during execution. It has been shown by [Delzanno et al. \[2012b\]](#) that such networks are equivalent to a certain class of lossy ad hoc networks. Generally, the models discussed in this section, though inspired by ad hoc networks (cf. [Karl \[2005\]](#)), have not yet found their practical applications, so that in the next section we provide a toy example to help us to illustrate the theoretical results.

### 7.1 RUNNING EXAMPLE

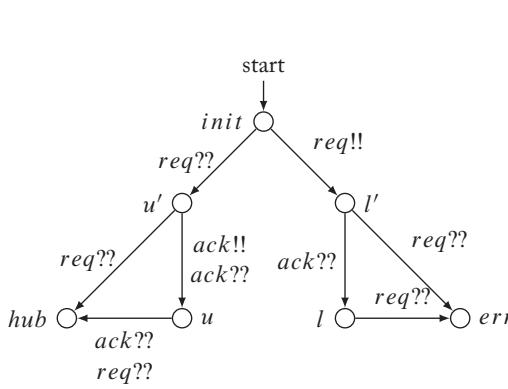
Consider the process template in Figure 7.1. Processes start in the initial state *init*. Processes that are in state *u* will be called “user” processes, processes in *l* are “leader” processes, processes in *hub* are “hub” processes, and processes in *err* are “error” processes. Figure 7.2 gives two connectivity graphs with reachable global states of the system in AHNs or LAHNs, respectively. For AHNs composed of processes running this protocol, the following holds:

- (i) in AHNs arranged on any undirected graph, state *err* is unreachable;
- (ii) in AHNs arranged in cliques, state *hub* is unreachable; and
- (iii) in AHNs arranged on any undirected graph, any process adjacent to a leader is not a leader (and will eventually be either a hub or a user process).

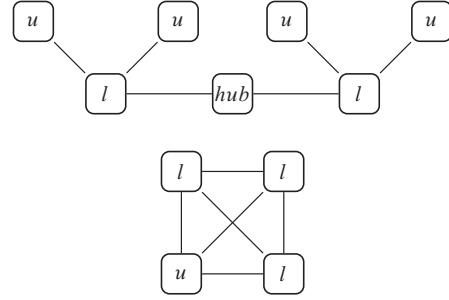
## 104 7. AD HOC NETWORKS

Consider the first item. Let process  $p_{err}$  be one of the processes that reach state  $err$  first (hypothetically, there may be several processes that reach  $err$  at the same time). Let us assume that  $p_{err}$  reaches  $err$  via transition  $l' \xrightarrow{req??} err$ . Also, wlog. assume  $p_{err}$  is the first among the error processes that reaches  $l'$ . Just before getting into  $l'$  the process sends  $req!!$  which makes it impossible for any adjacent process to send  $req!!$ . The last fact implies that  $p_{err}$  cannot reach  $err$  from  $l'$ . Contradiction. In a similar way, we can derive a contradiction for the case of reaching  $err$  via transition  $l \xrightarrow{req??} req$ . Later in this chapter, in Example 7.20 we prove this formally by applying an algorithm that can answer reachability questions of this kind.

The items (i)–(iii) do not hold for LAHNs, namely: on cliques, states  $err$  and  $hub$  are reachable, and leader processes can be adjacent. Later in this chapter, Example 7.22 proves item (i) adapted to LAHNs on cliques formally. An example of the protocol execution that falsifies (iii) for LAHNs is in Figure 7.2.



**Figure 7.1:** Process template for the running example.



**Figure 7.2:** On top: reachable global state after executing the protocol in an AHN. Processes are labeled with their local state. On the bottom: reachable global state after executing the protocol in a LAHN.

## 7.2 SYSTEM MODEL

**Process and system template.** The only communication primitive allowed is a broadcast as defined in Section 2.2.1, i.e., the synchronization constraint is  $\text{card} = \mathbb{N}_0$ . In the literature, only systems with a single process template  $P$  are considered, i.e.,  $d = 1$ . Denote by  $\mathcal{P}_{\text{bcast}}$  the set of all process templates that communicate via broadcasts.

**1-ary parameterized connectivity graph  $\mathbf{G}$ .** In this chapter, we consider 1-ary parameterized connectivity graphs  $\mathbf{G}$  as a *set* of 1-colored connectivity graphs, rather than a sequence as defined in Section 2.2. This is simply a matter of convenience, as the order of graphs will not matter.

AHN system instance  $P^G$ . The definition of AHN system instance  $P^G$  follows that of Section 2.2.

**LAHN system instance  $P_{lossy}^G$ .** To define LAHN system instance we introduce a *lossy broadcast transition* by dropping the (MAX) requirement from the definition in Section 2.2.1. Recall that the (MAX) requirement forces all processes that can take a “receive” broadcast transition to take it. Dropping the (MAX) requirement allows a process to “ignore” it: for every process that is a recipient of the initiator and can take a transition labeled with some  $b??$ , whether it actually takes it is chosen non-deterministically. Informally, this may be viewed as a “message loss.” Denote by  $P_{lossy}^G$  a system instance with all broadcast transitions being lossy.

**Mobile AHNs.** In *mobile ad hoc networks*, the underlying connectivity graph can change during a run of the system. We will not include such changes of the connectivity graph into our system model. However, it has been shown by [Delzanno et al., 2012b, Proposition 1] that mobile ad hoc networks are equivalent to LAHNs on cliques, and therefore the decidability results for LAHNs on cliques also apply to mobile ad hoc networks.

**Example 7.1** Figure 7.1 illustrates a process template from  $\mathcal{P}_{\text{bcast}}$ . Figure 7.2 gives global states that can be reached in an AHN and a LAHN, respectively, after execution of the protocol.

**Deadlocks.** In this chapter, a run is a finite or infinite maximal path that starts in the initial state, i.e., we handle finite paths directly and neither extend nor ignore deadlocked system runs. Note that the problem of ensuring the absence of deadlocks in parameterized AHNs is undecidable already for AHNs on cliques; this can be proven using a modified construction from Theorem 5.11. In contrast, this problem for LAHNs and AHNs is decidable for parameterized cliques and general graphs, since we can reduce it to the deadlock detection problem in Petri nets, which is decidable [Cheng et al., 1993].

### 7.3 PMC PROBLEMS FOR AD HOC NETWORKS

For ad hoc networks, three special PMC problems were studied in the literature—COVER, REPEAT, and TARGET.

- **COVER<sub>G</sub>**, or *control state reachability*:

**input:** process template  $P = (Q, Q_0, \Sigma, \delta, \lambda) \in \mathcal{P}_{\text{bcast}}$ , control states  $C \subseteq Q$

**output:** “Yes” if there exists  $G = (V, E) \in \mathbf{G}$  such that system  $P^G$  has a run  $s_0, s_1, \dots$  (finite or infinite) where for some process index  $i \in V$  and some state  $s_n$  of the run:  $s_n(i) \in C$ . That is, there is a system where some process reaches a control state. “No” otherwise.

- **REPEAT<sub>G</sub>**, or *repeated control state reachability*:

**input:** process template  $P = (Q, Q_0, \Sigma, \delta, \lambda) \in \mathcal{P}_{\text{bcast}}$ , control states  $R \subseteq Q$

**output:** “Yes” if there exists  $G = (V, E) \in \mathbf{G}$  such that  $P^G$  has an infinite run  $\sigma = s_0, s_1, \dots$

## 106 7. AD HOC NETWORKS

where for some process index  $i \in V$  the set  $\{k \mid s_k(i) \in R\}$  is infinite. That is, there is a system where some process can visit a control state infinitely often. “No” otherwise.

- **TARGET<sub>G</sub>**, or *target reachability*:

**input:** process template  $P = (Q, Q_0, \Sigma, \delta, \lambda) \in \mathcal{P}_{\text{bcast}}$ , control states  $T \subseteq Q$

**output:** “Yes” if there exists  $G = (V, E) \in \mathbf{G}$  such that system  $P^G$  has a run  $\sigma = s_0, s_1, \dots$  (finite or infinite) where for all process indices  $i \in V$   $s_n(i) \in T$ . That is, there is a system that reaches a global state with all processes being in a control state (the control state may be different for different processes). “No” otherwise.

Under certain conditions (for example, in systems where all runs infinite), these PMC problems can be expressed in indexed temporal logic<sup>1</sup>. In particular:

- any instance of **COVER<sub>G</sub>** can be expressed as a negation of an instance of the PMCP with respect to a 1-indexed safety property;
- any instance of **REPEAT<sub>G</sub>**—with respect to a universal 1-index persistence property; and
- any instance of **TARGET<sub>G</sub>**—with respect to a safety property in non-prenex fragment.

The PMC problems for LAHNs are defined similarly, except that  $P^G$  is replaced with  $P_{\text{lossy}}^G$ . Delzanno et al. [2012b] studied PMC problems for LAHN arranged in cliques. We write  $P_{\text{lossy}}^{\mathbf{C}}$  for such parameterized systems, where  $\mathbf{C}$  is a parameterized clique. To the best of our knowledge, the case of LAHN on other parameterized connectivity graphs was not studied in the literature.

Table 7.1 (on page 128) gives an overview of the results that we discuss in this chapter.

**Example 7.2** Now we can relate instances of PMCPs from Section 7.1 with **COVER** with fixed inputs. Denote the process template from the running example by  $P_{re}$ . Then:

- PMC in (i) is equivalent to **COVER<sub>All</sub>** with  $C = \{err\}$  and  $P = P_{re}$ , where **All** is the set of all undirected connected graphs;
- PMC in (ii) is equivalent to **COVER<sub>C</sub>** with  $C = \{hub\}$  and  $P = P_{re}$ , where **C** is the set of all cliques; and
- PMC (iii) cannot be expressed directly via an instance of **COVER**, **TARGET** or **REPEAT**. But in case of AHNs we can express the PMC in (iii) via an instance of **COVER** if we modify the process template: add an outgoing broadcast transition from state  $l$ , and a corresponding receive transition from  $l$  into a special state  $s$ . Then, the PMC in (iii) becomes equivalent to **COVER<sub>All</sub>** with  $C = \{s\}$  and  $P = P_{re}$ .

<sup>1</sup>Recall that in this survey we defined the temporal logics on infinite runs only.

## 7.4 PARAMETERIZED CONNECTIVITY GRAPHS $\mathbf{BP}_k$ , $\mathbf{BPC}_k$ , $\mathbf{BD}_k$ , $\mathbf{C}$ , $\mathbf{All}$

[Delzanno et al. \[2010, 2011, 2012b\]](#) and [Abdulla et al. \[2013a\]](#) studied the PMC problems in ad hoc networks for different classes of parameterized connectivity graphs. The connectivity graphs studied model common topologies met in ad hoc networks. These topologies can be divided into two classes—flat and hierarchical (cf. [Karl \[2005, Chapter 10\]](#)).

In hierarchical topologies, nodes are grouped into clusters, and each cluster has a representative node. All nodes in a cluster communicate directly or via other nodes of the same cluster. Nodes from different clusters do not communicate directly but rather via representative nodes. Below we define bounded path graphs ( $\mathbf{BP}_k$ ) and bounded path clique graphs ( $\mathbf{BPC}_k$ ) that can model hierarchical topologies.

In contrast to hierarchical topologies, in flat topologies there are no representative nodes, and every node is equal with respect to communication. We will define bounded diameter graphs ( $\mathbf{BD}_k$ ) that model flat topologies with a bounded distance between nodes. Cliques ( $\mathbf{C}$ ) are a special case where the distances is always 1, and can be used to model dense ad hoc networks in which every node is in the transmission range of every other node.

The assumption of undirected connection between nodes does not always hold as the discrepancy between a node's transmission and reception range makes the following possible: a node can send messages to some other node, but cannot receive a message from this node. Connectivity graphs with directed edges can model nodes with such discrepancies.

We use the following basic definitions. A *path in a graph*  $G = (V, E)$  is a sequence  $v_0e_0v_1e_1 \dots v_n \in (VE)^*V$  such that  $(v_i, v_{i+1}) \in e_i$  for  $i < n$ . It is a *simple path* if no node appears twice. A graph is *connected* if there is a path between any two nodes of the graph. The *distance* between two nodes is the number of edges on the shortest simple path between them. The *length* of a simple path is the number of edges on it, thus the length of  $v_0e_0v_1e_1 \dots v_n \in (VE)^*V$  is  $n$ . A graph  $G = (V, E)$  is *undirected* if for any  $(v, w) \in E$  it holds that  $(w, v) \in E$ .

In the following definitions, let  $k \in \mathbb{N}_0$ .

**All, daAll, C.**  $\mathbf{All}$  is the parameterized connectivity graph consisting of all undirected connected graphs,  $\mathbf{daAll}$  the parameterized connectivity graph of all (directed) acyclic graphs, and  $\mathbf{C}$  is the parameterized connectivity graph of all clique graphs.

**BD<sub>k</sub>.** The *diameter* of a connected undirected graph is the maximal distance between any two nodes of the graph. An undirected connected graph is a *k-bounded diameter graph* if its diameter is less or equal to  $k$ . Denote by  $\mathbf{BD}_k$  the parameterized connectivity graph of all  $k$ -bounded diameter graphs. Intuitively, the graph diameter equals the maximal number of times a message needs to be broadcast in order to reach all nodes.

**BP<sub>k</sub>, daBP<sub>k</sub>.** A graph is a *k*-bounded path graph if all simple paths are of length smaller or equal to *k*. BP<sub>k</sub> consists of all undirected connected *k*-bounded path graphs. daBP<sub>k</sub> consists of all directed acyclic *k*-bounded path graphs. Note that for any  $k \in \mathbb{N}_0$ : BP<sub>k</sub>  $\subseteq$  BD<sub>k</sub>.

**BPC<sub>k</sub>.** An undirected connected graph is a *k*-bounded path maximal cliques graph if all simple paths between any of its maximal cliques have lengths smaller or equal to *k*. BPC<sub>k</sub> consists of all *k*-bounded path maximal cliques graphs. In terms of expressivity, BPC<sub>k</sub> is between BP<sub>k</sub> and BD<sub>2k+1</sub>: for any  $k \geq 1$ , it holds that BP<sub>k</sub>  $\subset$  BPC<sub>k</sub>  $\subset$  BD<sub>2k+1</sub>.

**Example 7.3** Some examples of graphs as defined above.

- A 1-bounded diameter graph is a clique, and BD<sub>1</sub> consists of all cliques, i.e., is equal to C.
- An undirected star is a 2-bounded path graph, the set of all undirected rings with a central node connected to everyone (“wheel”) is in BD<sub>2</sub> but not in BP<sub>k</sub> for any *k*.
- Undirected trees of depth *d* are contained in BP<sub>2d</sub>.
- A directed forest of depth *k* where all edges are directed away from the roots is contained in daBP<sub>k</sub>.
- Finally, BPC<sub>0</sub> coincides with the set of all cliques, and stars of cliques are in BPC<sub>2</sub>.

## 7.5 RESULTS FOR (NON-LOSSY) AHNS

The results of Delzanno et al. [2010, 2011, 2012b] for the PMCPs in the classes of graphs defined above are summarized in Figure 7.3.

In the figure the results for the clique column follow from Esparza et al. [1999].

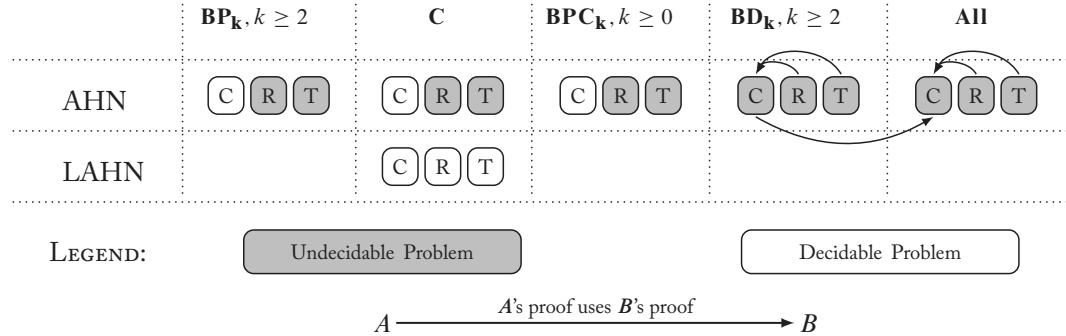
**Corollary 7.4** COVER<sub>C</sub> is decidable, REPEAT<sub>C</sub> and TARGET<sub>C</sub> are undecidable for AHNS.

*Proof idea.* The first item: it is straightforward to reduce COVER<sub>C</sub> for AHNS to PMCP( $\mathcal{P}$ , C<sub>0</sub>, Reg(A)) for broadcast systems which is decidable by Theorem 5.7. The proof reduces the PMCP problem to the (decidable) coverability problem for WSTSs. The second and third items follow from a straightforward modification of the construction described in Theorem 5.11 that reduces the (undecidable) non-halting problem for 2CMs to PMCP( $\mathcal{P}$ , C<sub>0</sub>, ωReg(A)). □

### 7.5.1 UNDECIDABILITY RESULTS FOR (NON-LOSSY) AHNS

This section explains all undecidability results for AHNS from Figure 7.3, namely:

- Theorem 7.6 proves that COVER<sub>AII</sub> is undecidable;
- Theorems 7.8 and 7.9 cover the case of graphs BD<sub>k</sub>;



**Figure 7.3:** Decidability results and reductions for ad hoc networks. The letters c, r, t in the cells stand for COVER, REPEAT, TARGET. The blank cells mean that to the best of our knowledge the problems were not studied in the literature. The results for directed graphs are not shown here—see Theorems 7.12 and 7.19.

- Theorem 7.10 proves that REPEAT and TARGET are undecidable on  $\mathbf{BP}_k$  for  $k \geq 2$ ;
- Corollary 7.11 proves that REPEAT and TARGET are undecidable on  $\mathbf{BPC}_k$  for  $k \geq 0$ ; and
- Theorem 7.12 proves that COVER is undecidable on directed graphs (not in the figure).

In the literature, undecidability results for ad hoc networks are usually established via reduction from the halting problem for 2CMs to COVER or, equivalently, from the non-halting problem for 2CMs to the PMC problem.

Most of the undecidability proofs build, for a given 2CM, a protocol that has the following two steps:

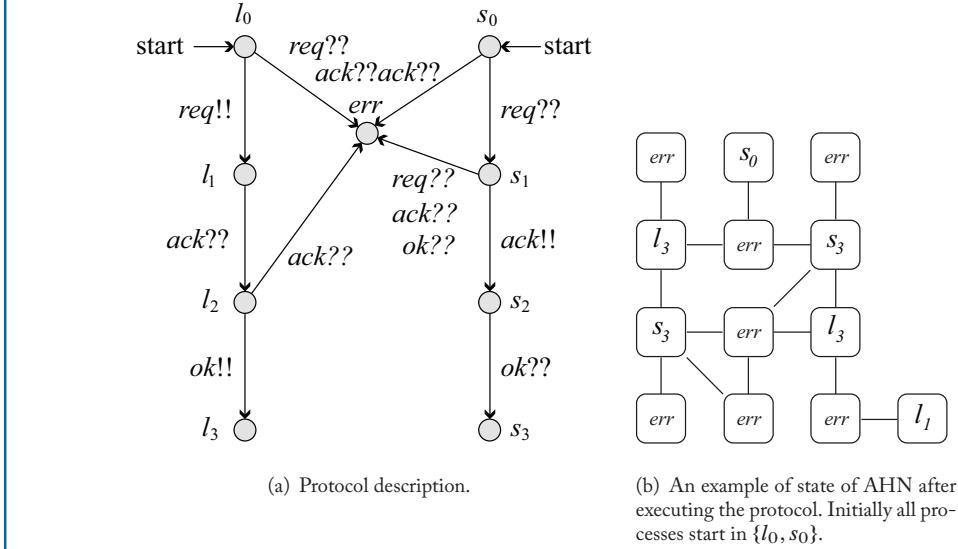
1. find a sub-graph of a special form, and
2. simulate the 2CM. The simulation assumes the connectivity graph is of the special form.

**Building block: RAO protocol.** All the protocols used in Delzanno et al. [2010, 2011] to prove undecidability results for All,  $\mathbf{BD}_k$ ,  $\mathbf{BP}_k$ ,  $\mathbf{BPC}_k$  as the first step use a variant of the Req-Ack-OK (RAO) protocol (Figure 7.4) that has the following property.

**Lemma 7.5** [Delzanno et al., 2010, Proposition 1] Let  $P$  be the RAO protocol (Figure 7.4(a)), and consider an AHN system instance  $P^G$ . If  $P^G$ , starting from the initial state, reaches a system state such that some process  $v \in V_G$  is in state  $s_3$ , then:

1. process  $v$  is adjacent to exactly one process in state  $l_3$  denoted by  $w$ ; and
2. any other process adjacent to  $v$  or  $w$  is in state err.

110 7. AD HOC NETWORKS



**Figure 7.4:** RAO protocol, the building block for protocols to find sub-graphs of a special form [Delzanno et al., 2010].

*Proof idea.* Note that process  $v$  can end in state  $s_3$  only if it receives a broadcast message  $ok$ , hence there must be an adjacent process that is in state  $l_3$ . By applying the RAO protocol backward, one can show that processes  $w, v$  can reach states  $l_3, s_3$  only if they, starting from initial states  $l_0, s_0$ , receive messages only from each other. Then, by applying the RAO protocol forwards to processes  $w, v$  starting in the initial states, one can show that if  $w, v$  reach  $l_3, s_3$  then all their adjacent processes will end in state  $err$ .  $\square$

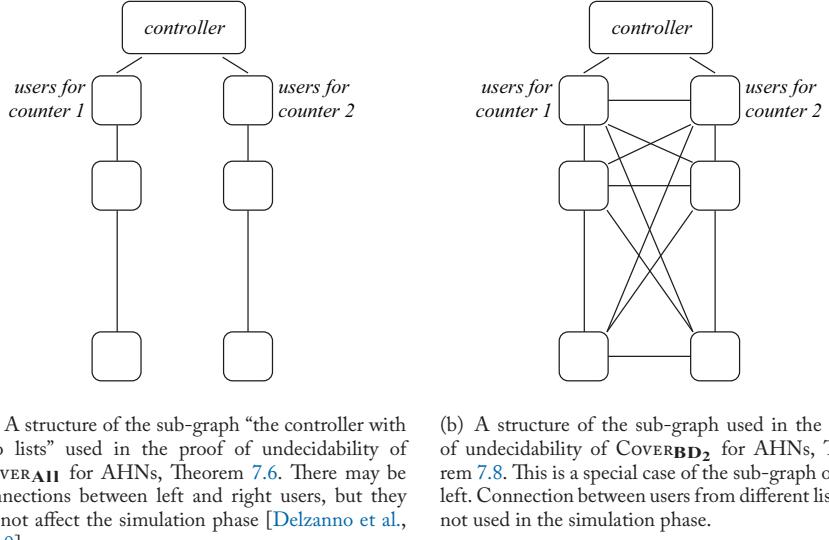
Figure 7.4(b) shows an example of a system state after completion of the RAO protocol. Intuitively, the protocol links two adjacent processes (that is, a process in state  $s_3$  is linked to a neighbor in state  $l_3$ ) starting in states  $l_0, s_0$  and sends all other adjacent processes into state  $err$ .

**Undecidability of CoverAll.** We are now ready to discuss the first undecidability result in the case of AHNs.

**Theorem 7.6** [Delzanno et al., 2010] CoverAll for AHNs is undecidable.

*Proof idea.* To simulate a 2CM consider process template  $P$  that implements a two-phase protocol.

1. Find a sub-graph of the form in Figure 7.5(a), we call it “the controller with two lists.”
2. Simulate a 2CM.



**Figure 7.5:** Sub-graphs used to simulate a 2CM.

The second phase starts only if the first phase succeeds. Using the protocol we will show that the 2CM halts if and only if there is a graph  $G \in \text{All}$  such that system  $P^G$  reaches a state with some process in state *halt*.

*1st phase: Finding a sub-graph “the controller with two lists.”*

The first phase can succeed only if the connectivity graph has a sub-graph of a special form: a controller process connected to two lists of user processes. The protocol in Figure 7.6 tries to find such a sub-graph. Initial states are  $\{R, C'_i, C''_i, C_i | i = 1, 2\}$ . Note that it is possible that after the protocol execution the connectivity graph is split into more than a single sub-graph each of the desired form, but these sub-graphs are isolated (nodes that connect them are in the error state and will not participate in the second phase). The transitions leading to the error state are not shown in the figure.

Roughly, the phase guesses a sub-graph of the desired form and then verifies it exists.

The phase can succeed only in executions in which the system starts in an initial state with at least one process in state  $R$  (call it the controller) that is connected to two lists of the form  $C'_i - C''_i - C_i - C'_i - C''_i - C_i - \dots$  (call them list 1 and list 2 for  $i = 1, 2$ , and call the first nodes of two lists head 1 and 2). Note that non-determinism of the process initial state implies: even if there is a sub-graph “the controller with two lists,” it is still possible that the first phase fails because the initial system state is not of the desired form. But it is certain that if there is a desired sub-graph, then there is a desired initial system state.

## 112 7. AD HOC NETWORKS

The phase uses a modified RAO protocol that has the following differences. The original RAO protocol ensures that two adjacent processes that are initially in states  $l_0$  and  $s_0$  can reach states  $l_3$  and  $s_3$  while sending all other adjacent processes into state  $err$ . In contrast, the modified RAO protocol in Figure 7.6 describes several-steps protocol that links an  $R$ -process with a unique  $C'_1$ -process (step 1), that  $C'_1$ -process with a  $C''_1$ -process (step 2), that  $C''_1$ -process with a  $C_1$ -process (step 3), and that  $C_1$ -process with a  $C'_1$ -process (similar to step 1 but the  $C_1$ -process plays the role of an  $R$ -process).

Step 1 links an  $R$ -process with a unique adjacent  $C'_1$ -process. All other processes adjacent to the  $R$ -process will go either into  $err$  state or they belong to another counter and do not change their states. All other processes adjacent to the  $C'_1$  process will go either into  $err$  except those in  $C''$  state.

Step 2 links a  $C'_1$ -process with a unique adjacent  $C''_1$ -process, while sending all others adjacent to  $C'_1$  into  $err$  except the previously linked  $C_1$ -process, and sending all others adjacent to  $C''_1$  into  $err$  except those in state  $C_1$ .

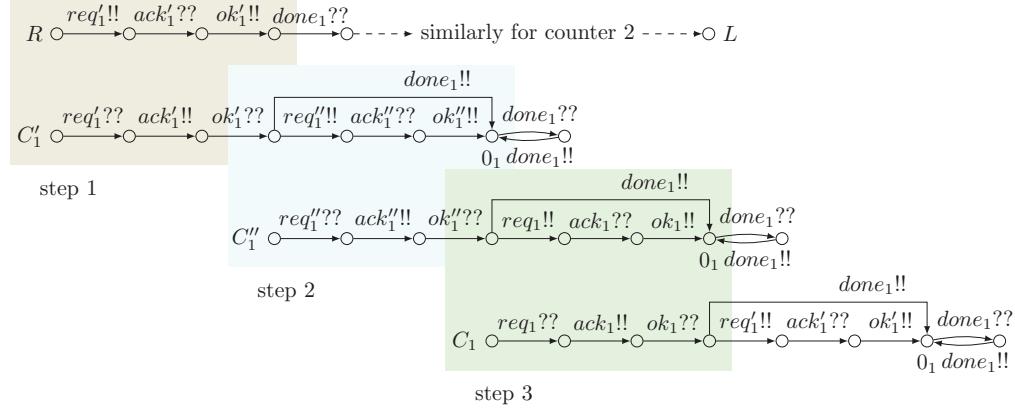
Step 3 links a  $C''_1$ -process with a unique adjacent  $C_1$ -process, while sending all others adjacent to  $C''_1$  into  $err$  except the previously linked  $C'_1$ -process, and sending all others adjacent to  $C_1$  into  $err$  except those in state  $C'_1$ .

In all steps a process that initiates linking with another process (in Step 1 –  $C_1$ -process, in Step 2 –  $C'_1$ -process, in Step 3 –  $C''_1$ -process) non-deterministically chooses between linking with another process or sending  $done_1$ . Sending  $done_1$  completes the initialization of list 1 – the message is transmitted back to the controller (an  $R$ -process), and the controller starts initializing list 2 in a similar way. If the initialization of both lists completes (thus the controller is not hanged waiting for a message from head 1 or 2), then the system state is of the form  $0_1 - \dots - 0_1 - L - 0_2 - \dots - 0_2$ , and the controller starts simulating the 2CM.

*2nd phase: Simulating a 2CM.* If the first phase succeeds, then there is at least one sub-graph of the form in Figure 7.5(a). Consider one such sub-graph. In the simulation phase the controller process simulates transitions of the control state of the 2CM, while user nodes encode values of counters (each user “stores” a bit, and the sum of bits in list  $i$  is equal to the value of counter  $i$ ). To increase the value of counter 1 the controller broadcasts message  $inc_1$  which is received by two head processes. Only head 1 reacts to the message: if it has its bit set, then it passes the message further along list 1, otherwise head 1 sets its bit and sends  $ack$  to acknowledge the message. When the controller receives  $ack$ , it continues simulation. “Decrease” and “test for zero” work similarly. This completes the description of the simulation phase.

The protocol ensures that the 2CM halts if and only if there is a system where the controller process can reach a halting state, which is an instance of **COVER<sub>II</sub>** problem.  $\square$

**Example 7.7** Now we can relate the results in Figure 7.3, in particular Corollary 7.4 and Theorem 7.6, with the running example from Section 7.1. Since **COVER<sub>II</sub>** for AHNs is undecidable, there is no single algorithm that can be used to solve all PMCP instances of the form (i)



**Figure 7.6:** A part of the protocol to find a sub-graph “controller with two lists” used in Theorem 7.6. Only the part which looks for a list of processes that store the value of counter 1 is shown. The protocol can be naturally extended for both counters: after receiving  $done_1$  the controller runs a similar protocol but with messages  $req_2, ack_2 \dots$  instead of  $req_1, ack_1 \dots$ . Transitions to state  $err$  are omitted for readability [Delzanno et al., 2010].

from Section 7.1 (recall from Example 7.2 that the PMC in (i) is equivalent to an instance of  $\text{COVER}_{\text{All}}$ ). In contrast, the corresponding PMCP for AHNs arranged in cliques is decidable by Corollary 7.4—thus, an algorithm that solves  $\text{COVER}_{\text{C}}$  can also be used to answer the PMC question in (ii).

**Undecidability for  $\mathbf{BD}_k$ .** Consider the case of a more restricted parameterized connectivity graph  $\mathbf{BD}_k$ . Since  $\mathbf{BD}_1$  coincides with the set of all cliques, it follows from Corollary 7.4 that  $\text{COVER}_{\mathbf{BD}_1}$  is decidable, and that  $\text{REPEAT}_{\mathbf{BD}_1}$  and  $\text{TARGET}_{\mathbf{BD}_1}$  are undecidable. For  $k \geq 2$   $\text{COVER}_{\mathbf{BD}_k}$  becomes also undecidable.

**Theorem 7.8** [Delzanno et al., 2011, Theorem 1]  $\text{COVER}_{\mathbf{BD}_k}$ , for AHNs, is undecidable for  $k \geq 2$ . Notice that this theorem does not directly follow from the previous one. We prove it differently from Delzanno et al. [2011], where the authors developed a separate protocol for  $\mathbf{BD}_k$ .

**Proof idea.** When one tries to prove undecidability of  $\text{COVER}_{\mathbf{BD}_2}$  (other  $k$ s follow), the first natural question is “Can we reuse the protocol from **All** case?” It might happen that the sub-graph extracted from **All** (of the form in Figure 7.5(a)) is not present in  $\mathbf{BD}_k$ , which would mean that the protocol that was able to guess and then extract two lists of unbounded length from **All** will not be able to extract such lists from  $\mathbf{BD}_2$ . I.e., when working on graphs in  $\mathbf{BD}_2$ , the protocol from **All** case can guess the presence of a long list but it cannot complete its execution, because  $\mathbf{BD}_2$  does not contain the list of such length.

## 114 7. AD HOC NETWORKS

This is not the case though, since the protocol for **All** does not extract two completely “isolated” lists (connected to the controller), but rather two lists that can have interconnections between nodes from different lists. In other words, **BD<sub>2</sub>** has the sub-graph shown in Figure 7.5(b) that i) can be extracted by the protocol from **All** case, ii) is of the form needed by the second phase when simulating a given 2CM. Hence, the protocol from **All** can be reused for **BD<sub>2</sub>** and **Cover<sub>BD<sub>k</sub></sub>** for  $k \geq 2$  is undecidable.  $\square$

It is not difficult to show that undecidability of **Cover<sub>BD<sub>k</sub></sub>** implies undecidability of **Repeat<sub>BD<sub>k</sub></sub>** and **Target<sub>BD<sub>k</sub></sub>**, hence:

**Corollary 7.9** [Delzanno et al., 2011, Esparza et al., 1999] **Repeat<sub>BD<sub>k</sub></sub>**, **Target<sub>BD<sub>k</sub></sub>**, for AHNs, are undecidable for  $k \geq 1$ .

**Undecidability of Target<sub>BP<sub>k</sub></sub>**, Repeat<sub>BP<sub>k</sub>, Target<sub>BPC<sub>k</sub>, Repeat<sub>BPC<sub>k</sub>. Now consider even a more restricted parameterized connectivity graph **BP<sub>k</sub>**. Restricting the graph makes **Cover** decidable (see Theorem 7.19), but **Repeat** and **Target** are still undecidable:</sub></sub></sub>

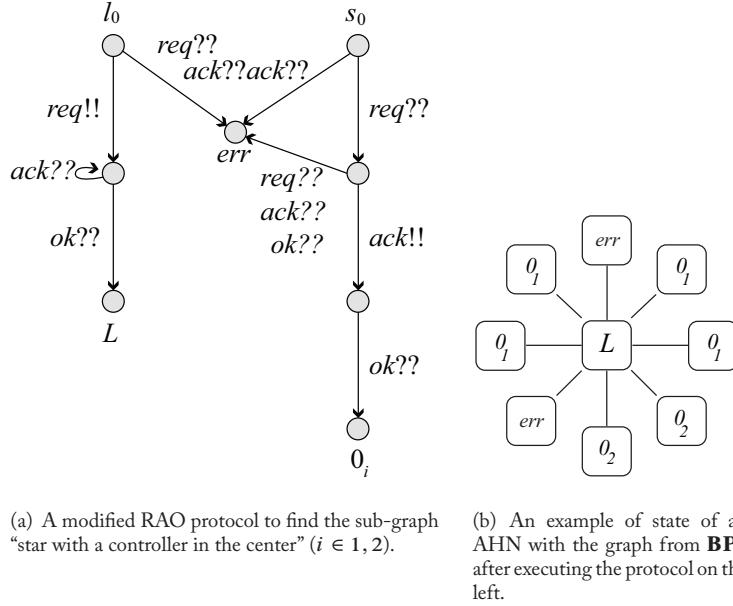
**Theorem 7.10** [Delzanno et al., 2010, Theorem 6] **Target<sub>BP<sub>k</sub></sub>** and **Repeat<sub>BP<sub>k</sub></sub>**, for AHNs, are undecidable for  $k \geq 2$ .

**Proof idea.** Here we give an idea how to reduce the halting problem of 2CMs to **Target<sub>BP<sub>2</sub></sub>**. The idea can be adapted to **BP<sub>k</sub>** with  $k > 2$ . Consider a process template that implements a two-phase protocol:

1. Find a sub-graph of a form “star with a controller in the center.”
2. Simulate the 2CM.

The first phase of the protocol is simple in comparison to that of the previous theorems and is shown in Figure 7.7(a). If the first phase succeeds, then nodes that are not in the error state form a graph of the form “star with a controller in the center” with a node in the center called the controller.

In the second phase the controller simulates 2CM, while user nodes store counters values (each user “stores” a bit, the total number of bits being equal to the value of a counter). In the second phase  $ack_i, ok$  are different from those used in the first phase. To increase the value of counter  $i$  the controller broadcasts  $inc_i!!$ . If there is a user process in state  $0_i$ , then it responds with  $ack_i!!$  and goes to state  $1_i$ . The controller then in its turn acknowledges it with  $ok_i!!$  which prevents other processes from sending  $ack_i!!$ . If the controller though receives several  $ack_i!!$  before it manages to send the acknowledgement then it moves into state  $err$  and never leaves it. Operation “decrease” works in a similar way. Instead of simulating operation “test for zero” for counter  $i$ , the controller non-deterministically guesses the result, and if the guess is zero, then the controller broadcasts  $zero_i!!$  and continues simulation. If the guess is wrong and there is a user in state  $1_i$ , then that user moves into state  $err$ .



**Figure 7.7:** Finding a star-like sub-graph in graph from **BP2** [Delzanno et al., 2010].

On reaching a halting state the controller sends all users except those in state *err* to a special state *halt* and goes there itself. Therefore, 2CM halts if and only if there is a system where all processes reach *halt*, which is an instance of  $\text{TARGET}_{\mathbf{BP}_2}$ .  $\square$

Recall that if each clique in a graph from  $\mathbf{BPC}_k$  is replaced with a single node, then the resulting graph belongs to  $\mathbf{BP}_k$ . Hence in order to show undecidability of  $\text{TARGET}_{\mathbf{BPC}_k}$  and  $\text{REPEAT}_{\mathbf{BPC}_k}$  we modify the protocol used in the previous proof and add an initial phase in which in each clique a single representative node is chosen, and the rest of the protocol is unchanged. Hence  $(\mathbf{BPC}_0)$  coincides with the parameterized clique and was considered in Corollary 7.4):

**Corollary 7.11**  $\text{TARGET}_{\mathbf{BPC}_k}$  and  $\text{REPEAT}_{\mathbf{BPC}_k}$ , for AHNs, are undecidable for  $k \geq 0$ .

**Undecidability of Cover<sub>daAll</sub>.** Finally, we state the result of Abdulla et al. [2013a] for the case of directed acyclic graphs:

**Theorem 7.12** [Abdulla et al., 2013a, Theorem 1]  $\text{COVER}_{\text{daAll}}$  for AHNs is undecidable.

**Proof idea.** The previous proofs used the fact that the parameterized graphs are undirected: the basic RAO protocol in Figure 7.13(c) assumes the ability of a process to send and receive broadcast messages to and from all its adjacent nodes. If this is not the case as can be in directed graphs,

## 116 7. AD HOC NETWORKS

the RAO protocol's main property ("isolation of pairs of processes," Lemma 7.5) does not hold. Furthermore, the simulation construction described in the proof of Theorem 7.6 breaks in the general case of directed graphs, because the controller is not able to get the feedback about its commands from the user processes. These facts make adaption of the standard proof technique (the reduction from the halting problem for 2CMs to COVER) not easy.

Hence, for the case of parameterized graphs **daAll**, Abdulla et al. [2013a] considered a different undecidable problem called TRANSD. Informally, TRANSD takes as input two automata  $A$  and  $B$ , and one transducer  $T$ , all with the same alphabet (transducer is an automaton that outputs a symbol in each transition). TRANSD outputs "Yes" iff there is  $i \in \mathbb{N}_0$  and a word  $w$  accepted by  $A$  such that  $i$  successive applications of the transducer  $T$  to the word  $w$  results in a word accepted by  $B$ .

The authors briefly show in the paper that TRANSD is undecidable. Indeed, a transducer can compute one step of a Turing machine, and thus iterating this transducer simulates arbitrarily many steps of that Turing machine.

To reduce TRANSD to COVER<sub>**daAll**</sub>, the authors build a process template that simulates a given automata and a transducer. Initially, every process decide whom it simulates –  $A$ ,  $B$  or  $T$ . For the simulation to be correct, it is necessary that the processes will form a graph of the form  $A \rightarrow T \rightarrow \dots \rightarrow T \rightarrow B$ , i.e., the first process simulates automaton  $A$  followed by a (possibly zero) number of processes simulating transducer  $T$ , and then a process that simulates automaton  $B$ . Initially every  $A$ -process sends a special initiating message intended for a  $T$ - or  $B$ -process—on receiving it every  $T$ -process sends the same message. Every  $A$ -,  $T$ -,  $B$ - process on receiving initiating message twice goes into error state, it also goes into the error if received a message not intended for them or received two messages with no sending in between. After sending the initiating message the  $A$ -process starts transmitting a word accepted by  $A$ , letter by letter, followed by the end marker. The  $T$ -process receives a letter of the word and sends the transduced one further, and so on, to the  $B$ -process. The  $B$ -process receives a letter and simulates transitions of the automaton  $B$ . The  $B$ -process goes into state *halt* if it receives the end marker and it is in an accepting state of the automaton.

The control state of COVER consists of a single state *halt*. The construction above ensures: there is an AHN  $P^G$  with  $G \in \textbf{daAll}$  where a  $B$ -process can go into the control state, if and only if some number of applications of  $T$  to a word of automaton  $A$  results in a word of automaton  $B$ .  $\square$

### 7.5.2 DECIDABILITY RESULTS FOR (NON-LOSSY) AHNS

In the previous section, in the proofs of undecidability of COVER<sub>**All**</sub> (Theorem 7.6) and COVER<sub>**BD<sub>k</sub>**</sub> (Theorem 7.8) we used "lists" of user nodes to store the values of counters of a 2CM. In the general case there is no bound on counter values, so the proofs only work if we can find lists of unbounded length in the parameterized connectivity graph. Thus, the undecidability proofs break for parameterized connectivity graphs in which all graphs have only "lists" of a bounded length,

or, in other words, if we restrict the length of simple paths on which a message in the system can travel. In the literature several such parameterized connectivity graphs are considered, namely,  $\mathbf{BP}_k$ ,  $\mathbf{BPC}_k$ , and  $\mathbf{dabP}_k$ .

This section proves Theorem 7.19 that states decidability of COVER on  $\mathbf{BP}_k$ ,  $\mathbf{BPC}_k$  and on  $\mathbf{dabP}_k$ , thus covering all decidability results from Figure 7.3 for (non-lossy) AHNs.

The proofs of decidability of COVER of AHNs on these parameterized connectivity graphs use the machinery of well structured transition systems (WSTS) (see Section 3.2.1). The proofs have a similar structure for all the graphs except  $\mathbf{dabP}_k$ : Abdulla et al. [2013a] first reduced COVER on  $\mathbf{dabP}_k$  to COVER on inverted trees—that is, trees that contain a node  $v$  such that all other nodes have a directed path to  $v$ —of bounded depth.

This section is organized as follows.

1. We define the induced sub-graph relation  $\leq_{is}$  on directed graphs.
  2. Lemma 7.13 proves that COVER on directed acyclic bounded path graphs ( $\mathbf{dabP}_k$ ) can be reduced to COVER on bounded inverted trees ( $\mathbf{BIT}_k$ , defined later).
  3. Lemma 7.14 and 7.15 prove that  $\leq_{is}$  is a well quasi-order on  $\mathbf{BPC}_k$ ,  $\mathbf{BP}_k$ ,  $\mathbf{BIT}_k$ .
  4. We define transition system  $M$ , which for given  $P$ ,  $G$ , represents the behavior of all instances of the parameterized AHN  $P^G$ .
  5. Lemma 7.18 proves that the transition system  $M$  for AHNs on  $\mathbf{BP}_k$ ,  $\mathbf{BPC}_k$ ,  $\mathbf{BIT}_k$  is a WSTS with respect to  $\leq_{is}$  (the proof uses Lemmas 7.14 and 7.15, and also intermediate Lemmas 7.16 and 7.17).
  6. Finally, using Lemmas 7.13 and 7.18, Theorem 7.19 proves decidability of COVER for AHNs on  $\mathbf{BP}_k$ ,  $\mathbf{BPC}_k$ , and  $\mathbf{dabP}_k$ .
  7. We describe the parameterized model checking algorithm that solves COVER for AHNs on  $\mathbf{BP}_k$ ,  $\mathbf{BPC}_k$ , and  $\mathbf{dabP}_k$ , and illustrate how it works.
- 1. The induced sub-graph relation  $\leq_{is}$ .** For graphs  $G_1$ ,  $G_2$  and labeling functions  $s_1 : V_1 \rightarrow Q$  and  $s_2 : V_2 \rightarrow Q$  with a set of labels  $Q$  define:  $(G_1, s_1) \leq_{is} (G_2, s_2)$  iff there is a label preserving injection  $h : V_1 \rightarrow V_2$  such that  $(v, w) \in E_1 \iff (h(v), h(w)) \in E_2$ . This is different from the sub-graph relation which preserves labels but allows new edges, i.e.,  $(h(v), h(w)) \in E_2$  does not necessarily imply  $(v, w) \in E_1$ . Figure 7.8 gives an example.
- 2. Reducing Cover $_{\mathbf{dabP}_k}$  to Cover $_{\mathbf{BIT}_k}$ .** Notice that the induced sub-graph relation  $\leq_{is}$  is not a well quasi-order on  $\mathbf{dabP}_k$  (an example of infinite decreasing chain is in Abdulla et al. [2013a, Section 8]). This is why we first need to reduce Cover $_{\mathbf{dabP}_k}$  to COVER on inverted trees of bounded depth for which  $\leq_{is}$  is a well quasi-order (Lemma 7.15). Let  $\mathbf{BIT}_k$  be a parameterized connectivity graph consisting of all inverted trees of depth  $k$ . Informally, an inverted tree is a



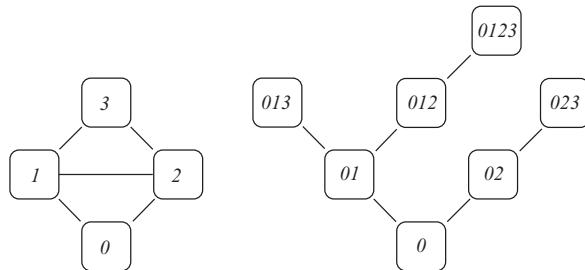
**Figure 7.8:** Example for the induced sub-graph relation  $\leq_{is}$ .

tree with all edges pointing toward the root. For the formal definition see, for example, [Abdulla et al., 2013a].

Then the following reduction from  $\text{COVER}_{\text{daBP}_k}$  to  $\text{COVER}_{\text{BIT}_k}$  holds.

**Lemma 7.13** [Abdulla et al., 2013a, Theorems 2 and 5] Given a decision procedure for  $\text{COVER}_{\text{BIT}_k}$ , one can construct a decision procedure for  $\text{COVER}_{\text{daBP}_k}$ .

**Proof idea.** For a given graph in  $\text{daBP}_k$  we build an inverted forest of depth  $k$  that preserves all the paths of the the original graph but has new nodes (example in Figure 7.9).



**Figure 7.9:** [Abdulla et al., 2013a] Directed acyclic graph and a corresponding inverted forest used in Lemma 7.13.

Then: if a process state is reachable in the AHN on a graph from  $\text{daBP}_k$ , then this state is also reachable in the AHN on the inverted forest of depth  $k$ . The idea is to simulate the behavior of the original nodes (e.g., in Figure 7.9 all nodes with names starting with 0 will repeat the behavior of node 0). The other direction—if a process state is reachable in an AHN on an inverted forest of depth  $k$ , then it is reachable in the AHN on a graph from  $\text{daBP}_k$ —is trivial since any inverted forest of depth  $k$  is in  $\text{daBP}_k$ .

Recall that a forest consists of trees and processes from different trees do not communicate with each other. Hence, a process can reach a control state in the AHN on an inverted forest if and only if a process from one of the trees can reach this state. This means that instead of considering  $\text{COVER}$  on all inverted forests of depth  $k$  we can consider  $\text{COVER}$  on all trees of those forests,

i.e., COVER on all inverted trees of depth  $k$ . Finally, this means that COVER<sub>**daBP<sub>k</sub>**</sub> is equivalent to COVER<sub>**BIT<sub>k</sub>**</sub>.  $\square$

### 3. $\leq_{is}$ is a well-quasi-order on **BP<sub>k</sub>**, **BPC<sub>k</sub>**, **BIT<sub>k</sub>**.

**Lemma 7.14** [Delzanno et al., 2011, Lemma 2] For fixed  $k \in \mathbb{N}$ , the induced sub-graph relation  $\leq_{is}$  is a well quasi-order on labeled graphs from **BPC<sub>k</sub>**.

**Proof idea.** The proof extends that of Ding [1992, Theorem 2.1] for the case of **BP<sub>k</sub>**. The crucial property of **BPC<sub>k</sub>** (and of **BP<sub>k</sub>**) is that in any graph from **BPC<sub>k</sub>** there is a vertex such that its removal splits the graph into non-connected graphs from **BPC<sub>k-1</sub>**. This property, together with Higman's lemma [Higman, 1952] (informally: if  $\leq$  is a well quasi-order on  $Q$ , then the component-wise variation  $\leq^*$  is a well quasi-order on  $Q^*$ ), leads to an inductive proof.  $\square$

**BP<sub>k</sub>**  $\subset$  **BPC<sub>k</sub>** for any  $k \geq 1$ , hence by the lemma  $\leq_{is}$  is a well quasi-order on **BP<sub>k</sub>** too (the case of  $k = 0$  is trivial).

Abdulla et al. [2013a] introduced a special class of multi-sets that can encode inverted trees of depth  $k$ , and then prove that these sets are well quasi ordered and so are inverted trees of depth  $k$ . Here, instead, we show that inverted trees of depth  $k$  are well quasi-ordered with the induced sub-graph relation  $\leq_{is}$ .

**Lemma 7.15** For a fixed  $k \in \mathbb{N}$ , the induced sub-graph relation  $\leq_{is}$  is a well quasi-order on labeled graphs from **BIT<sub>k</sub>**.

**Proof idea.** We use the fact that  $\leq_{is}$  is a well quasi-order on labeled **BP<sub>k</sub>** [Ding, 1992, Theorem 2.1].

First, any inverted tree of depth  $k$  can be uniquely up to isomorphism mapped to a  $3k$ -bounded path graph. For example, the inverted tree  $a \rightarrow b$  maps to  $a - v - w - b$ , where  $v, w$  are two auxiliary nodes to help encode edge directions<sup>2</sup>.

Take any infinite sequence  $G_1, G_2, \dots$  of inverted trees of depth  $k$ . We will show that there is  $i < j$  with  $G_i \leq_{is} G_j$ .

Consider the sequence  $G'_1, G'_2, \dots$  of encoding graphs. They are from  $3k$ -bounded path graphs which are well quasi ordered with  $\leq_{is}$  [Ding, 1992, Theorem 2.1], and therefore the sequence has  $i < j$  with  $G'_i \leq_{is} G'_j$ .

Thus, for original graphs,  $G_i \leq_{is} G_j$  also holds: by induction we can show that the induced sub-graph relation preserves exact paths, hence for any path  $a - v - w - b$  in  $G'_i$  there is a path  $h(a) - h(v) - h(w) - h(b)$  in  $G'_j$ , hence for edge  $a \rightarrow b$  in  $G_i$  there is edge  $h(a) \rightarrow h(b)$  in  $G_j$ . The other direction (for any edge  $h(a) \rightarrow h(b)$  there is edge  $a \rightarrow b$ ) can be proven similarly.<sup>3</sup>  $\square$

<sup>2</sup>This encoding was suggested to us by authors of [Abdulla et al., 2013a], in particular by Othmane Rezine.

<sup>3</sup>Here,  $h$  is a label preserving injection  $V(G'_i) \rightarrow V(G'_j)$  as required by the definition of  $\leq_{is}$ , we also assume that for any  $i$ :  $V(G'_i) = V(G_i) \cup \text{auxiliary nodes}$ , and we reused  $h$  to show that  $G_i \leq_{is} G_j$ .

## 120 7. AD HOC NETWORKS

4 & 5. Parameterized AHNs on **BP<sub>k</sub>**, **BPC<sub>k</sub>**, **BIT<sub>k</sub>** are WSTSs wrt.  $\leq_{is}$ . For given parameterized AHN  $\overline{P}^{\mathbf{G}}$  define the *transition system M* (in Lemma 7.18 we prove it is a WSTS).

- The set of states consists of all pairs  $(G, s)$  with  $G \in \mathbf{G}, s \in S$  where  $S$  is the set of states of  $P^G$ , and the set  $I$  of initial states  $I = \{(G, s_0) \mid G \in \mathbf{G}, s_0 \in S_0\}$ . Notice that  $I$  is downward closed with respect to  $\leq_{is}$ .
- The transition relation contains  $(G, s) \rightarrow (G, s')$  iff there is a transition  $s \rightarrow s'$  in  $P^G$ .

Intuitively, the transition system  $M$  is an infinite state system that represents all instances of a given parameterized AHN  $P^G$ .

Now we can state the following.

**Lemma 7.16** [Delzanno et al., 2010, Lemma 2]<sup>4</sup> Fix  $\overline{P}^{\mathbf{G}}$ , and let  $M$  be defined as above. Then the induced sub-graph relation  $\leq_{is}$  is monotonic with respect to the transition system  $M$ .

**Proof idea.** Let  $(G, s) \leq (G', s')$  for some states  $s, s'$  in  $S$  and  $G, G'$  in  $\mathbf{G}$  (where  $S$  is the set of states of  $\overline{P}^{\mathbf{G}}$ ). Consider the case when  $(G, s) \rightarrow (G, t)$  happens via a broadcast by process  $v$  in system  $\overline{P}^G$ . Then there is a process  $v'$  of  $\overline{P}^{G'}$  which is in the same state as process  $v$ , and  $v'$  is connected to processes in the same states as neighbors of  $v$ . Then if the process  $v'$  in  $\overline{P}^{G'}$  makes the same broadcast action as  $v$  in  $\overline{P}^G$  then all the neighbors of  $v'$  in  $G'$  behave the same way as neighbors of  $v$  in  $G$ , and possibly neighbors of  $v'$  in  $G' \setminus G$  also change state.  $\square$

Recall from Section 3.2: for a given set  $C$  of states of transition system  $M$ ,  $Pred(C)$  is defined as the union of  $C$  and all one-step predecessors of  $C$ ; a basis of an upward closed set  $D$  is set  $B$  s.t.  $D = \uparrow B$ . Now we are ready to state:

**Lemma 7.17** [Abdulla et al., 2013a, Delzanno et al., 2010, 2011]

Fix  $k \in \mathbb{N}_0$ ,  $\overline{P}^{\mathbf{G}}$  where  $\mathbf{G}$  is one of **BP<sub>k</sub>**, **BPC<sub>k</sub>**, **BIT<sub>k</sub>**, and let  $M$  be the transition system defined as before. Then given a basis of an upward closed set  $C$ , one can compute a basis of  $Pred(C)$  for the transition system  $M$ .

**Proof idea.** A variation of the algorithm is in Algorithm 1. The authors do not provide the proof of the algorithm correctness, but the insight is: to compute  $Pred(\uparrow B)$  for some finite  $B = ((G_1, s), \dots, (G_i, s_i), \dots, (G_k, s_k))$ , it is enough to compute all predecessors of all  $(G_i, s_i)$  in  $B$ , and compute all predecessors of all  $(G_i^+, s_i^+)$  resulting from  $(G_i, s_i)$  by adding a single node making a broadcast transition. The termination follows from the finiteness of iterated objects.  $\square$

<sup>4</sup> The original lemma is for graphs from **BP<sub>k</sub>**, but the proof applies to general graphs as well.

---

**Algorithm 1:** BPred: Computing a finite basis of a predecessor (Lemma 7.17).

---

**Input:**  $P, G \in \{\text{BP}_k, \text{BPC}_k, \text{BIT}_k\}$ , a finite set  $T$  of states of the transition system  $M$

**Output:** a basis of  $\text{Pred}(\uparrow T)$

```

def BPred( $P, G, T$ ):
     $B = T$ 
    for  $(G, s) \in T$  :
        for  $q' \xrightarrow{\text{act}} q$  of  $P$  :
             $A = \text{pred1}(G, s, q' \xrightarrow{\text{act}} q)$ 
            add to  $B$  all  $(G, s) \in A$  that are not in  $\uparrow B$ 
            if  $q' \xrightarrow{\text{act}} q$  is a broadcast send transition :
                 $T^+ = \{(G^+, s^+) \mid it\ is\ (G, s)\ with\ one\ new\ node\ in\ state\ q\ and\ G^+\in G\}^5$  for
                 $(G^+, s^+) \in T^+$  :
                     $A = \text{pred1}(G^+, s^+, q' \xrightarrow{\text{act}} q)$ 
                    add to  $B$  all  $(G, s) \in A$  that are not in  $\uparrow B$ 
    return  $B$ 

def pred1( $G, s, q' \xrightarrow{\text{act}} q$ ) :
     $A = \emptyset$ 
    for  $v \in G$  with  $s(v) = q$  :
        if  $q' \xrightarrow{\text{act}} q$  is internal :
            add  $(G, s')$  to  $A$  where  $s'$  is the predecessor of  $s$  corresponding to  $q' \xrightarrow{\text{act}} q$ 
        elif  $q' \xrightarrow{\text{act}} q$  is broadcast send  $q' \xrightarrow{a!!} q$  :
            if process  $v$  has a neighbour6 in state  $r$  and  $r' \xrightarrow{a??} r$  for some  $r$  :
                continue
             $\{v_1, \dots, v_k\} =$  the neighbors of  $v$  that received  $a$  (for every  $v_i \exists r'_i \xrightarrow{a??} s(v_i)$ )7
            for subset  $\in 2^{\{v_1, \dots, v_k\}}$  :
                 $S' = \{s' \mid s'\ is\ a\ predecessor\ of\ s\ where\ v\ synchronizes\ only\ with\ subset$ 8
                add  $(G, s')$  to  $A$  for every  $s' \in S'$ 
    return  $A$ 

```

---

<sup>5</sup>To optimize, consider only  $G^+$ 's where the new node is connected to at least one node that will be able to receive the broadcast.

<sup>6</sup>" $v$  has a neighbour  $v'$ " means there is an edge from  $v$  to  $v'$ .

<sup>7</sup>For each  $v_i$  there might be more than one  $r'_i$  with  $r'_i \xrightarrow{a??} s(v_i)$ .

<sup>8</sup>I.e.,  $\forall v_i \in \text{subset} : s'(v_i) \xrightarrow{a??} s(v_i)$  is a transition of  $P$ ,  $\forall v' \in G \setminus \{v, v_i\} : s'(v') = s(v')$ ,  $s'(v) = q'$ .

## 122 7. AD HOC NETWORKS

**Lemma 7.18** Fix  $k \in \mathbb{N}_0$ ,  $\overline{P}^{\mathbf{G}}$  where  $\mathbf{G}$  is one of  $\mathbf{BP}_k$ ,  $\mathbf{BPC}_k$ ,  $\mathbf{BIT}_k$ , and let  $M$  be the transition system defined as before. Then  $M$  is a well structured transition system with respect to  $\leq_{is}$ .

*Proof.*  $M$  and  $\leq_{is}$  satisfy all items of the definition of a WSTS wrt.  $\leq_{is}$  (Section 3.2.1):

- $\leq_{is}$  is a well quasi-order on states of  $M$  (Lemmas 7.14 and 7.15),
- $\leq_{is}$  is monotonic with respect to  $M$  (Lemma 7.16), and
- a basis of  $Pred(\uparrow B)$  for a given finite set  $B$  of states of  $M$  is computable (Lemma 7.17)

□

6. Cover is decidable. Finally, we are ready to prove the main result of this section.

**Theorem 7.19** [Abdulla et al., 2013a, Delzanno et al., 2010, 2011]  $\text{COVER}$  is decidable for AHNs on i)  $\mathbf{BPC}_k$ , ii)  $\mathbf{BP}_k$ , iii)  $\mathbf{daBP}_k$ .

*Proof idea.* Consider the case of  $\mathbf{BPC}_k$ , the cases of  $\mathbf{BP}_k$  and  $\mathbf{BIT}_k$  (which implies the result for  $\mathbf{daBP}_k$  by Lemma 7.13) is similar. For simplicity assume that the set of control states given to  $\text{COVER}$  consists of a single state.

We reduce  $\text{COVER}_{\mathbf{BPC}_k}$  to the coverability problem for WSTSs and then apply Theorem 3.3.

Fix process template  $P$ ,  $k \in \mathbb{N}_0$ , and a single control state  $c$  in  $\text{COVER}_{\mathbf{BPC}_k}$ . For given  $P$ ,  $k$  build the transition system  $M$  defined as before. Consider the instance of coverability problem for WSTS  $M$  with control states set  $\{(G_1, s) \mid s(1) = c\}$  where  $G_1$  is the graph with a single vertex.  $M$  is a WSTS (by Lemma 7.18) with downward closed initial states, hence by Theorem 3.3 we can compute the answer to the coverability problem for  $M$  which is equal to the answer to  $\text{COVER}_{\mathbf{BPC}_k}$  for  $P^{\mathbf{BPC}_k}$ . Hence,  $\text{COVER}_{\mathbf{BPC}_k}$  for AHNs is decidable. □

7. Putting it all together: the parameterized model checking algorithm. Let us develop the parameterized model checking algorithm that solves  $\text{COVER}_{\mathbf{G}}$  for AHNs for  $\mathbf{G} \in \{\mathbf{BPC}_k, \mathbf{BP}_k, \mathbf{BIT}_k\}$ . The soundness and completeness of the algorithm will follow from Theorem 7.19 and Theorem 3.3. The algorithm computes predecessors of  $C$  using procedure  $\text{BPred}$  from Algorithm 1 until the fixpoint is reached, and checks if predecessors has a non-empty intersection with the initial states. Algorithms of such form are called set saturation algorithms in Finkel and Schnoebelen [2001].

For simplicity assume that the set of control states given to COVER consists of a single state called  $c$ , thus  $C = \{c\}$ . Also, let  $G_c$  denote the graph with a single node in state  $c$ .

---

**Algorithm 2:** Solving COVER.

---

```

Input:  $P, c, G \in \{\mathbf{BP}_k, \mathbf{BPC}_k, \mathbf{BIT}_k\}$ 
Output: the answer to COVERG for  $P, c, G$ 
 $b\_reach = \emptyset$ 
 $b\_reach' = \{G_c\}$ 
while  $b\_reach \neq b\_reach'$  :
   $b\_reach = b\_reach'$ 
   $b\_reach' = \text{BPred}(P, G, b\_reach)$ 
  if  $b\_reach'$  contains an initial system state :
    return YES
  return NO

```

---

**Example 7.20** Let us apply Algorithm 2 to the process template  $P_{re}$  from the running example in Figure 7.1, the parameterized graph **BP<sub>2</sub>** (“stars”), and for different control states  $C$ .

First, consider  $C = \{err\}$ . Thus, we answer PMC question (i) from Section 7.1 specialized to **BP<sub>2</sub>**: “In AHNs composed of processes of  $P_{re}$ , is there a graph from **BP<sub>2</sub>** and a process that reaches  $err$ ?“

- The first iteration gives:  $\text{BPred}(P, \mathbf{BP}_2, \{G_{err}\}) = \{err, init - l'\}^9$ .
- The second iteration gives:  $\text{BPred}(P, \mathbf{BP}_2, \{err, init - l'\}) = \{err, init - l'\}$ . Fixpoint!
- $\{err, init - l'\}$  does not contain an initial system state, hence return NO.

Now consider  $C = \{hub\}$  and thus the question is “In AHNs composed of processes of  $P_{re}$ , is there a graph from **BP<sub>2</sub>** and a process that reaches  $hub$ ?“

- The first iteration gives:  $\text{BPred}(P, \mathbf{BP}_2, \{G_{hub}\}) = \{hub, init - u, init - u', u - u'\}$ .
- For simplicity, consider only labeled graph  $init - u'$  calculated in the previous item:  $\text{BPred}(P, \mathbf{BP}_2, \{init - u'\}) = \{init - u', init - init - init\}$ . The set contains initial system state  $init - init - init$ , hence return YES.

## 7.6 DECIDABILITY RESULTS FOR LOSSY AD HOC NETWORKS

Recall from Section 7.2 that in LAHNs a broadcast message can be non-deterministically lost by some of the receivers.

<sup>9</sup>Here,  $a$  denotes the graph consisting of a single node labeled  $a$ . Also,  $a - b$  denotes the graph consisting of two connected vertices labeled  $a$  and  $b$ . Finally,  $a - b - c$  denotes the graph which is a pipeline of three nodes labeled  $a, b$ , and  $c$ , respectively.

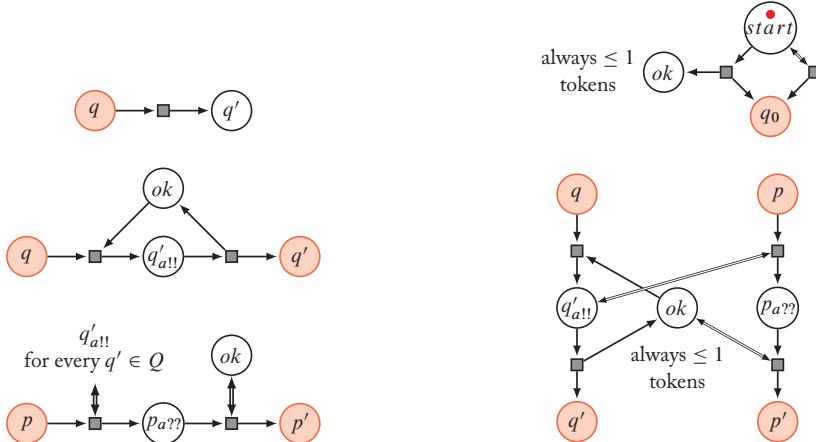
## 124 7. AD HOC NETWORKS

The following two theorems were proved for *mobile* ad hoc networks, but as shown in Delzanno et al. [2012b, Proposition 1], mobile ad hoc networks are equivalent to lossy ad hoc networks on cliques.

**Theorem 7.21** [Delzanno et al., 2010, Corollary 2]  $\text{COVER}_C$ ,  $\text{REPEAT}_C$ ,  $\text{TARGET}_C$  are decidable for LAHNS.

*Proof idea.* The problems can be reduced to decidable problems for Petri nets (or equivalently to vector addition systems) described in Chapter 3. The reduction works only for systems arranged in cliques, so does not apply to  $\mathbf{BP}_k$ ,  $\mathbf{BD}_k$ ,  $\mathbf{BPC}_k$ , etc.

To reduce the problems for LAHN to decidable problems for Petri nets the authors suggest the construction of the Petri net shown in Figures 7.10 and 7.11 (the construction depends on the process template of a LAHN under consideration). The construction ensures that the answer to a given LAHN problem is YES if and only if the answer to a corresponding problem for the Petri net is YES.



**Figure 7.10:** Top to bottom, Petri net constructions for transitions: internal, broadcast-send  $q \xrightarrow{a!!} q'$ , and receive  $p \xrightarrow{a??} p'$ .

**Figure 7.11:** On the top—the initialization, on the bottom—illustration of broadcast synchronization  $q \xrightarrow{a!!} q'$  and  $p \xrightarrow{a??} p'$ .

Assuming that  $\text{COVER}$ ,  $\text{REPEAT}$ ,  $\text{TARGET}$  has control sets with a single state, the reductions are the following.

- $\text{COVER}_C$  is reduced to the coverability for Petri nets: the control configuration has “1” in a place that corresponds to a control state, and “0” in all other places.
- $\text{REPEAT}_C$  is reduced to the repeated coverability for Petri nets: the control configuration is defined similarly to the previous case.

- $\text{TARGET}_{\mathbf{C}}$  is reduced to the reachability for Petri nets, but the Petri net constructed according to Figures 7.10 and 7.11 additionally has:
  - place  $end$ ,
  - transition from  $ok$  to  $end$ , and
  - transition from  $end, q$  to  $end$  for every state  $q$  of the process template.

Then the control configuration is: “1” in place  $end$ , and “0” in all other places.

Now let us look how the reduction works.

*Direction “Run in the LAHN  $\rightarrow$  computation in the Petri net”:* take a run in the LAHN that satisfies a given problem property, let us build a run in the Petri net that also satisfies the corresponding property. First, the Petri net builds an initial state of the LAHN (Figure 7.11, top): the Petri net “pumps” an arbitrary number of tokens from  $start$  into place  $q_0$ , then the token from  $start$  moves into  $ok$ . At this moment the configuration of the Petri net (with  $n$  tokens in  $q_0$ ) corresponds to the initial state of the LAHN with  $n$  processes.

After the initial configuration is built, the Petri net starts simulating transitions of the LAHN. Consider the case of a broadcast transition: send  $q \xrightarrow{a!!} q'$  and receive  $p \xrightarrow{a??} p'$  (Figure 7.11, bottom). Suppose there is a process in state  $q$  in the LAHN and some number of processes in state  $p$  – in the Petri net this corresponds to a token in place  $q$ , and a number of tokens in  $p$  (the number of tokens in  $p$  is equal to the number of processes in state  $p$ ). Let the  $q$ -process and  $m$  processes in state  $p$  take the broadcast synchronized transition—in the Petri net this corresponds to a sequence of transitions that moves a single token from  $q$  to  $q'$  and  $m$  tokens from  $p$  to  $p'$  (and no tokens are left in  $p_{a??}$ ). This completes the description of the simulation of the broadcast transition. The simulation ensures that the computation of the Petri net satisfies the corresponding property (over Petri net configurations).

*Direction “Faithful computation in the Petri net  $\rightarrow$  run in the LAHN.”* We call *faithful computation* a computation in the Petri net that does not leave tokens in place  $p_{a??}$  between broadcast simulations. Then the proof of this direction follows from two claims.

- If there is a computation that (repeatedly) covers the corresponding control configuration mentioned in the reductions before, then there is a faithful computation in the Petri net that does this,
- Any faithful computation of the Petri net has a corresponding run in the LAHN.

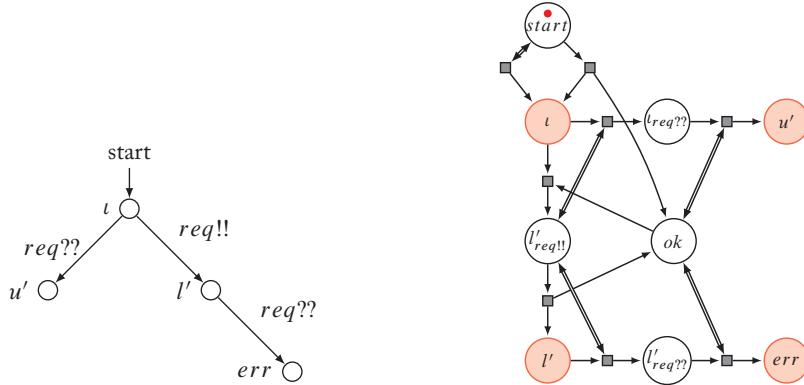
The intuition behind item (i) is as follows. If in a non-faithful computation there are tokens left in state  $p_{a??}$ , then in a faithful corresponding computation we move all the tokens from  $p_{a??}$  to state  $p'$  and do not move them until the tokens in  $p_{a??}$  get moved in the original non-faithful computation. The second item (ii) follows from the observation that any faithful transition of the Petri corresponds to some transition in the LAHN.

## 126 7. AD HOC NETWORKS

Finally, the decidability of  $\text{COVER}_C$ ,  $\text{TARGET}_C$ ,  $\text{REPEAT}_C$  for LAHNs follows from the decidability of corresponding problems for Petri nets.  $\square$

**Example 7.22** Consider process template in Figure 7.12(a) which is a part of the running example process template  $P_{re}$  in Figure 7.1 from Section 7.1. Constructing the Petri net according to Figures 7.10 and 7.11 gives the net in Figure 7.12(b).

Consider PMC question (i) from Section 7.1 adapted to LANHs: “Is there a LAHN build from process template in Figure 7.12(a) that reaches state  $err$ ?.” It is easy to see that two tokens in the initial state  $i$  are enough to cover state  $err$  in the Petri net: consider computation where, first, one token moves from  $i$  to  $l'$ , and then both tokens move: the token in  $l'$  moves to  $err$  while the token from  $i$  moves to  $l'$ . Hence, the answer to our PMC question is YES.



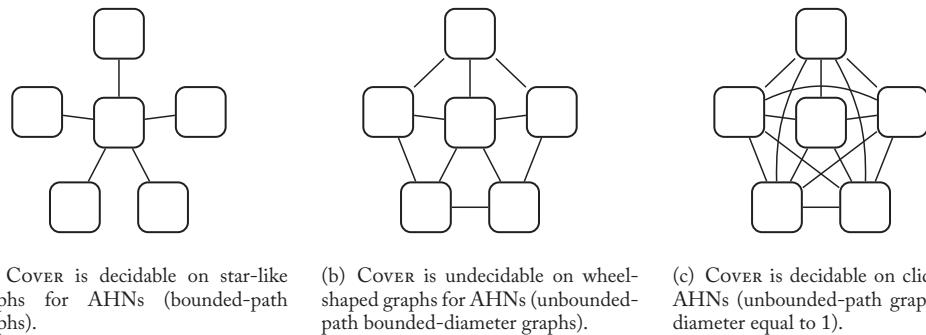
**Figure 7.12:** Example of the conversion of a protocol into a Petri net simulating LAHNs.

**Note on complexities.** The proof of Theorem 7.21 reduces  $\text{COVER}_C$  and  $\text{TARGET}_C$  for LAHNs to EXPSPACE-complete and -hard problems on Petri nets, and does not provide the lower bound. In the follow-up paper [Delzanno et al. \[2012a\]](#) explore the complexities. We state their results without providing the proofs.

**Theorem 7.23** [[Delzanno et al., 2012a](#), Theorem 1 and 2] For LAHNs  $\text{COVER}_C$  is P-complete and  $\text{TARGET}_C$  is NP-complete (with respect to the size of the process template).

## 7.7 DISCUSSION

In ad hoc networks, the nature of the parameterized connectivity graph separates the decidable cases from undecidable. Delzanno et al. [2010, 2011] gave the first characterization of those graphs on which COVER is decidable ( $\mathbf{BPC}_k$ ) and on which it is undecidable ( $\mathbf{BD}_k$ ). Figure 7.13 illustrates the decidability boundary for this problem. If we consider the more expressive problems



**Figure 7.13:** The decidability boundary for Cover for AHNs.

REPEAT or TARGET in ad hoc networks without losses, then we get undecidability for all the graphs considered in this chapter.<sup>10</sup> Finally, in ad hoc networks with lossy broadcast transitions, all these problems are decidable. Table 7.1 summarizes the decidability results for ad hoc networks.

Decidability proof techniques in both non-lossy and lossy cases are based on well-structuredness of the systems, and do not exhibit cutoffs that are independent of the process template as in Chapter 4.

Undecidability proofs, similarly to other chapters, reduce from the halting problem for 2CMs. Thus, for a given 2CM, the proofs build a protocol that simulates its execution. In contrast to other chapters, the exact connectivity graph is unknown *a priori* (what is known is a class to which the graph belongs to), so the protocols for simulating a given 2CM have two steps. In the first step they ensure the right form of the connectivity graph, and the second step does the simulation relying on the particular form of the graph. Another difference is the case of directed acyclic graphs (Theorem 7.12) where the information from process to process flows in one direction only. Thus the controller that simulates the control of a given 2CM cannot receive the feedback from processes it broadcasts messages to. In this case the undecidability proof uses the fact that TMs can be simulated by iterating finite-state transducers.

<sup>10</sup>Possibly except for directed parameterized connectivity graph  $\mathbf{dabP}_k$ .

**Table 7.1:** Decidability results for Ad Hoc Networks and their sources

Result	Network type	Graphs	Specification	References
undecidability	AHN	All	COVER	[Delzanno et al. 2010]
undecidability	AHN	<b>BD<sub>k</sub></b>	COVER	[Delzanno et al. 2011]
undecidability for $k \geq 2$	AHN	<b>BP<sub>k</sub></b>	TARGET, REPEAT	[Delzanno et al. 2010]
undecidability	AHN	<b>daAll</b>	COVER	[Abdulla et al. 2013a]
decidability	AHN	<b>BPC<sub>k</sub>,</b> <b>BP<sub>k</sub>,</b> <b>daBP<sub>k</sub></b>	COVER	[Delzanno et al. 2010]
				[Delzanno et al. 2011]
				[Abdulla et al. 2013a]
decidability	LAHN	C	COVER, REPEAT, TARGET	[Delzanno et al. 2010]

### 7.7.1 VARIATIONS OF THE MODEL

Delzanno et al. [2012b] study AHNs with many other failure types (we considered only message losses). They distinguish between failures in nodes and in communication.

Node failures include non-deterministic restarts and crashes. They show by reduction from the undecidable COVER in AHNs (without failures) that the PMCPs for such models are undecidable.

Communication failures include non-deterministic message losses (covered here as LAHNS), message conflicts, and message conflicts with conflict detection. A message conflict happens when a node senses broadcasts from two or more nodes simultaneously. The model with message conflicts makes COVER decidable (and the proof uses the ideas from the LAHN section), while the model with conflict detection makes the PMCPs undecidable (via reduction from undecidable COVER for AHNs).

Abdulla et al. [2011] studied PMC problems for AHNs composed of processes being timed automata, and Delzanno et al. [2013] studied the case of LAHNS of processes being register automata.

Delzanno and Traverso [2013] studied PMC problems for AHNs where broadcasts are asynchronous. In this model each process has an unbounded (local) buffer for incoming messages. When a process broadcasts a message, it is enqueued into buffers of all receiving processes, and later the processes asynchronously handle the messages (in our model broadcasts are synchronous, i.e., a broadcast message is immediately and simultaneously handled by all receiving processes).

## CHAPTER 8

# Related Work

In this book we focused on established decidability results in parameterized model checking. As this topic is a lively field, we cannot make any claim about completeness of our survey. In particular, we left out the large body of research that includes invariant-based techniques, regular model checking, symbolic methods, techniques for counter automata, and abstraction-based methods. In this chapter we give a brief overview of these methods and give further references to the interested reader. Our classification of the techniques is unavoidably subjective, as except for several cornerstone papers, many techniques adopted multiple different ideas.

For the reader interested in practical applications of parameterized model checking techniques, we give an overview of PMC tools in Chapter 9.

### 8.1 ABSTRACTION TECHNIQUES

The general idea of abstraction-based techniques is as follows. One identifies a class of process templates that have a common structure.<sup>1</sup> To construct a new abstraction, one defines an abstraction mapping  $\alpha$  that builds a finite-state system  $\alpha(P)$  of a given process template  $P$ . Further, one proves that  $\alpha$  preserves the specifications from a predefined class  $\mathcal{S}$ : for every specification  $\varphi \in \mathcal{S}$  and every process template  $P$ , if  $\alpha(P) \models \varphi$ , then for every number of processes  $n \geq 1$ , it holds that  $P^n \models \varphi$ . This proof is the most complicated part of the technique. Typically, the proof is done by showing that  $\alpha(P)$  simulates each system  $P^n$ , for  $n \geq 1$ , which implies that  $\alpha(P)$  satisfies all  $\text{ACTL}^*$ -formulas that are satisfied by every system instance from  $P^n$  for  $n \geq 1$  (see Clarke et al. [1999, Theorem 16]). Hence, for a given process template  $P$ , the parameterized model checking problem amounts to constructing the abstraction  $\alpha(P)$  and checking whether the property holds for  $\alpha(P)$ .

Abstraction introduces new behavior and thus, when  $\alpha(P)$  does not satisfy the property  $\varphi$  to be verified, one cannot immediately conclude whether the property is satisfied in the parameterized family  $\{P^n \mid n \geq 1\}$ . If  $\alpha(P) \not\models \varphi$ , the model checking tool will provide a counterexample, and the user has to analyze the counterexample with the goal of refining the abstraction  $\alpha(P)$ . Sometimes,  $\alpha(P)$  can be refined automatically, see e.g., Abdulla et al. [2010] and John et al. [2013]. However, refinement techniques for parameterized model checking are, by design, incomplete. Therefore, abstraction-based techniques do not characterize decidable classes of the parameterized model checking problem, in contrast to the techniques described in this book.

<sup>1</sup>For instance,  $\{0, 1, \infty\}$ -counter abstraction by Pnueli et al. [2002] is applicable to process templates that can be represented with the SLP language.

## 130 8. RELATED WORK

Notwithstanding, this incompleteness property, abstraction methods have been successfully applied to many practical examples.

Pnueli et al. [2002] highlighted  $\{0, 1, \infty\}$ -counter abstraction. There, a state of the abstraction  $\alpha(P)$  consists of global variables over some finite domain and abstract counters  $\kappa_1, \dots, \kappa_m$  over the domain  $\{0, 1, 2\}$ . Counter  $\kappa_i$  represents the abstract number of processes in the  $i$ -th local state as follows: value 0 corresponds to zero processes; value 1 corresponds to exactly one process; value 2 corresponds to “many” processes (i.e., more than one). By using the finite abstract domain to count processes, one constructs the finite abstraction  $\alpha(P)$  for a process template  $P$ . The atomic propositions allow one to check how many processes there are in a certain local state, e.g.,  $\kappa_{\ell_1} = 1$  asserts that there is exactly one process in the local state  $\ell_1$ , while  $\kappa_{\ell_2} > 1$  asserts that that several processes are in the state  $\ell_2$ . Pnueli et al. [2002] showed that for every 1-LTL\X-formula  $\varphi$ , if  $\alpha(P) \models \varphi$  holds, then  $\forall n \geq 1. P^n \models \varphi$  holds as well. It is easy to see that this abstraction works especially well for mutual exclusion properties: the formula  $G(\kappa_{\ell_C} < 2)$  expresses that never two processes are in the critical section at the same time. Pnueli et al. [2002] applied their abstraction to reason about safety and liveness of Szymanski’s mutual exclusion algorithm and of simplified Bakery. The abstractions were checked with the NuSMV model checker [Cimatti et al., 2002].

Ip and Dill [1999] discussed an abstraction similar to  $\{0, 1, \infty\}$ -counter abstraction, but they focused only on safety properties. They implemented their abstraction in Murφ [Dill, 1996, 2008] and verified safety of Peterson’s mutual exclusion algorithm and of the cache coherence protocols MESI, ICCP, and DCCP.

John et al. [2013] generalized  $\{0, 1, \infty\}$ -counter abstraction to counter abstraction over *parametric intervals* (PIA-counter abstraction). In their abstraction, each counter is assigned an abstract value that corresponds to an interval, e.g., if  $\kappa_i = [(n+1)/2, n+1]$ , then a majority out of  $n$  processes is in state  $\ell_i$ . Hence,  $\{0, 1, \infty\}$ -counter abstraction is a special case of PIA-counter abstraction over the intervals  $[0, 1)$ ,  $[1, 2)$ , and  $[2, n+1)$ . John et al. [2013] implemented their abstraction in the tool ByMC and verified safety and liveness of reliable broadcast algorithms in the presence of a parameterized number of faults (e.g., crash and Byzantine). The abstract systems were checked with the Spin model checker [Holzmann, 2003].

Clarke et al. [2008] introduced environment abstraction, which can also be seen as a generalization of  $\{0, 1, \infty\}$ -counter abstraction. In their framework, an abstraction of a global state is a formula that describes the state of a special process (called the *reference* process) and a summary of the states of the other processes (called the *environment*). A system transition is abstracted as a step of the reference process as follows: the reference process checks whether the environment satisfies the precondition; if so, the reference process changes its local state and the state of the environment. After the transition has been made, another process in the environment can become the reference process. The abstract semantics of the transitions is defined individually for each protocol class. Clarke et al. [2008] applied their technique to verify the following protocols: readers/writers, Szymanski’s mutual exclusion, simplified Bakery, German’s cache coherence pro-

tocol, and the Flash cache coherence protocol. Their work was extended by [Talupur and Tuttle \[2008\]](#).

[Basler et al. \[2009\]](#) applied  $\{0, 1, \infty\}$ -counter abstraction on boolean programs to verify concurrent programs with the parameterized number of threads. They proposed a symbolic exploration algorithm that has been implemented in the tool BOOM and scales well for boolean programs in practice.

[Jensen and Lynch \[1998\]](#) introduced a parameterized simulation relation and used it to prove soundness of their specific abstraction of Burns' mutual exclusion algorithm. In their abstraction, they mapped a system of  $n$  processes to a system of two processes  $AP_0$  and  $AP_1$ : intuitively,  $AP_0$  and  $AP_1$ , respectively, represent the processes with indices  $i$  and  $j$  for  $1 \leq i < j \leq n$ . [Jensen and Lynch \[1998\]](#) proved soundness of the abstraction in the Larch Proof Assistant and verified safety of the abstraction with Spin.

[Calder and Miller \[2003\]](#) verified the Firewire protocol with Spin and introduced a specific abstraction for the parameterized case.

## 8.2 REGULAR MODEL CHECKING

Regular model checking (RMC) is a symbolic technique built upon automata representation of transition relations; see [Abdulla \[2012\]](#) for a detailed survey. RMC assumes a linear order  $\prec$  on a set of processes. Such an order comes naturally when the processes of a parameterized system are organized in a ring or a line. Let  $P$  be a process template with a *finite* set  $Q$  of local states. Then, a global state of a system  $P^n$  is represented as a word  $w \in Q^n$  of length  $n$ , where the letter  $w[i]$  corresponds to the local state of the  $i$ -th process (in the order  $\prec$ ) for  $1 \leq i \leq n$ .

In RMC, the set of global states  $S$  is required to be a language of a finite-state automaton  $A_S$  over the alphabet  $Q$ . Similarly, a finite-state automaton  $A_R$  over the alphabet  $Q \times Q$  represents the union of transition relations in the parameterized family  $\{P^n \mid n \geq 1\}$ . As the automaton  $A_R$  models systems that do not dynamically create or destroy processes, the language of  $A_R$  is size-preserving, that is, the language  $\mathcal{L}(A_R)$  of the automaton  $A_R$  is a subset of  $\{(w, w') \mid w, w' \in Q, |w| = |w'|\}$ .

Given a process template  $P$ , one has to construct the following automata that represent the union of the transition systems in the parameterized family  $\{P^n \mid n \geq 1\}$ :

- an automaton  $A_I$  to represent the set of the initial states,
- an automaton  $A_B$  to represent the set of bad states (violating the reachability specification), and
- an automaton  $A_R$  to represent the transition relation.

These automata are given as the input to the RMC technique. Their construction is not part of the RMC technique.

## 132 8. RELATED WORK

The main goal of the RMC technique is to compute the reflexive and transitive closure  $A_R^*$  of the automaton  $A_R$ : an automaton that only accepts pairs of words  $w, w' \in Q^*$  that are connected by a path  $w_1, \dots, w_k \in Q^*$ , that is,  $w = w_1$ ,  $w_k = w'$  and  $(w_i, w_{i+1}) \in \mathcal{L}(A_R)$  for  $1 \leq i < k$ . Once  $A_R^*$  is built, one applies automata-theoretic operations to check whether one of the systems in the family  $\{P^n \mid n \geq 1\}$  violates the specification: let us denote by  $\mathcal{L}(A_R^*(A_I))$  the set of words  $\{w' \in Q^* \mid w \in \mathcal{L}(A_I), (w, w') \in \mathcal{L}(A_R^*)\}$ , which symbolically represents all reachable states. Then, no system  $P^n$  reaches a bad global state (i.e., a state in the language of the automaton  $A_B$ ), if and only if  $\mathcal{L}(A_R^*(A_I)) \cap \mathcal{L}(A_B) = \emptyset$ .

Computing  $A_R^*$  is the main bottleneck of the RMC technique. Indeed, in general, the closure is neither regular, nor computable. Several less general solutions were introduced, e.g., *acceleration* of process transitions [Abdulla et al., 1999], *quotienting* of automata by state merging [Abdulla et al., 2002, Bouajjani et al., 2000], and *abstraction* [Bouajjani et al., 2004].

We make two observations about the representation. First, alphabet  $Q$  is finite and for each  $n \geq 1$ , automaton  $A_S$  accepts finitely many words of length  $n$ . However, as the language of  $A_S$  can be infinite, the automaton is able to capture global states of all systems  $P^n$ , for  $n \geq 1$ . This brings us to the second observation: not every set of global states  $S$  is regular, i.e., there may not be an ordering  $\prec$  under which  $S$  can be encoded with a finite-state automaton, and thus RMC techniques may not apply. The same applies to the automaton  $A_R$ .

Finally, To and Libkin [2010] introduced algorithmic metatheorems that generalize reasoning found in regular model checking research. A tool that implements regular model checking is T(O)RMC [Legay, 2008].

## 8.3 SYMBOLIC TECHNIQUES

Similar to regular model checking (Section 8.2), symbolic techniques do not represent state sets and transition relations explicitly, but use symbolic representations. While RMC is based on one form of symbolic representation, namely, finite automata, the symbolic techniques presented in this section apply various logical theories to represent states and transitions.

Henriksen et al. [1995] proposed to use *monadic second order logic* as a “highly succinct alternative to regular expressions.” Elgaard et al. [1998] implemented their framework in the tool MONA.

General principles for parameterized symbolic model checking were laid out by Kesten et al. [1997]. There, the authors used a *finitary second-order theory of one successor* (FS1S) and FS\* $S$ , which have the same expressive power as finite automata and tree automata respectively. Further, Pnueli and Shahar [2000] investigated liveness and acceleration in this framework. These techniques were implemented in the tool called TLV [Pnueli and Shahar, 1996], and have been re-implemented in JTLV [Pnueli et al., 2010].

Maidl [2001] introduced a framework, in which parameterized systems are represented as formulas over Presburger arithmetic. Apart from local and global variables, the formulas contain special variables that represent process indices. The framework encompasses token passing and

broadcast systems. The authors give a semi-algorithm based on proof tree construction and give sufficient conditions of its termination for safety properties.

**Reachability checking for array-based systems.** There are a number of techniques that are similar to RMC in that they model the state of a parameterized system as a finite word, or equivalently an array of values from some logical background theory, but are different from RMC in that they represent sets of states and the transition relation of the system as constraints in this theory.

A first representative of this approach developed directly out of RMC and was initially called *regular model checking without transducers* [Abdulla et al., 2007]. The system model assumes an order on the processes, and allows them to communicate via shared variables, rendezvous, or broadcast. In addition, transitions can be guarded with global conditions that take into account the order of processes. Systems are verified by an approximate backward reachability check, where potentially infinite sets of system configurations are represented as (finite) constraints on system variables. To ensure termination, the approach considers a *monotonic abstraction* of the transition relation, which guarantees that transitions are monotonic with respect to the subword relation on configurations. As a result, the algorithm can always work on *upwards closed sets* of configurations, and thus one considers an abstraction of the original system which is a *well-structured transition system*. The approach has later been extended by Abdulla et al. [2009] to systems with infinite-state components by also supporting numerical variables. To ensure decidability of the reachability computation, sets of valuations of numerical variables can only be expressed as *gap-order constraints*. The tool Undip implements both the finite- and infinite-state case. As an alternative to monotonic abstraction, the approach for finite-state processes has recently been combined with counter abstraction, which in contrast to monotonic abstraction allows to refine the abstraction if verification is not successful [Ganty and Rezine, 2014]. An implementation of this extension is available in the tool PWC.

Ghilardi et al. [2008] introduced an approach that generalizes the basic idea of representing states and transitions as constraints on arrays over a suitable background theory. They define the notion of *array-based systems*, where a state of the system is represented as an unbounded array of values from a (parametric) first-order theory of elements. As another generalization, the approach also supports a parametric index theory for the array, allowing to model non-linear topologies. Then, sets of states can be represented as formulas over the theories of indices and elements, and transition relations as formulas that relate the possible pre- and post-states. In both cases, formulas can be quantified, making the problem potentially undecidable due to the undecidability of first-order logic. Instead of an abstraction to a well-structured system, this approach uses SMT solvers and quantifier elimination methods to directly reason on these constraints. For combinations of systems and theories such that all quantifiers can be effectively eliminated and the ground satisfiability problem is decidable, one obtains the fully automated technique of *Model Checking Modulo Theories (MCMT)* [Ghilardi and Ranise, 2010]. In addition to an implementation (called MCMT) by its inventors, the approach has been implemented by Conchon et al. [2012] in the tool Cubicle.

## 134 8. RELATED WORK

Completeness of MCMT relies on the one hand on existing decidability results for first-order theories, and on the other hand on proving new results for decidability or the possibility of quantifier elimination in the theories that are necessary for describing systems and properties. For some expressive classes of array-based systems, full model checking may not be possible. In such cases, one may resort to invariant checking or bounded model checking, which can both be expressed as a single first-order constraint over a suitable theory. The framework of *local theory extensions* allows to define configurations and transitions of array-based systems as extensions of a given first-order theory, providing a decision procedure for such constraints [Faber et al., 2010, Jacobs and Sofronie-Stokkermans, 2007].

## 8.4 DYNAMIC CUTOFF DETECTION

Many of the results presented in this survey solve the PMCP by providing a *cutoff*, i.e., a maximal number of processes that need to be considered for a given class of systems and specifications. While such cutoff results let us decide the PMCP, they are restricted to the classes of systems and specifications for which the proof methods have been designed. A related approach is *dynamic cutoff detection*, which does not depend on *a priori* proven cutoff theorems, but instead tries to automatically detect a suitable cutoff for a given system template, and possibly also a given specification.

Kaiser et al. [2010] considered the problem of finding cutoffs that are specific to a given parameterized system and a safety property. To this end, their algorithm first computes the set of reachable *thread-states*, i.e., combinations of shared and local state, in a system of fixed size  $k$ . For the class of shared-memory systems under consideration, a sufficient and necessary condition for the existence of new reachable thread-states in systems of size  $n > k$  is identified. In a nutshell, if any new thread-states are reachable for  $n > k$ , then among those there is one thread-state  $(s, l)$  with minimal distance from the initial state. Furthermore, by the characteristics of their model, this thread-state  $(s, l)$  must be reachable from those that are reached in a system of size  $k$  by a transition that does not change the shared memory part  $s$ . This gives rise to an analysis for *thread-state candidates* that could be reached in a system of size  $k + 1$ . Whether this actually is the case can be decided for each candidate by a backward coverability analysis, which is decidable for fixed  $k$ . Since there can only be a finite number of candidates, this procedure will terminate for the given  $k$ . If none of the candidates can be reached, then  $k$  is a cutoff, otherwise the procedure is iterated for  $k + 1$ .

Abdulla et al. [2013b] also detected property- and system-specific cutoffs for safety properties. Their system model is close to the one presented in Chapter 6, with connectivity graphs that can be different from cliques and guarded updates that can take into account this topology. To find a cutoff, the procedure computes in parallel the exact set of reachable states for a system of size  $k$ , and an over-approximation of the reachable states with respect to a *view abstraction*, also parameterized by  $k$ . Roughly, for this overapproximation we view a global state as a word over local states of the system, and not only consider words that are reachable, but also words that

can be constructed from subwords of reachable words. By a separate argument, the authors show that for their class of systems it is sufficient to consider words of size  $k + 1$  in this procedure in order to obtain completeness. If the concrete computation finds an error trace, then the given safety property does not hold. If the abstract computation does not find an error trace, then  $k$  is a cutoff and the safety property holds for all instances of the system. If neither is the case, then the procedure is iterated for  $k + 1$ .

The dynamic cutoff techniques are applicable to very expressive classes of systems, such that the PMCP is in general undecidable even for safety properties. Thus, the search for a suitable cutoff can in general not be guaranteed to terminate, but only in restricted cases that are decidable. For instance, the approach by Abdulla et al. [2013b] is guaranteed to terminate for well-quasi ordered transition systems. Cutoffs that are specific to a property and a process template can be much smaller than pre-determined cutoffs that hold for *all* elements of a class of systems and properties.

## 8.5 NETWORK INVARIANTS

Invariant-based techniques use inductive arguments to reason about parameterized systems. Intuitively, these arguments state that if a certain property is satisfied in a system of  $n$  processes it also holds in a system of  $n + 1$  processes. There is a number of such techniques in the literature that apply induction to reason about parameterized systems. The techniques discussed here thus exploit invariants over a sequence of systems, rather than invariants of computations, that is, over a sequence of steps. The techniques discussed differ in the following ingredients.

**(invariant specification)** This defines the object that is invariant for a sequence of system instances, e.g., an invariant can be a labeled transitions system that simulates every system instance in the sequence.

**(invariant preservation)** A formal definition of what it means for a system instance  $P^n$  to preserve an invariant, e.g., simulation.

**(inductive step)** An inductive step to prove that a system instance  $P^{n+1}$  preserves an invariant, if a system instance  $P^n$  does.

The first results on using induction for parameterized model checking were obtained independently by Kurshan and McMillan [1989] and Wolper and Lovinfosse [1989]. Wolper and Lovinfosse [1989] introduced the framework of *network invariants*. Kurshan and McMillan [1989] introduced the *structural induction theorem*.

In the framework of network invariants, an invariant  $I$  is specified in the same language as a process template  $P$ , namely, as a process in the framework of Hoare's Communicating Sequential Processes (CSP). Invariant preservation is defined with the means of an implementation relation  $\preceq$  on labeled transition systems, e.g., trace inclusion, bisimulation, or observational equivalence. The technique consists of proving two properties for some  $k \geq 1$ :

## 136 8. RELATED WORK

- the base system  $P^k$  implements  $I$ , i.e.,  $P^k \preceq I$ , and
- the parallel composition of  $I$  and  $P$  implements  $I$ , i.e.,  $I \parallel P \preceq I$ .

[Wolper and Lovinfosse \[1989\]](#) showed by induction that whenever  $P$  and  $I$  satisfy these two properties, it holds that  $P^n \preceq I$  for  $n \geq k$ . Hence to prove a property for all  $n$ , it just remains to check the specification against  $I$ , and against instances  $P^1, \dots, P^{k-1}$ . The crucial point of this method is how to generate invariants. One may try  $I = P$  or  $I = P^2$ , which works for simple examples. If this guess fails, an invariant candidate must be provided by the user.

[Kurshan and McMillan \[1989\]](#) obtained similar results in the form of the structural induction theorem in the framework of Milner’s Calculus of Communicating Systems (CCS). They also note that one can use various implementation relations, e.g., “may” preorder on CCS processes, language containment, strong and weak bisimulation, or the “implements” relation of I/O automata.

**Network invariants and network grammars.** [Shtadler and Grumberg \[1990\]](#) extended the framework of network invariants to parameterized systems whose connectivity graph is derived by a network grammar. In their work, they used a special form of computation equivalence.

[Clarke et al. \[1995\]](#) also used network grammars to capture the connectivity graph. Instead of equivalence relations, e.g., bisimulation, they use a preorder on labeled transition systems, namely, simulation. The key feature of their work is to specify expected behavior with automata over regular expressions,<sup>2</sup> as opposite to indexed temporal logics. For invariant generation, [Clarke et al. \[1995\]](#) gave a method to generate invariant candidates. The method takes the specification as input and has better chances of generating useful invariants than a more general technique that does not consider the specification. Still, if the generated invariant candidate is not an invariant, the user must help the verification tool. The original framework was formulated for synchronous systems. Later, [Clarke et al. \[1997\]](#) extended the framework to asynchronous systems.

[Lesens et al. \[1997\]](#) followed up the work by [Clarke et al. \[1995\]](#) and introduced several invariant synthesis heuristics based on *widening techniques*. If the technique fails to find an invariant, the user can manually refine the widening operator and thus give a hint to the technique. This is the key improvement over other techniques, which cannot be tuned by the user. [Kesten and Pnueli \[2000\]](#) transferred the framework of network invariants to fair-discrete systems, in which processes communicate via shared variables, as opposed to rendezvous communication used in the other work discussed in this section. [Konnov and Zakharov \[2010\]](#) also applied network grammars to generate connectivity graphs. As simulation is typically too restrictive for asynchronous systems, they introduced several weak forms of simulation: block simulation, quasi-block simulation, and semi-block simulation. They have implemented the technique in the tool **CHEAPS**.

<sup>2</sup>Similar to regular model checking, the alphabet is the set of local states  $\mathcal{Q}$ .

## 8.6 INVISIBLE INVARIANTS

The notion of invariants used in the techniques based on invisible invariants is closer to the one used in the analysis of programs (loop invariants) or algorithms. Pnueli et al. [2001] introduced an approach to invariant detection in concurrent programs parameterized by the number of processes. The method targets verification of  $k$ -indexed formulas, e.g., the 2-indexed formula  $\forall i, j \neq i. G(at_i \neq critical \vee at_j \neq critical)$ . In the technique of invisible invariants, a model checker is run on a fixed-size system instance to collect the reachable *global* states  $R$ . Given  $k$ , the technique projects the reachable global states  $R$  on all combinations of  $k$  process indices and thus computes the reachable combinations of the *local* states of  $k$  processes. The result of this projection—that is, the set of combinations of local states—is called an *invisible invariant*. For parametrized model checking, Pnueli et al. [2001] proved a small model property for concurrent programs with (bounded) shared variables, from which follows that an invisible invariant of a fixed-size system instance is also an invariant of the parameterized system.

Fang et al. [2006] combined invisible invariants with ranking functions, in order to verify liveness properties. McMillan and Zuck [2011] investigated connections between invisible invariants and abstract interpretation. Johnson and Mitra [2012] extended the small model theorem by Pnueli et al. [2001] to the networks of rectangular hybrid automata. They implemented their technique in the tool PASSEL [Johnson, 2013].

## 8.7 OTHER ASPIRING APPROACHES

Fully symmetric parameterized systems, e.g., cache coherence protocols, can be naturally encoded as *counter automata* [Delzanno, 2003, Leroux and Sutre, 2005]. A counter automaton is a graph, whose edges are labeled with counter updates and guards over counters and parameters (such as the number of processes). Typically, the guards and updates are expressed in linear integer arithmetic. Delzanno [2003] applied backward reachability on the systems with linear arithmetic constraints to analyze cache coherence protocols. Leroux and Sutre [2005] introduced flat (and flatable) counter automata, whose reachability can be checked with terminating acceleration techniques. This technique is implemented in the tool FAST [Bardin et al., 2008].

Esparza et al. [2013] introduced *non-atomic networks* to model parameterized asynchronous shared memory systems. A non-atomic network consists of a single leader and many contributors that write to and read from a shared register store. The authors showed that safety verification of non-atomic networks of processes modeled with state machines is decidable and at least PSPACE-hard. Further, they showed that safety verification of non-atomic networks of processes modeled with pushdown machines is undecidable.

Bouajjani et al. [2008] studied parameterized model checking of *resource allocation systems* (RASs). Such systems have a bounded number of resources, each owned by at most one process at any time. Processes are pushdown automata, and can request resources with high or normal priority. RASs are similar to conjunctive guarded protocols in that certain transitions are disabled

## 138 8. RELATED WORK

unless a process has a certain resource. RAs without priorities and with processes being finite state automata can be converted to conjunctive guarded protocols (at the price of blow up), but not vice versa. The authors study parameterized model checking wrt.  $LTL \setminus X$  properties on strong-fair or all runs, and wrt. (local or global) deadlocks on all runs. The proof structure resembles that of [Emerson and Kahlon \[2000\]](#).

[Farzan et al. \[2014\]](#) introduced *counting proofs* to reason about parameterized shared memory programs. Their technique automatically generates auxilliary counters and synthesizes a small proof of partial correctness of a concurrent program. [Farzan et al. \[2015b\]](#) also introduced *proof spaces*, where they construct a finite set of correctness proofs that capture all traces of a parameterized program. The proofs are constructed from predicate automata [[Farzan et al., 2015a](#)], an infinite-state generalization of alternating finite automata.

## CHAPTER 9

# Parameterized Model Checking Tools

The results covered in this book show that the parameterized model checking problem is only decidable for rather restricted classes of system models and specifications. In addition to that, parameterized model checking is computationally hard even in cases where it is decidable. As a result, there are only few software tools that implement decision procedures for the PMCP, and most of the available tools are implementations of the semi-decision procedures covered in Chapter 8. For researchers who want to get some hands-on experience with parameterized model checking, we give an overview of parameterized model checking tools that are available at the time of this writing. Table 9.1 summarizes the approaches of the available tools, and Table 9.2 shows where they can be obtained.

The only tool that uses decidability results from this book is PARTY [Khalimov et al., 2013b]. PARTY supports synthesis of concurrent systems in a token ring topology, with specifications in indexed LTL\X. It uses the cutoff results from Chapter 4, together with an SMT-based approach to solve the synthesis problem for a process template that satisfies the specification in the cutoff topology. By constraining the implementation of the process template such that it only allows one implementation, the approach can also be used to solve the PMCP in token rings.

Several of the approaches from the previous chapter have been implemented in tools such as BOOM [Basler et al., 2009], T(O)RMC [Legay, 2008], MONA [Elgaard et al., 1998], TLV [Pnueli and Shahar, 1996] and JTLV [Pnueli et al., 2010], Undip [Abdulla et al., 2009], MCMT [Ghilardi and Ranise, 2010], CHEAPS [Konnov and Zakharov, 2010], Cubicle [Conchon et al., 2012], ByMC [John et al., 2013], and PCW [Ganty and Rezine, 2014]. We refer to the previous chapter for an explanation of their respective approaches to parameterized model checking.

Furthermore, there is a number of tools that implement more general forms of infinite-state model checking, in particular with support for integer-valued variables. With a suitable encoding of parameterized systems into systems with integer variables, these tools can be used for parameterized model checking. Examples of such tools are ALV [Yavuz-Kahveci and Bultan, 2009], BRAIN [Rybina and Voronkov, 2002], FAST [Bardin et al., 2008], and nuXmv [Cavada et al., 2014].

Finally, there is a number of tools for the verification of hybrid systems, like KeYmaera [Platzer, 2012, Platzer and Quesel, 2008] and Passel [Johnson, 2013]. Both systems allow

## 140 9. PARAMETERIZED MODEL CHECKING TOOLS

**Table 9.1:** Parameterized model checking tools: approaches

Tool	Authors	Implements
ALV	Yavuz-Kahveci and Bultan [2009]	approximate infinite-state reachability
BRAIN	Rybina and Voronkov [2002]	symbolic infinite-state reachability
BOOM	Basler et al. [2009]	counter abstraction
ByMC	John et al. [2013]	counter abstraction
CHEAPS	Konnov and Zakharov [2010]	network invariants
Cubicle	Conchon et al. [2012]	symbolic backward reachability
FAST	Bardin et al. [2008]	acceleration-based inf.-state reachability
IIV	Balaban et al. [2005]	invisible invariants
JTLV	Pnueli et al. [2010]	semi-automatic verification platform
KeYmaera	Platzer and Quesel [2008]	semi-automatic verification platform
Murφ	Dill [1996]	counter abstraction
MCMT	Ghilardi and Ranise [2010]	symbolic backward reachability
MONA	Henriksen et al. [1995]	monadic second-order logic
nuXmv	Cavada et al. [2014]	infinite-state model checking
PARTY	Khalimov et al. [2013b]	(static) cutoff detection
Passel	Johnson [2013]	invisible invariants
PCW	Ganty and Rezine [2014]	symbolic reachability, counter abstraction
TLV	Pnueli and Shahar [1996]	semi-automatic verification platform
T(O)RMC	Legay [2008]	regular model checking
Undip	Abdulla et al. [2009]	approximate backward reachability

for the verification of distributed hybrid systems with a parametric number of components. While Passel is based on a completely automatic combination of reachability checking and detection of invisible invariants, KeYmaera is a very powerful semi-automatic theorem prover.

**Table 9.2:** Parameterized model checking tools: availability

Tool	URL
ALV	<a href="http://www.cs.ucsb.edu/~bultan/composite/">http://www.cs.ucsb.edu/~bultan/composite/</a>
BOOM	<a href="http://www.cprover.org/boom/">http://www.cprover.org/boom/</a>
BRAIN	<a href="http://www.cs.man.ac.uk/~voronkov/BRAIN/">http://www.cs.man.ac.uk/~voronkov/BRAIN/</a>
ByMC	<a href="http://forsyte.tuwien.ac.at/software/bymc/">http://forsyte.tuwien.ac.at/software/bymc/</a>
CHEAPS	<a href="http://lvk.cs.msu.ru/~konnov/cheaps/index_en.html">http://lvk.cs.msu.ru/~konnov/cheaps/index_en.html</a>
Cubicle	<a href="http://cubicle.lri.fr">http://cubicle.lri.fr</a>
FAST	<a href="http://tapas.labri.fr/trac/wiki/FASTER">http://tapas.labri.fr/trac/wiki/FASTER</a>
JTLV	<a href="http://jtlv.ysaar.net/">http://jtlv.ysaar.net/</a>
KeYmaera	<a href="http://symbolaris.com/info/KeYmaera.html">http://symbolaris.com/info/KeYmaera.html</a>
MCMT	<a href="http://users.mat.unimi.it/users/ghilardi/mcmt/">http://users.mat.unimi.it/users/ghilardi/mcmt/</a>
MONA	<a href="http://www.brics.dk/mona/">http://www.brics.dk/mona/</a>
Murφ	<a href="http://formalverification.cs.utah.edu/">http://formalverification.cs.utah.edu/</a>
NUXMV	<a href="https://nuxmv.fbk.eu">Murphi/https://nuxmv.fbk.eu</a>
PARTY	<a href="https://github.com/5nizza/Party">https://github.com/5nizza/Party</a>
Passel	<a href="https://publish.illinois.edu/passel-tool/">https://publish.illinois.edu/passel-tool/</a>
PWC	<a href="http://www.ahmedrezine.com/tools/">http://www.ahmedrezine.com/tools/</a>
TLV	<a href="http://www.cs.nyu.edu/acsys/tlv/">http://www.cs.nyu.edu/acsys/tlv/</a>
T(O)RMC	<a href="http://www.montefiore.ulg.ac.be/~legay/TORMC/index-tormc.html">http://www.montefiore.ulg.ac.be/~legay/TORMC/index-tormc.html</a>
Undip	<a href="http://www.ahmedrezine.com/tools/">http://www.ahmedrezine.com/tools/</a>



## CHAPTER 10

# Conclusions

In this book, we provided a general model for uniform concurrent systems that captures a large class of systems from the literature. Our model includes different forms of communication, like token-passing, rendezvous, or broadcast, as well as different communication graphs, like cliques, rings, stars, or even dynamic topologies that change at runtime.

For this class of systems, we surveyed the existing decidability results, drawing a map that focuses on the border between decidability and undecidability, and highlighting some of the uncharted territory in this research area.

Regarding undecidability results, we noticed that unrestricted forms of communication make it possible even for uniform parameterized systems to simulate Turing machines. Such communication is thus a source of undecidability. This even holds for quite simple systems, e.g., with a simple token in a bidirectional ring architecture, or a two-valued token in a unidirectional ring architecture. The reason is essentially that arbitrary amounts of information can be exchanged over an infinite run of a system, even if a single synchronization can only transmit one bit of information. Typically, this allows to represent Turing machines with tapes of arbitrary length, which leads to undecidability.

Many (but not all) decidability results give a constructive way to solve the PMCP for interesting subclasses, e.g., by a decomposition into several finite-state model checking problems. However, in many cases there either is no constructive result, or the decomposition is too big to be model checked in practice. Thus, additional research is needed to obtain effective ways to solve the PMCP.

Regarding specifications, it is worth noting that the token passing systems of Chapter 4 are the only ones known to us for which branching-time logics, like certain fragments of indexed- $\text{CTL}^*\backslash X$ , are known to have decidable PMCP. This is in contrast to pairwise rendezvous and disjunctively guarded systems that are known to have undecidable PMCP for  $\text{CTL}^*$ , while for ad hoc networks and conjunctively guarded systems the problem does not seem to have been studied.

Another set of open problems concerns lower bounds and computational complexity of the PMCP. A few exact bounds for PMCP are known: rendezvous systems and 1-index  $\text{LTL}\backslash X$  specifications are EXPSPACE-complete (and PSPACE-complete if there is no controller) [Esparza, 2014, German and Sistla, 1992]; disjunctively guarded systems and 1-index  $\text{LTL}\backslash X$  specifications are PSPACE-complete [Aminof et al., 2014b]; and broadcast systems and safety specifications have “Ackermanian complexity” [Schmitz and Schnoebelen, 2013].

## 144 10. CONCLUSIONS

While the theoretical results presented give some insight in the problems of verifying large concurrent systems, it should be noted that the systems considered here are still quite limited compared to practical concurrent programs. In this book, we considered parallel compositions  $P \parallel \dots \parallel P$  of  $n$  copies of a fixed finite-state process  $P$  — that is,  $P$  is independent of  $n$ . As systems consist of copies of the same process, processes cannot use unique identifiers (IDs). IDs are, however, quite natural in many applications, and from a theoretical viewpoint they are quite powerful as they can be used to break symmetries. Similarly, the processes have a fixed state space independent of the system size. Consequently, processes cannot use counting arguments (e.g., a majority of processes are in certain states) that are useful to decide on coordinated actions.

Of course, we can think of more complicated systems. Consider parallel compositions  $P(1, n) \parallel \dots \parallel P(n, n)$  where  $P(i, n)$  is a finite-state process that has a unique ID  $i$  as well as “knowledge” of the number of processes  $n$  in the system; consequently, the local state space of the processes is parameterized. Such systems seem quite natural, and actually there is a sizeable amount of literature on such systems, e.g., in the area of distributed algorithms [[Attiya and Welch, 2004](#), [Lynch, 1996](#)]. This model is rarely addressed in the parameterized model checking literature, and as discussed above, in most cases the PMCP is undecidable. However, we believe that in this area theoretical work must also be done in order to design procedures that allow one to reason about such systems.

# Bibliography

- Parosh Aziz Abdulla. Regular model checking. *STTT*, 14(2):109–118, 2012. DOI: [10.1007/s10009-011-0216-8](https://doi.org/10.1007/s10009-011-0216-8). 131
- Parosh Aziz Abdulla, Kārlis Čerāns, Bengt Jonsson, and Yih-Kuen Tsay. General decidability theorems for infinite-state systems. In *LICS*, pages 313 –321, 1996. DOI: [10.1109/LICS.1996.561359](https://doi.org/10.1109/LICS.1996.561359). 25, 26, 99
- Parosh Aziz Abdulla, Ahmed Bouajjani, Bengt Jonsson, and Marcus Nilsson. Handling global conditions in parametrized system verification. In *CAV*, pages 134–145. Springer, 1999. DOI: [10.1007/3-540-48683-6\\_14](https://doi.org/10.1007/3-540-48683-6_14). 132
- Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, and Julien d’Orso. Regular model checking made simple and efficient. In *CONCUR*, pages 116–130, 2002. DOI: [10.1007/3-540-45694-5\\_94](https://doi.org/10.1007/3-540-45694-5_94). 132
- Parosh Aziz Abdulla, Giorgio Delzanno, Noomene Ben Henda, and Ahmed Rezine. Regular model checking without transducers (on efficient verification of parameterized systems). In *TACAS*, volume 4424 of *LNCS*, pages 721–736. Springer, 2007. DOI: [10.1007/978-3-540-71209-1\\_56](https://doi.org/10.1007/978-3-540-71209-1_56). 133
- Parosh Aziz Abdulla, Giorgio Delzanno, and Ahmed Rezine. Approximated parameterized verification of infinite-state processes with global conditions. *Formal Methods in System Design*, 34(2):126–156, 2009. DOI: [10.1007/s10703-008-0062-9](https://doi.org/10.1007/s10703-008-0062-9). 133, 139
- Parosh Aziz Abdulla, Yu-Fang Chen, Giorgio Delzanno, Frédéric Haziza, Chih-Duo Hong, and Ahmed Rezine. Constrained monotonic abstraction: A CEGAR for parameterized verification. In *CONCUR*, pages 86–101, 2010. DOI: [10.1007/978-3-642-15375-4\\_7](https://doi.org/10.1007/978-3-642-15375-4_7). 129
- Parosh Aziz Abdulla, Giorgio Delzanno, Othmane Rezine, Arnaud Sangnier, and Riccardo Traverso. On the verification of timed ad hoc networks. In *FORMATS*, volume 6919 of *LNCS*, pages 256–270. Springer, 2011. DOI: [10.1007/978-3-642-24310-3\\_18](https://doi.org/10.1007/978-3-642-24310-3_18). 128
- Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Othmane Rezine. Verification of directed acyclic ad hoc networks. In *FORTE*, volume 7892 of *LNCS*, pages 193–208. Springer, 2013a. DOI: [10.1007/978-3-642-38592-6\\_14](https://doi.org/10.1007/978-3-642-38592-6_14). 19, 103, 107, 115, 116, 117, 118, 119, 120, 122

## 146 BIBLIOGRAPHY

- Parosh Aziz Abdulla, Frédéric Haziza, and Lukáš Holík. All for the price of few. In *VMCAI*, volume 7737 of *LNCS*, pages 476–495. Springer, 2013b. [DOI: 10.1007/978-3-642-35873-9\\_28](https://doi.org/10.1007/978-3-642-35873-9_28). 134, 135
- Benjamin Aminof, Swen Jacobs, Ayrat Khalimov, and Sasha Rubin. Parameterized model checking of token-passing systems. In *VMCAI*, volume 8318 of *LNCS*, pages 262–281, January 2014a. [DOI: 10.1007/978-3-642-54013-4\\_15](https://doi.org/10.1007/978-3-642-54013-4_15). 31, 35, 39, 40, 41, 42, 45, 46, 47
- Benjamin Aminof, Tomer Kotek, Sasha Rubin, Francesco Spegni, and Helmut Veith. Parameterized model checking of rendezvous systems. In *CONCUR*, volume 8704, pages 109–124. Springer, 2014b. [DOI: 10.1007/978-3-662-44584-6\\_9](https://doi.org/10.1007/978-3-662-44584-6_9). 46, 63, 101, 143
- Benjamin Aminof, Sasha Rubin, Florian Zuleger, and Francesco Spegni. Liveness of parameterized timed networks. In *ICALP (Part II)*, volume 9135 of *LNCS*, pages 375–387, 2015. [DOI: 10.1007/978-3-662-47666-6\\_30](https://doi.org/10.1007/978-3-662-47666-6_30). 64
- Krysztof R. Apt and Dexter C. Kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, 15:307–309, 1986. [DOI: 10.1016/0020-0190\(86\)90071-2](https://doi.org/10.1016/0020-0190(86)90071-2). 2, 3
- Hagit Attiya and Jennifer Welch. *Distributed Computing*, 2nd ed. John Wiley & Sons, 2004. [DOI: 10.1002/0471478210](https://doi.org/10.1002/0471478210). 144
- Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008. 1, 2, 12, 14, 83, 97
- Ittai Balaban, Yi Fang, Amir Pnueli, and Lenore D. Zuck. IIV: an invisible invariant verifier. In *CAV*, pages 408–412, 2005.
- Valmir C. Barbosa and Eli Gafni. Concurrency in heavily loaded neighborhood-constrained systems. *ACM Trans. Program. Lang. Syst.*, 11(4):562–584, 1989. [DOI: 10.1145/69558.69560](https://doi.org/10.1145/69558.69560). 48
- Sébastien Bardin, Alain Finkel, Jérôme Leroux, and Laure Petrucci. FAST: acceleration from theory to practice. *STTT*, 10(5):401–424, 2008. [DOI: 10.1007/s10009-008-0064-3](https://doi.org/10.1007/s10009-008-0064-3). 137, 139
- Gerard Basler, Michele Mazzucchi, Thomas Wahl, and Daniel Kroening. Symbolic counter abstraction for concurrent software. In *CAV*, volume 5643 of *LNCS*, pages 64–78. Springer, 2009. [DOI: 10.1007/978-3-642-02658-4\\_9](https://doi.org/10.1007/978-3-642-02658-4_9). 131, 139
- Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili. Regular model checking. In *CAV*, pages 403–418, 2000. [DOI: 10.1007/10722167\\_31](https://doi.org/10.1007/10722167_31). 132

- Ahmed Bouajjani, Peter Habermehl, and Tomás Vojnar. Abstract regular model checking. In *CAV*, pages 372–386, 2004. DOI: [10.1007/978-3-540-27813-9\\_29](https://doi.org/10.1007/978-3-540-27813-9_29). 132
- Ahmed Bouajjani, Peter Habermehl, and Tomás Vojnar. Verification of parametric concurrent systems with prioritised FIFO resource management. *Formal Methods in System Design*, 32(2):129–172, 2008. DOI: [10.1007/s10703-008-0048-7](https://doi.org/10.1007/s10703-008-0048-7). 48, 137
- Michael C. Browne, Edmund M. Clarke, and Orna Grumberg. Characterizing finite kripke structures in propositional temporal logic. *Theor. Comput. Sci.*, 59:115–131, 1988. DOI: [10.1016/0304-3975\(88\)90098-9](https://doi.org/10.1016/0304-3975(88)90098-9). 29
- Michael C. Browne, Edmund M. Clarke, and Orna Grumberg. Reasoning about networks with many identical finite state processes. *Information and Computation*, 81(1):13–31, 1989. DOI: [10.1016/0890-5401\(89\)90026-6](https://doi.org/10.1016/0890-5401(89)90026-6). 14
- Muffy Calder and Alice Miller. Using spin to analyse the tree identification phase of the ieee 1394 high-performance serial bus (firewire) protocol. *Formal Aspects of Computing*, 14(3):247–266, 2003. DOI: [10.1007/s001650300004](https://doi.org/10.1007/s001650300004). 131
- Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Michelini, Sergio Mover, Marco Roveri, and Stefano Tonetta. The nuXmv symbolic model checker. In *CAV*, volume 8559 of *LNCS*, pages 334–342, 2014. DOI: [10.1007/978-3-319-08867-9\\_22](https://doi.org/10.1007/978-3-319-08867-9_22). 139
- K. M. Chandy and J. Misra. The drinking philosophers problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 6(4):632–646, 1984. DOI: [10.1145/1780.1804](https://doi.org/10.1145/1780.1804). 48
- Allan Cheng, Javier Esparza, and Jens Palsberg. Complexity results for 1-safe nets. In *FSTTCS*, volume 761 of *LNCS*, pages 326–337. Springer, 1993. DOI: [10.1007/3-540-57529-4\\_66](https://doi.org/10.1007/3-540-57529-4_66). 105
- Alessandro Cimatti, Edmund M. Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. NuSMV 2: An opensource tool for symbolic model checking. In *CAV*, volume 2404 of *LNCS*, pages 359–364, 2002. DOI: [10.1007/3-540-45657-0\\_29](https://doi.org/10.1007/3-540-45657-0_29). 130
- Edmund Clarke, Orna Grumberg, Hiromi Hiraishi, Somesh Jha, David Long, Kenneth McMillan, and Linda Ness. Verification of the futurebus+ cache coherence protocol. Technical report, DTIC Document, 1992. DOI: [10.1007/BF01383968](https://doi.org/10.1007/BF01383968). 1
- Edmund Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 1999. 1, 2, 13, 14, 129

## 148 BIBLIOGRAPHY

- Edmund Clarke, Muralidhar Talupur, Tayssir Touili, and Helmut Veith. Verification by network decomposition. In *CONCUR 2004*, volume 3170, pages 276–291, 2004. DOI: [10.1007/978-3-540-28644-8\\_18](https://doi.org/10.1007/978-3-540-28644-8_18). 2, 19, 25, 31, 40, 41, 42
- Edmund Clarke, Murali Talupur, and Helmut Veith. Proving ptolemy right: The environment abstraction framework for model checking concurrent systems. In *TACAS*, pages 33–47. Springer, 2008. DOI: [10.1007/978-3-540-78800-3\\_4](https://doi.org/10.1007/978-3-540-78800-3_4). 130
- Edmund M Clarke, Orna Grumberg, and Somesh Jha. Verifying parameterized networks using abstraction and regular languages. In *CONCUR*, pages 395–407. Springer, 1995. DOI: [10.1007/3-540-60218-6\\_30](https://doi.org/10.1007/3-540-60218-6_30). 136
- Edmund M. Clarke, Orna Grumberg, and Somesh Jha. Verifying parameterized networks. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 19(5):726–750, 1997. DOI: [10.1145/265943.265960](https://doi.org/10.1145/265943.265960). 136
- Sylvain Conchon, Amit Goel, Sava Krstic, Alain Mebsout, and Fatiha Zaïdi. Cubicle: A parallel smt-based model checker for parameterized systems - tool paper. In *CAV*, volume 7358 of *LNCS*, pages 718–724. Springer, 2012. DOI: [10.1007/978-3-642-31424-7\\_55](https://doi.org/10.1007/978-3-642-31424-7_55). 133, 139
- Giorgio Delzanno. Constraint-based verification of parameterized cache coherence protocols. *Formal Methods in System Design*, 23(3):257–301, 2003. DOI: [10.1023/A:1026276129010](https://doi.org/10.1023/A:1026276129010). 137
- Giorgio Delzanno and Riccardo Traverso. Decidability and complexity results for verification of asynchronous broadcast networks. In *LATA*, volume 7810 of *LNCS*, pages 238–249. Springer, 2013. DOI: [10.1007/978-3-642-37064-9\\_22](https://doi.org/10.1007/978-3-642-37064-9_22). 128
- Giorgio Delzanno, Jean-François Raskin, and Laurent Van Begin. Towards the automated verification of multithreaded Java programs. In *TACAS*, volume 2280 of *LNCS*, pages 173–187, 2002. DOI: [10.1007/3-540-46002-0\\_13](https://doi.org/10.1007/3-540-46002-0_13). 51
- Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. Parameterized verification of ad hoc networks. In *CONCUR*, volume 6269 of *LNCS*, pages 313–327, 2010. DOI: [10.1007/978-3-642-15375-4\\_22](https://doi.org/10.1007/978-3-642-15375-4_22). 12, 19, 103, 107, 108, 109, 110, 111, 113, 114, 115, 120, 122, 124, 127
- Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. On the power of cliques in the parameterized verification of ad hoc networks. In *FOSSACS*, volume 6604 of *LNCS*, pages 441–455. Springer, 2011. DOI: [10.1007/978-3-642-19805-2\\_30](https://doi.org/10.1007/978-3-642-19805-2_30). 19, 103, 107, 108, 109, 113, 114, 119, 120, 122, 127
- Giorgio Delzanno, Arnaud Sangnier, Riccardo Traverso, and Gianluigi Zavattaro. The cost of parameterized reachability in mobile ad hoc networks. *CoRR*, abs/1202.5850, 2012a. DOI: [10.1007/978-3-642-30793-5\\_15](https://doi.org/10.1007/978-3-642-30793-5_15). 126

- Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. Verification of ad hoc networks with node and communication failures. In *FORTE*, volume 7273 of *LNCS*, pages 235–250. Springer, 2012b. DOI: [10.1007/978-3-642-41036-9\\_11](https://doi.org/10.1007/978-3-642-41036-9_11). 19, 103, 105, 106, 107, 108, 124, 128
- Giorgio Delzanno, Arnaud Sangnier, and Riccardo Traverso. Parameterized verification of broadcast networks of register automata. In *RP*, volume 8169 of *LNCS*, pages 109–121. Springer, 2013. DOI: [10.1007/978-3-540-61474-5\\_86](https://doi.org/10.1007/978-3-540-61474-5_86). 128
- David L Dill. The mur  $\phi$  verification system. In *CAV*, pages 390–393. Springer, 1996. DOI: [10.1007/978-3-540-69850-0\\_5](https://doi.org/10.1007/978-3-540-69850-0_5). 130
- David L. Dill. A retrospective on murphi. In *25 Years of Model Checking - History, Achievements, Perspectives*, volume 5000 of *LNCS*, pages 77–88. Springer, 2008. DOI: [10.1002/jgt.3190140406](https://doi.org/10.1002/jgt.3190140406). 130
- Guoli Ding. Subgraphs and well-quasi-ordering. *Journal of Graph Theory*, 16(5):489–502, 1992. DOI: [10.1002/jgt.3190160509](https://doi.org/10.1002/jgt.3190160509). 119
- Jacob Elgaard, Nils Klarlund, and Anders Møller. MONA 1.x: new techniques for WS1S and WS2S. In *CAV*, volume 1427 of *LNCS*, pages 516–520. Springer, 1998. DOI: [10.1007/BFb0028773](https://doi.org/10.1007/BFb0028773). 132, 139
- E. Allen Emerson and Edmund M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *ICALP*, volume 85 of *LNCS*, pages 169–181. Springer, 1980. DOI: [10.1007/3-540-10003-2\\_69](https://doi.org/10.1007/3-540-10003-2_69). 1
- E. Allen Emerson and Vineet Kahlon. Reducing model checking of the many to the few. In *CADE*, volume 1831 of *LNCS*, pages 236–254. Springer Berlin Heidelberg, 2000. DOI: [10.1007/10721959\\_19](https://doi.org/10.1007/10721959_19). 19, 29, 65, 73, 74, 81, 82, 84, 86, 89, 95, 100, 101, 138
- E. Allen Emerson and Vineet Kahlon. Model checking large-scale and parameterized resource allocation systems. In *TACAS*, volume 2280 of *LNCS*, pages 251–265, 2002. DOI: [10.1007/3-540-46002-0\\_18](https://doi.org/10.1007/3-540-46002-0_18). 48
- E. Allen Emerson and Vineet Kahlon. Exact and efficient verification of parameterized cache coherence protocols. In *CHARME*, volume 2860 of *LNCS*, pages 247–262. Springer, 2003a. DOI: [10.1007/978-3-540-39724-3\\_22](https://doi.org/10.1007/978-3-540-39724-3_22). 19, 65, 74, 99, 101
- E. Allen Emerson and Vineet Kahlon. Model checking guarded protocols. In *LICS*, pages 361–370. IEEE, 2003b. DOI: [10.1109/LICS.2003.1210076](https://doi.org/10.1109/LICS.2003.1210076). 18, 19, 23, 51, 58, 59, 62, 73, 74, 76, 79, 81, 95, 96, 97, 98, 99, 100

## 150 BIBLIOGRAPHY

- E. Allen Emerson and Vineet Kahlon. Rapid parameterized model checking of snoopy cache coherence protocols. In *TACAS*, volume 2619 of *LNCS*, pages 144–159. Springer, 2003c. DOI: [10.1007/3-540-36577-X\\_11](https://doi.org/10.1007/3-540-36577-X_11). 19, 65, 74, 99, 101
- E. Allen Emerson and Vineet Kahlon. Parameterized model checking of ring-based message passing systems. In *CSL*, volume 3210 of *LNCS*, pages 325–339. Springer, 2004. DOI: [10.1007/978-3-540-30124-0\\_26](https://doi.org/10.1007/978-3-540-30124-0_26). 19, 44, 45, 48
- E. Allen Emerson and Kedar S. Namjoshi. Reasoning about rings. In *POPL*, pages 85–94, 1995. DOI: [10.1145/199448.199468](https://doi.org/10.1145/199448.199468). 2, 12, 14, 16, 19, 23, 24, 25, 29, 31, 35, 36, 38, 39, 40, 42, 84
- E. Allen Emerson and Kedar S. Namjoshi. Automatic verification of parameterized synchronous systems. In *CAV*, volume 1102 of *LNCS*, pages 87–98. Springer, 1996. DOI: [10.1007/3-540-61474-5\\_60](https://doi.org/10.1007/3-540-61474-5_60). 18, 73, 74, 77
- E. Allen Emerson and Kedar S. Namjoshi. On model checking for non-deterministic infinite-state systems. In *LICS*, pages 70–80. IEEE Computer Society, 1998. DOI: [10.1109/LICS.1998.705644](https://doi.org/10.1109/LICS.1998.705644). 18, 19, 100
- E. Allen Emerson and Kedar S. Namjoshi. On reasoning about rings. *Int. J. Found. Comput. Sci.*, 14(4):527–550, 2003. DOI: [10.1142/S0129054103001881](https://doi.org/10.1142/S0129054103001881). 12, 14, 16, 19, 24, 25, 29, 31, 35, 36, 38, 39, 40, 42, 43
- E. Allen Emerson and A. Prasad Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9:105–131, 1996. DOI: [10.1007/BF00625970](https://doi.org/10.1007/BF00625970). 25
- Javier Esparza. Decidability and complexity of petri net problems - an introduction. In *In Lectures on Petri Nets I: Basic Models*, pages 374–428. Springer-Verlag, 1998. DOI: [10.1007/3-540-65306-6\\_20](https://doi.org/10.1007/3-540-65306-6_20). 25, 27
- Javier Esparza. Keeping a crowd safe: On the complexity of parameterized verification. In *STACS*, 2014. DOI: [10.4230/LIPIcs.STACS.2014.1](https://doi.org/10.4230/LIPIcs.STACS.2014.1). 63, 64, 143
- Javier Esparza and Mogens Nielsen. Decidability issues for petri nets - a survey. *Bulletin of the EATCS*, 52:244–262, 1994. 95
- Javier Esparza, Alain Finkel, and Richard Mayr. On the verification of broadcast protocols. *LICS*, page 352, 1999. ISSN 1043-6871. DOI: [10.1109/LICS.1999.782630](https://doi.org/10.1109/LICS.1999.782630). 10, 19, 51, 58, 59, 99, 100, 108, 114
- Javier Esparza, Pierre Ganty, and Rupak Majumdar. Parameterized verification of asynchronous shared-memory systems. In *CAV*, pages 124–140, 2013. DOI: [10.1007/978-3-642-39799-8\\_8](https://doi.org/10.1007/978-3-642-39799-8_8). 137

- Johannes Faber, Carsten Ihleman, Swen Jacobs, and Viorica Sofronie-Stokkermans. Automatic verification of parametric specifications with complex topologies. In *IFM*, volume 6396 of *LNCS*, pages 152–167. Springer, 2010. DOI: [10.1007/978-3-642-16265-7\\_12](https://doi.org/10.1007/978-3-642-16265-7_12). 134
- Yi Fang, Kenneth L. McMillan, Amir Pnueli, and Lenore D. Zuck. Liveness by invisible invariants. In *FORTE*, pages 356–371, 2006. DOI: [10.1007/11888116\\_26](https://doi.org/10.1007/11888116_26). 137
- Azadeh Farzan, Zachary Kincaid, and Andreas Podelski. Proofs that count. In *POPL*, pages 151–164, 2014. DOI: [10.1145/2535838.2535885](https://doi.org/10.1145/2535838.2535885). 138
- Azadeh Farzan, Matthias Heizmann, Jochen Hoenicke, Zachary Kincaid, and Andreas Podelski. Automated program verification. In *LATA*, pages 25–46, 2015a. DOI: [10.1007/978-3-319-15579-1\\_2](https://doi.org/10.1007/978-3-319-15579-1_2). 138
- Azadeh Farzan, Zachary Kincaid, and Andreas Podelski. Proof spaces for unbounded parallelism. In *POPL*, pages 407–420. ACM, 2015b. DOI: [10.1145/2676726.2677012](https://doi.org/10.1145/2676726.2677012). 138
- Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, 2001. DOI: [10.1016/S0304-3975\(00\)00102-X](https://doi.org/10.1016/S0304-3975(00)00102-X). 25, 26, 122
- Matthias Függer and Josef Widder. Efficient checking of link-reversal-based concurrent systems. In *CONCUR*, volume 7454 of *LNCS*, pages 486–499, 2012. DOI: [10.1007/978-3-642-32940-1\\_34](https://doi.org/10.1007/978-3-642-32940-1_34). 48
- Pierre Ganty and Ahmed Rezine. Ordered counter-abstraction — refinable subword relations for parameterized verification. In *LATA*, volume 8370 of *LNCS*, pages 396–408. Springer, 2014. DOI: [10.1007/978-3-319-04921-2\\_32](https://doi.org/10.1007/978-3-319-04921-2_32). 133, 139
- Steven M. German and A. Prasad Sistla. Reasoning about systems with many processes. *J. ACM*, 39(3):675–735, 1992. ISSN 0004-5411. DOI: [10.1145/146637.146681](https://doi.org/10.1145/146637.146681). 12, 18, 19, 29, 51, 52, 54, 55, 58, 59, 64, 99, 143
- Silvio Ghilardi and Silvio Ranise. Backward reachability of array-based systems by SMT solving: Termination and invariant synthesis. *Logical Methods in Computer Science*, 6(4), 2010. DOI: [10.2168/LMCS-6\(4:10\)2010](https://doi.org/10.2168/LMCS-6(4:10)2010). 133, 139
- Silvio Ghilardi, Enrica Nicolini, Silvio Ranise, and Daniele Zucchelli. Towards SMT model checking of array-based systems. In *Automated Reasoning*, volume 5195 of *LNCS*, pages 67–82. Springer, 2008. DOI: [10.1007/978-3-540-71070-7\\_6](https://doi.org/10.1007/978-3-540-71070-7_6). 133
- Jim Gray. Notes on data base operating systems. In *Operating Systems, An Advanced Course*, volume 60 of *LNCS*, pages 393–481. Springer, 1978. DOI: [10.1007/3-540-08755-9\\_9](https://doi.org/10.1007/3-540-08755-9_9). 103

## 152 BIBLIOGRAPHY

- Orna Grumberg and Helmut Veith, editors. *25 Years of Model Checking – History, Achievements, Perspectives*, volume 5000 of *LNCS*, 2008. [1](#)
- Jesper G. Henriksen, Jakob L. Jensen, Michael E. Jørgensen, Nils Klarlund, Robert Paige, Theis Rauhe, and Anders Sandholm. Mona: Monadic second-order logic in practice. In *TACAS*, volume 1019 of *LNCS*, pages 89–110. Springer, 1995. DOI: [10.1007/3-540-60630-0\\_5](https://doi.org/10.1007/3-540-60630-0_5). [132](#)
- Graham Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society*, s3-2(1):326–336, 1952. DOI: [10.1112/plms/s3-2.1.326](https://doi.org/10.1112/plms/s3-2.1.326). [119](#)
- Gerard Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, 2003. [130](#)
- C. Norris Ip and David L. Dill. Verifying systems with replicated components in Murφ. *Formal Methods in System Design*, 14(3):273–310, 1999. DOI: [10.1023/A:1008723125149](https://doi.org/10.1023/A:1008723125149). [130](#)
- Swen Jacobs and Viorica Sofronie-Stokkermans. Applications of hierarchical reasoning in the verification of complex systems. *Electr. Notes Theor. Comput. Sci.*, 174(8):39–54, 2007. DOI: [10.1016/j.entcs.2006.11.038](https://doi.org/10.1016/j.entcs.2006.11.038). [134](#)
- Henrik Jensen and Nancy Lynch. A proof of Burns n-process mutual exclusion algorithm using abstraction. In *TACAS*, volume 1384 of *LNCS*, pages 409–423. Springer, 1998. DOI: [10.1007/BFb0054186](https://doi.org/10.1007/BFb0054186). [131](#)
- Annu John, Igor Konnov, Ulrich Schmid, Helmut Veith, and Josef Widder. Parameterized model checking of fault-tolerant distributed algorithms by abstraction. In *FMCAD*, pages 201–209, 2013. DOI: [10.1145/2484239.2484285](https://doi.org/10.1145/2484239.2484285). [3](#), [129](#), [130](#), [139](#)
- Taylor T. Johnson. *Uniform Verification of Safety for Parameterized Networks of Hybrid Automata*. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL 61801, 2013. [137](#), [139](#)
- Taylor T. Johnson and Sayan Mitra. A small model theorem for rectangular hybrid automata networks. In *FORTE*, pages 18–34, 2012. DOI: [10.1007/978-3-642-30793-5\\_2](https://doi.org/10.1007/978-3-642-30793-5_2). [137](#)
- Alexander Kaiser, Daniel Kroening, and Thomas Wahl. Dynamic cutoff detection in parameterized concurrent programs. In *CAV*, volume 6174 of *LNCS*, pages 654–659. Springer, 2010. DOI: [10.1007/978-3-642-14295-6\\_55](https://doi.org/10.1007/978-3-642-14295-6_55). [134](#)
- Holger Karl. *Protocols and architectures for wireless sensor networks*. Wiley, Hoboken, NJ, 2005. ISBN 978-0-470-09510-2. DOI: [10.1002/0470095121](https://doi.org/10.1002/0470095121). [103](#), [107](#)
- Yonit Kesten and Amir Pnueli. Control and data abstraction: The cornerstones of practical formal verification. *International Journal on Software Tools for Technology Transfer*, 2(4):328–342, 2000. DOI: [10.1007/s100090050040](https://doi.org/10.1007/s100090050040). [136](#)

- Yonit Kesten, Oded Maler, Monica Marcus, Amir Pnueli, and Elad Shahar. Symbolic model checking with rich assertional languages. In *CAV*, pages 424–435. Springer, 1997. DOI: [10.1007/3-540-63166-6\\_41](https://doi.org/10.1007/3-540-63166-6_41). 1, 132
- Ayrat Khalimov, Swen Jacobs, and Roderick Bloem. Towards efficient parameterized synthesis. In *VMCAI*, volume 7737 of *LNCS*, pages 108–127. Springer, 2013a. DOI: [10.1007/978-3-642-35873-9\\_9](https://doi.org/10.1007/978-3-642-35873-9_9). 38, 40
- Ayrat Khalimov, Swen Jacobs, and Roderick Bloem. PARTY parameterized synthesis of token rings. In *CAV*, volume 8044 of *LNCS*, pages 928–933. Springer, 2013b. DOI: [10.1007/978-3-642-39799-8\\_66](https://doi.org/10.1007/978-3-642-39799-8_66). 139
- Igor V. Konnov and Vladimir A. Zakharov. An invariant-based approach to the verification of asynchronous parameterized networks. *J. Symb. Comput.*, 45(11):1144–1162, 2010. DOI: [10.1016/j.jsc.2008.11.006](https://doi.org/10.1016/j.jsc.2008.11.006). 136, 139
- Panagiotis Kouvaros and Alessio Lomuscio. Automatic verification of parameterised multi-agent systems. In *AAMAS*, pages 861–868, 2013a. 64
- Panagiotis Kouvaros and Alessio Lomuscio. A cutoff technique for the verification of parameterised interpreted systems with parameterised environments. In *IJCAI*, 2013b. 64
- Panagiotis Kouvaros and Alessio Lomuscio. A counter abstraction technique for the verification of robot swarms. In *AAAI Conference on Artificial Intelligence*, pages 2081–2088, 2015. 64
- Robert P Kurshan and Ken McMillan. A structural induction theorem for processes. In *Proceedings of the eighth annual ACM Symposium on Principles of distributed computing*, pages 239–247. ACM, 1989. DOI: [10.1145/72981.72998](https://doi.org/10.1145/72981.72998). 135, 136
- Leslie Lamport. Checking a multithreaded algorithm with +CAL. In *Distributed Computing*, pages 151–163. Springer, 2006. DOI: [10.1007/11864219\\_11](https://doi.org/10.1007/11864219_11). 2
- Axel Legay. T(O)RMC: A tool for (omega)-regular model checking. In *CAV*, volume 5123 of *LNCS*, pages 548–551. Springer, 2008. DOI: [10.1007/978-3-540-70545-1\\_52](https://doi.org/10.1007/978-3-540-70545-1_52). 132, 139
- Jérôme Leroux and Grégoire Sutre. Flat counter automata almost everywhere! In *ATVA*, volume 3707 of *LNCS*, pages 489–503, 2005. DOI: [10.1007/11562948\\_36](https://doi.org/10.1007/11562948_36). 137
- David Lesens, Nicolas Halbwachs, and Pascal Raymond. Automatic verification of parameterized linear networks of processes. In *POPL*, pages 346–357. ACM, 1997. DOI: [10.1145/263699.263747](https://doi.org/10.1145/263699.263747). 136
- Nancy Lynch. *Distributed Algorithms*. Morgan Kaufman Publishers, Inc., San Francisco, USA, 1996. 14, 144

## 154 BIBLIOGRAPHY

- Monika Maidl. A unifying model checking approach for safety properties of parameterized systems. In *CAV*, pages 311–323. Springer, 2001. DOI: [10.1007/3-540-44585-4\\_29](https://doi.org/10.1007/3-540-44585-4_29). 132
- Kenneth L. McMillan and Lenore D. Zuck. Invisible invariants and abstract interpretation. In *SAS*, pages 249–262, 2011. DOI: [10.1007/978-3-642-23702-7\\_20](https://doi.org/10.1007/978-3-642-23702-7_20). 137
- Robin Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989. ISBN 978-0-13-115007-2. 3, 31
- Marvin L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967. ISBN 0-13-165563-9. 23, 24
- André Platzer. A complete axiomatization of quantified differential dynamic logic for distributed hybrid systems. *Logical Methods in Computer Science*, 8(4), 2012. DOI: [10.2168/LMCS-8\(4:17\)2012](https://doi.org/10.2168/LMCS-8(4:17)2012). 139
- André Platzer and Jan-David Quesel. Keymaera: A hybrid theorem prover for hybrid systems (system description). In *IJCAR*, volume 5195 of *LNCS*, pages 171–178. Springer, 2008. DOI: [10.1007/978-3-540-71070-7\\_15](https://doi.org/10.1007/978-3-540-71070-7_15). 139
- Amir Pnueli and Elad Shahar. A platform for combining deductive with algorithmic verification. In *CAV*, volume 1102 of *LNCS*, pages 184–195. Springer, 1996. DOI: [10.1007/3-540-61474-5\\_68](https://doi.org/10.1007/3-540-61474-5_68). 132, 139
- Amir Pnueli and Elad Shahar. Liveness and acceleration in parameterized verification. In *CAV*, pages 328–343, 2000. DOI: [10.1007/10722167\\_26](https://doi.org/10.1007/10722167_26). 132
- Amir Pnueli, Sitvanit Ruah, and Lenore Zuck. Automatic deductive verification with invisible invariants. In *TACAS*, volume 2031 of *LNCS*, pages 82–97. Springer, 2001. DOI: [10.1007/3-540-45319-9\\_7](https://doi.org/10.1007/3-540-45319-9_7). 137
- Amir Pnueli, Jessie Xu, and Lenore Zuck. Liveness with  $(0,1,\infty)$ - counter abstraction. In *CAV*, volume 2404 of *LNCS*, pages 93–111. Springer, 2002. DOI: [10.1007/3-540-45657-0\\_9](https://doi.org/10.1007/3-540-45657-0_9). 3, 129, 130
- Amir Pnueli, Yaniv Sa’ar, and Lenore D. Zuck. JTLV: A framework for developing verification algorithms. In *CAV*, volume 6174 of *LNCS*, pages 171–174. Springer, 2010. 132, 139
- Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in cesar. In *International Symposium on Programming*, volume 137 of *LNCS*, pages 337–351. Springer, 1982. DOI: [10.1007/3-540-11494-7\\_22](https://doi.org/10.1007/3-540-11494-7_22). 1
- Tatiana Rybina and Andrei Voronkov. BRAIN : Backward reachability analysis with integers. In *AMAST*, volume 2422 of *LNCS*, pages 489–494. Springer, 2002. DOI: [10.1007/978-3-642-14295-6\\_18](https://doi.org/10.1007/978-3-642-14295-6_18). 139

- Sylvain Schmitz and Philippe Schnoebelen. The power of well-structured systems. In *CONCUR*, volume 8052 of *LNCS*, pages 5–24. Springer, 2013. Invited paper. DOI: [10.1007/978-3-642-40184-8\\_2](https://doi.org/10.1007/978-3-642-40184-8_2). 63, 143
- Ze'ev Shtadler and Orna Grumberg. Network grammars, communication behaviors and automatic verification. In *Automatic Verification Methods for Finite State Systems*, pages 151–165. Springer, 1990. DOI: [10.1007/3-540-52148-8\\_13](https://doi.org/10.1007/3-540-52148-8_13). 136
- T.K. Srikanth and Sam Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2:80–94, 1987. DOI: [10.1007/BF01667080](https://doi.org/10.1007/BF01667080). 3, 14
- Ichiro Suzuki. Proving properties of a ring of finite-state machines. *Inf. Process. Lett.*, 28(4): 213–214, July 1988. DOI: [10.1016/0020-0190\(88\)90211-6](https://doi.org/10.1016/0020-0190(88)90211-6). 3, 23, 31, 42
- Murali Talupur and Mark R. Tuttle. Going with the flow: Parameterized verification using message flows. In *FMCAD*, pages 1–8. IEEE, 2008. DOI: [10.1109/FMCAD.2008.ECP.14](https://doi.org/10.1109/FMCAD.2008.ECP.14). 131
- Anthony Widjaja To and Leonid Libkin. Algorithmic metatheorems for decidable LTL model checking over infinite systems. In *Foundations of Software Science and Computational Structures*, pages 221–236. Springer, 2010. DOI: [10.1007/978-3-642-12032-9\\_16](https://doi.org/10.1007/978-3-642-12032-9_16). 132
- Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Banff Higher Order Workshop*, pages 238–266, 1995. DOI: [10.1007/3-540-60915-6\\_6](https://doi.org/10.1007/3-540-60915-6_6). 55
- Pierre Wolper and Vinciane Lovinfosse. Verifying properties of large sets of processes with network invariants. In *Automatic Verification Methods for Finite State Systems*, volume 407 of *LNCS*, pages 68–80, 1989. DOI: [10.1007/3-540-52148-8\\_6](https://doi.org/10.1007/3-540-52148-8_6). 3, 135, 136
- Tuba Yavuz-Kahveci and Tevfik Bultan. Action language verifier: an infinite-state model checker for reactive software specifications. *Formal Methods in System Design*, 35(3):325–367, 2009. DOI: [10.1007/s10703-009-0081-1](https://doi.org/10.1007/s10703-009-0081-1). 139
- Hsu-Chun Yen. A unified approach for deciding the existence of certain petri net paths. *Information and Computation*, 96(1):119 – 137, 1992. DOI: [10.1016/0890-5401\(92\)90059-O](https://doi.org/10.1016/0890-5401(92)90059-O). 25



# Authors' Biographies

## RODERICK BLOEM

**Roderick Bloem** is a professor at Graz University of Technology. He received an M.Sc. in computer science from Leiden University in the Netherlands (1996) and a Ph.D. from the University of Colorado at Boulder (2001). His thesis work, under the supervision of Fabio Somenzi, was on formal verification using Linear Temporal Logic.

Since 2002, he has been an assistant professor at Graz University of Technology and a full professor since 2008. His research interests are in formal methods for the design and verification of digital systems, including hardware, software, and combinations such as embedded systems. He studies applications of game theory to the automatic synthesis of systems from their specifications, connections between temporal logics and omega-automata, model checking, and automatic fault localization and repair.

## SWEN JACOBS

**Swen Jacobs** is a postdoc at Saarland University. He received his Ph.D. (Dr. Ing.) from Saarland University for his work on decision procedures for the verification of complex systems at the Max-Planck-Institute for Informatics.

He worked at École Polytechnique Fédérale de Lausanne (EPFL), at Technical University Graz, and has been a visiting professor at the University of Ljubljana. His current work focuses on the automated verification and synthesis of distributed systems, based on a combination of logical and game-theoretic methods.

## AYRAT KHALIMOV

**Ayrat Khalimov** is a Ph.D. student at Technical University of Graz, Austria. He received his master's degree in applied physics and Mathematics at Moscow Institute of Physics and Technology (MIPT), with the thesis focusing on a method of calculation of current leakages in hardware circuits. Later, he joined Dependable Systems Lab at École Polytechnique Fédérale de Lausanne (EPFL) for an internship where he researched symbolic execution techniques for software verification. His current area of research is parameterized synthesis and verification.

## 158 AUTHORS' BIOGRAPHIES

### IGOR KONNOV

**Igor Konnov** is a postdoc (Universitätsassistent) at the Formal Methods in Systems Engineering Group, Institute of Information Systems of TU Wien (Vienna University of Technology). His research interests include model checking, parameterized model checking, and verification of distributed algorithms.

He received his Specialist (comparable to M.Sc.) and Ph.D. degrees in applied mathematics and computer science from Lomonosov Moscow State University. In his Ph.D. thesis, he introduced new techniques for parameterized model checking.

### SASHA RUBIN

**Sasha Rubin** is a postdoc at the Università degli Studi di Napoli “Federico II” (University of Naples). He is broadly interested in the connections between automata theory and logic, and in particular, in formal verification, game theory, and finite model theory. He received his Ph.D. from the University of Auckland and was awarded the Vice-chancellor’s prize for the best doctoral thesis in the Faculty of Science. He previously held a New Zealand Science and Technology Postdoctoral Fellowship. He is currently a Marie Curie fellow of the Istituto Nazionale di Alta Matematica.

### HELMUT VEITH

**Helmut Veith** is a professor at the Faculty of Informatics of TU Wien (Vienna University of Technology), and an adjunct professor at Carnegie Mellon University. He has a diploma in Computational Logic and a Ph.D. sub auspiciis praesidentis in Computer Science, both from TU Wien. Prior to his appointment to Vienna, he held professor positions at TU Darmstadt and TU Munich.

In his research, Helmut Veith applies formal and logical methods to problems in software technology and engineering. His current work focuses on model checking, software verification and testing, embedded software, and computer security.

### JOSEF WIDDER

**Josef Widder** is an assistant professor (Privatdozent) at the Faculty of Informatics of TU Wien. His primary area of interest is the theoretical approach to distributed algorithms, currently focusing on automated verification of fault-tolerant distributed algorithms.

He received his Ph.D. (Doctor technicae), and his habilitation in computer science from TU Wien. He worked at the Embedded Computing Systems group and the Formal Methods in Systems Engineering group of TU Wien (Austria), the Laboratoire d’Informatique de l’Ecole polytechnique (France), and the Parasol Lab at Texas A&M University (USA).

## **4.5 Publication**

The rest of this page is intentionally left blank

# Parameterised Verification of Autonomous Mobile-Agents in Static but Unknown Environments

Sasha Rubin \*

Università degli Studi di Napoli "Federico II"

Naples, Italy

sasha.rubin@gmail.com

## ABSTRACT

Automata walking on graphs are a mathematical formalisation of autonomous mobile agents with limited memory operating in discrete environments. This paper establishes a framework in which to model and automatically verify that autonomous mobile agents correctly perform their tasks. The framework consists of a logical language tailored for expressing agent tasks, and an algorithm solving the *parameterised* verification problem, where the graphs are treated as the parameter. We reduce the parameterised verification problem to classic questions in automata theory and monadic second order logic, i.e., universality and validity problems.

We illustrate the framework by instantiating it to a popular model of robot from the distributed computing literature.

This work clarifies the border between classes of mobile-agent systems that have decidable parameterised verification problem, and those that do not.

## Categories and Subject Descriptors

D.2.4 [Software/Program Verification]: Formal Methods; F.1.1 [Computation by Abstract Devices]: Models of Computation—*Automata*; C.2.2 [Computer Communication Networks]: Network Protocols—*Protocol Verification*; C.2.4 [Computer Communication Networks]: Distributed Systems; I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence—*Multiagent systems*

## Keywords

Model Checking; Logic; Automata Theory; Distributed Networks; Autonomous Mobile Agents; Robots; Parameterised Verification

## 1. INTRODUCTION

Autonomous mobile agents are designed to achieve some task in an environment, e.g., exploration, or rendezvous. They are in an *unknown* environment if they do not have

\*Marie Curie fellow of the Istituto Nazionale di Alta Matematica.

Appears in: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015)*, Bordini, Elkind, Weiss, Yolum (eds.), May 4–8, 2015, Istanbul, Turkey.

Copyright © 2015, International Foundation for Autonomous Agents and Multiagent Systems ([www.ifaamas.org](http://www.ifaamas.org)). All rights reserved.

global information about the environment, e.g., mobile software exploring hostile computer networks, or physical robots that rendezvous in an environment not reachable by humans. This paper studies a mathematical formalisation of autonomous mobile agents in discrete environments, i.e., robots on finite graphs.

The distributed computing community has recently proposed and studied a number of models of robot systems [5, 33, 28, 18, 21, 16, 27]. Theorems from this literature are *parameterised* i.e., they may involve graph-parameters (e.g., the maximum degree, the number of vertices, the diameter), memory parameters (e.g., the number of internal states of the robot protocol), and the number of robots may be a parameter.

However, until recently [31, 4, 35, 32] there has been little emphasis on formal analysis of correctness of robots in a parameterised setting. In this paper we apply formal methods to the parameterised verification problem in which it is the *environment that is parameterised*. This is how we model that robots operate in unknown environments. Formally, we address the following decision problem.

**Parameterised Verification Problem:** Given robots  $\bar{R}$ , decide if they solve the task T on all graphs  $G \in \mathcal{G}$ .

In contrast, the classic (non-parameterised) verification problem states:

**Verification Problem:** Given robots  $\bar{R}$  and a graph  $G \in \mathcal{G}$ , decide if robots  $\bar{R}$  solve the task T on G.

The non-parameterised verification problem is often handled using model-checking [14]. The parameterised verification problem corresponds to a family of model-checking problems (one for each  $G \in \mathcal{G}$ ), or equivalently, a single infinite-state model-checking problem. A variety of techniques have been applied to such systems, fully discussed in Section 7 (Related Work), however none seem suited to the problem in which the environment is parameterised. Fortunately, the theory of automata and logic has produced a rich understanding of the expressive power of certain logics, e.g., monadic second-order logic MSOL, over certain classes of graphs, e.g., context-free sets of graphs [44, 45, 29, 17]. Our framework reduces the parameterised verification problem to classic questions in automata theory and MSOL, i.e., universality and validity problems. The key observations are that: i) robots are graph-walking automata, and can be defined in MSOL, and ii) robot tasks correspond to properties of runs of automata which are often definable in MSOL.

### Modeling Choices.

There are a number of different modeling choices for mo-

bile agents [38]. We list them, and emphasise (using underlining) the choices made in this paper: (i) is the environment continuous (e.g., the plane) or discrete (e.g. a finite graph whose edges are labeled by directions or port numbers)? (ii) do agents act synchronously or asynchronously? (iii) are agents probabilistic or non-deterministic? (iv) is there one agent or are there multiple agents, and are the number of agents known or unknown? (v) is the environment static, or can agents affect their environment (e.g., by marking the nodes of a graph)? Moreover, (vi) how much information about the environment is known, a priori, to the agents? We assume agents do not have global knowledge of the environment, and in particular the size is not known and nodes in the environment do not have unique identifiers. And finally, (vii) how do agents communicate and sense their environment? We assume agents can *sense their positions in the graph* as well as those of the other agents (in the case of multiple agents), i.e., robots acquire information about the current state of the environment solely by vision (we use logical formulas, which we call tests, to define these sensing abilities).

### Aim and Contributions.

The main aim of this work is to provide a clearer understanding of the border between classes of autonomous mobile-agent systems that have decidable parameterised verification problem, and those that do not.

In particular, we provide a quite general formalisation of robot systems (using the modeling choices above) in which all components (i.e., environments, robots, and tasks) are modeled separately. We formalise reasoning about robot systems as the parameterised verification problem (PVP) where the environment is treated as a parameter. We show how to reduce the PVP to problems about automata and logic, i.e., universality and validity problems. We then prove that the PVP is decidable for suitable restrictions of the components, notably the case of a single robot on context-free sets of graphs. We illustrate the power of the framework by instantiating it to a model of robot systems from the distributed computing literature. This instantiation falls into our decidable restrictions.

## 2. BACKGROUND: AUTOMATA THEORY

In order to make this paper self-contained, this section contains the necessary background. In particular, we will use basic notions from mathematical logic and automata theory to formalise the robot systems (a full logical background can be found in [22], and a smaller discussion of monadic second-order logic **MSOL** over graphs can be found in [17, Section 1.3.1]), and easy-to-state theorems (e.g., Kleene's Theorem about the equivalence of regular expressions and automata, and Courcelle's Theorem on satisfiability of **MSOL** over context-free sets of graphs [17]) to give an algorithm for solving the parameterised verification problem.

Write  $B^\omega$  for the set of infinite sequences over alphabet  $B$ , and write  $B^*$  for the set of finite sequences. The empty sequence is denoted  $\epsilon$ . Write  $[n]$  for the set  $\{1, 2, \dots, n\}$ .

### Graphs.

A  $\Sigma$ -graph, or *graph*,  $G$ , is a tuple  $(V, E, \Sigma, \lambda)$  where  $V$  is a finite set of vertices,  $E \subseteq V \times V$  is a relation called the *edge relation*,  $\Sigma$  is a finite set of *edge labels*, and  $\lambda : E \rightarrow \Sigma$  is the

*edge labeling function*. The *out-degree* of a vertex  $v$ , written  $\deg(v)$ , is the cardinality of the set  $\{(v, w) \in E : w \in V\}$  of outgoing edges of  $v$ .

Sometimes the edge labels give a sense of direction:

EXAMPLE 1. An  $L$ -labeled grid is a graph with  $V = [n] \times [m]$ ,  $\Sigma = L \times \{N, S, E, W\}$ , and labels  $\lambda((x, y), (x+1, y)) \in L \times \{E\}$ , etc., where  $L$  is a finite set and  $n, m \in \mathbb{N}$ . An  $L$ -labeled line is an  $L$ -labeled grid with  $V = [n] \times [1]$ . If  $|L| = 1$  then we call the grid (or line) unlabeled.

EXAMPLE 2. A  $\Delta$ -ary tree (for  $\Delta \in \mathbb{N}$ ) is a  $\Sigma$ -graph  $(V, E, \Sigma, \lambda)$  where  $(V, E)$  is a tree,  $\Sigma = [\Delta] \cup \{up\}$ , and  $\lambda$  labels the edge leading to the node in direction  $i$  (if it exists) by  $i$ , and the edge leading to the parent of a node (other than the root) is labelled by up. We may rename the labels for convenience, e.g., for binary trees ( $\Delta = 2$ ) we let  $\Sigma = \{lc, rc, up\}$  where  $lc$  replaces 1 and  $rc$  replaces 2.

### First-order and Monadic Second-order Logic.

Formulas will be interpreted in  $\Sigma$ -graphs  $G$  (an exception are formulas in Section 4.3). Define the set of monadic second-order formulas **MSOL**( $\Sigma$ ) as follows. Formulas of **MSOL**( $\Sigma$ ) are built using *first-order variables*  $x, y, \dots$  that vary over vertices, and *set variables*  $X, Y, \dots$  that vary over sets of vertices. The *atomic formulas* (when interpreted over  $\Sigma$ -graphs) are:  $x = y$  (denoting that vertex  $x$  is the same as vertex  $y$ ),  $x \in X$  (denoting that vertex  $x$  is in the set of vertices  $X$ ), and  $edg_\sigma(x, y)$  (denoting that there is an edge from  $x$  to  $y$  labeled  $\sigma \in \Sigma$ ). The formulas of **MSOL**( $\Sigma$ ) are built from the atomic formulas using the Boolean connectives (i.e.,  $\neg, \vee, \wedge, \rightarrow$ ) and variable quantification (i.e.,  $\forall, \exists$  over both types of variables). The fragment of **MSOL**( $\Sigma$ ) which does not mention set variables is called *first-order logic*, denoted **FOL**( $\Sigma$ ). Write **MSOL** $_k$ ( $\Sigma$ ) for formulas with  $k$ -many free variables.

Here are some examples of formulas and their meanings:

- MF1. The formula  $\forall x(x \in X \rightarrow x \in Y)$  means that  $X \subseteq Y$ . Similarly, there are formulas for the set operations  $\cup, \cap, =$ , and relative complement  $X \setminus Y$ .
- MF2. The formula  $edg(x, y) := \bigvee_{\sigma \in \Sigma} edg_\sigma(x, y)$  means that there is an edge from  $x$  to  $y$  (here  $\Sigma$  is assumed to be finite).
- MF3. The formula  $\exists x \exists y(x \neq y \wedge edg(z, x) \wedge edg(z, y))$  means that  $\deg(z) \geq 2$ .
- MF4. If  $\phi(x, y)$  is an **MSOL**( $\Sigma$ ) formula then define  $\phi^*(x, y)$  by  $\forall Z[(closed_\phi(Z) \wedge x \in Z) \rightarrow y \in Z]$ . Here  $closed_\phi(Z)$  is defined as  $\forall a \forall b[(a \in Z \wedge \phi(a, b)) \rightarrow b \in Z]$ . Note that  $\phi^*$  is an **MSOL**( $\Sigma$ )-formula.

Note that  $\phi^*(x, y)$  holds in a graph  $G$  if and only if there exists a finite sequence of vertices  $v_1 v_2 \dots v_m \in V^*$  (here  $m \geq 1$ ) such that  $x = v_1, y = v_m$  and  $\phi(v_i, v_{i+1})$  holds in  $G$  for all  $i < m$ . This shows that if a binary relation is **MSOL**-definable, then so is its transitive-closure.

Similarly, define  $\phi^\omega(x, y) := \phi^*(x, y) \wedge \exists z(\phi(y, z) \wedge \phi^*(z, y))$ . Note  $\phi^\omega(x, y)$  holds in a graph  $G$  if and only if there is an infinite sequence of vertices  $v_1 v_2 \dots \in V^\omega$  with  $x = v_1, v_i = y$  for infinitely many  $i \in \mathbb{N}$ , and  $\phi(v_i, v_{i+1})$  holds in  $G$  for all  $i \in \mathbb{N}$ . This uses the fact that  $V$  is finite.

MF5. The *k*-ary transitive-closure operator is the function that maps a *2k*-ary relation  $\rho(\bar{x}, \bar{y})$  to the *2k*-ary relation  $\rho^*(\bar{x}, \bar{y})$  such that, in every graph  $G$ :  $\rho^*(\bar{x}, \bar{y})$  holds if and only if there exists a sequence  $\bar{v}_1, \bar{v}_2, \dots, \bar{v}_m$  such that  $\bar{x} = \bar{v}_1$ ,  $\bar{y} = \bar{v}_m$ , and  $\rho(\bar{v}_i, \bar{v}_{i+1})$  holds for every  $i < m$ . For  $k > 1$ , it is not the case that if a *k*-ary relation  $\phi$  is MSOL-definable, then so is its *k*-ary transitive-closure, because, intuitively, this would require having *k*-ary relation variables  $Z$ .<sup>1</sup>

For a formula  $\phi(x_1, \dots, x_k)$ , a graph  $G$ , and  $v_1, \dots, v_k \in V$ , write  $G \models \phi(v_1, \dots, v_k)$  to mean that  $\phi$  holds in  $G$  with variable  $x_i$  substituted by vertex  $v_i$  (for  $i \leq k$ ).

### The Validity Problem and Courcelle's Theorem.

A *sentence* is a formula with no free variables. Let  $\Phi$  be a set of sentences, and let  $\mathcal{G}$  be a set of graphs. The  *$\Phi$ -validity problem* of  $\mathcal{G}$  is to decide, given  $\phi \in \Phi$ , whether or not for all graphs  $G \in \mathcal{G}$ , it holds that  $G \models \phi$ . Unfortunately for us, the  $\text{FOL}(\Sigma)$ -validity problem for  $\mathcal{G}$  = the set of all  $\Sigma$ -graphs is undecidable [22]. On the other hand, many interesting cases have decidable validity. One version of Courcelle's Theorem states that the MSOL-validity problem is decidable for every context-free set of graphs  $\mathcal{G}$  [17].<sup>2</sup> Context-free sets of graphs are the analogue of context-free sets of strings, and are generated by graph grammars or equations using certain graph operations. Examples include, for a fixed alphabet, the set of labeled lines, the set of rings, the set of trees, the set of series-parallel graphs, the set of cliques, but not the set of all grids.

### Automata and Regular Expressions.

Ordinary *regular-expressions* over a finite alphabet  $B$  are built from the sets  $\emptyset$ ,  $\{\epsilon\}$ , and  $\{b\}$  for  $b \in B$ , and the operations union  $+$ , concatenation  $\cdot$ , and Kleene-star  $^*$ . Kleene's Theorem states that the languages recognised by regular expressions over alphabet  $B$  are exactly those recognised by finite automata over alphabet  $B$ .

An  $\omega$ -*regular expression* over the finite alphabet  $B$  is inductively defined to be of the form:  $\exp^\omega$ ,  $\exp \cdot r$ , or  $r + r'$ , where  $\exp$  is an ordinary regular-expression over  $B$ , and  $r, r'$  are  $\omega$ -regular expressions over  $B$ . An  $\omega$ -regular language is one defined by an  $\omega$ -regular expression. A variation of Kleene's Theorem says that the languages recognised by  $\omega$ -regular expressions over alphabet  $B$  are exactly the languages recognised by Büchi automata over alphabet  $B$  (which are like non-deterministic finite automata except that they take infinite words as input, and accept if some accepting state occurs infinitely often).

## 3. THE MODEL OF ROBOT SYSTEMS

We provide a framework for modeling multi-robot systems parameterised by their environment.

<sup>1</sup>To prove this formally, note that 2-ary transitive closure on finite words (i.e., binary-labeled lines) can define the non-regular language  $\{0^n 1^n : n \in \mathbb{N}\}$ ; now use the fact that, over finite words, MSOL can only define the regular languages, part of a result known as the Büchi-Elgot-Trakhtenbrot Theorem, see [45].

<sup>2</sup>Actually, a strong form of the theorem states that there is an algorithm that given a description of a context-free set of graphs  $\mathcal{G}$  and an MSOL-formula  $\phi$  decides if every graph in  $\mathcal{G}$  satisfies  $\phi$ .

### 3.1 The model of robot systems

In this section we define robot systems, i.e., robot protocols, environments, and tasks.

Environments are modeled as  $\Sigma$ -graphs,<sup>3</sup> and robots are modeled as regular languages of instructions, as in [7]. An instruction either tells the robot to move along an edge, or it allows the robot to test the positions of the robots (e.g., a robot can learn which other robots are at the same vertex as it is, or if there is a robot north of it). Tests are formalised as logical formulas. As discussed in Section 5, our model is general enough to be able to express a popular model of robot systems found in the distributed computing literature [27, 21, 16, 33, 28, 18].

We first define robot systems consisting of a single robot, and then define multi-robot systems.

#### Robot Instruction Set.

A robot follows instructions from the *instruction set*

$$\text{INS}_{\Sigma,1} := \{\uparrow_\sigma : \sigma \in \Sigma\} \cup \text{MSOL}_1(\Sigma).$$

Instructions are of two kinds, *moves* and *tests*:<sup>4</sup>  $\uparrow_\sigma$  tells the robot to move from its current vertex along the edge labeled  $\sigma$ ; and  $\text{MSOL}_1(\Sigma)$  consists of formulas  $\tau(x)$  that allow the robot to test that  $\tau(x)$  holds in  $G$ , where  $x$  is the current vertex of the robot in  $G$ . Typical tests include: is the degree of the current vertex at least  $d$ ? is there an edge out of the current vertex labeled  $\sigma$ ? Here is a more complex test: is there a path from the current vertex that only uses edges labeled  $\sigma$  to a vertex of degree  $d$ ?

For a sequence  $\text{ins} \in (\text{INS}_{\Sigma,1})^*$ , write  $[[\text{ins}]]_G \subseteq V^2$  for the set of pairs of vertices  $(u, v)$  such that, in  $G$ , one can reach  $v$  from  $u$  by following the instructions  $\text{ins}$ . Formally,

1.  $(u, v) \in [[\epsilon]]_G$  iff  $u = v$ ,
2.  $(u, v) \in [[\uparrow_\sigma]]_G$  iff  $\lambda(u, v) = \sigma$ ,
3.  $(u, v) \in [[\tau(x)]]_G$  iff  $u = v$  and  $G \models \tau(u)$ , and
4.  $(u, v) \in [[d \cdot e]]_G$  for  $d \in (\text{INS}_{\Sigma,1})^*$  and  $e \in \text{INS}_{\Sigma,1}$  iff there exists  $z \in V$  such that  $(u, z) \in [[d]]_G$  and  $(z, v) \in [[e]]_G$ .

#### Robot Protocols.

Fix a set of edge-labels  $\Sigma$ . A *1-robot*, or *robot*,  $R$ , is a pair  $(Q, \delta)$  where  $Q$  is a finite set of *states*, and  $\delta \subset Q \times \text{INS}_{\Sigma,1} \times Q$  is a finite *transition* relation.

We sometimes designate certain states of the robot to be *initial*, denoted  $I \subseteq Q$  and certain states to be *repeating*, denoted  $A \subseteq Q$ . A state  $p \in Q$  is called *halting* if the robot has the transition  $(p, \text{true}, p)$ , i.e., state  $p$  is a sink. Thus we model a halting robot as one that stays in the same state forever. The set of halting states is denoted  $H \subseteq Q$ .

EXAMPLE 3. The robot in Figure 1 does a (perpetual) depth-first search of trees in which every node has zero or two children.<sup>5</sup> Tests are written *leaf*, *lc*, *rc*, *root* (whose meanings are “is the current node a leaf?”, “left-child?”, “right-child?”, “the root?”), and move instructions are written  $\uparrow_{up}$ ,

<sup>3</sup>The fact that  $\Sigma$ -graphs are finite corresponds to our modeling assumption that environments are *discrete*.

<sup>4</sup>Looking at the instruction set we see that robots cannot alter the graph, i.e., the environment is *static*.

<sup>5</sup>To make the diagram readable, an edge may be labeled by multiple successive actions separated by semicolons.

$\uparrow_{lc}$ , and  $\uparrow_{rc}$  (whose meanings are “move up to the parent”, “to the left-child”, “to the right-child”).

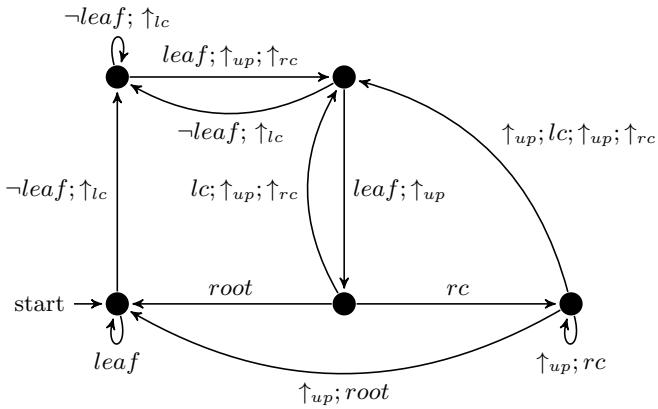


Figure 1: Robot that does a DFS on binary trees.

### *Configurations and Runs.*

A robot walks about a  $\Sigma$ -graph  $G$ . Let  $R = (Q, \delta)$  be a robot with instruction set  $\text{INS}_{\Sigma, 1}$ . A *configuration*  $c$  of  $R$  on  $G$  is a pair  $\langle v, q \rangle \in V \times Q$ . Say that the configuration  $\langle v, q \rangle$  has *position*  $v$ . A configuration is *initial* (resp. *halting, repeating*) if  $q$  is. The following definition expresses that one configuration results from another after the robot executes a single instruction: for configurations  $c = \langle v, p \rangle$  and  $d = \langle w, q \rangle$ , say that  $d$  *results from*  $c$  iff there is a transition  $(p, \text{ins}, q)$  of  $\delta$  such that  $(v, w) \in [[\text{ins}]]_G$ . A *run*  $\alpha$  of  $R$  on  $G$  starting with an initial configuration  $c$  is an infinite sequence  $c_1 c_2 c_3 \dots$  of configurations such that  $c_1 = c$  and for all  $i$ ,  $c_{i+1}$  results from  $c_i$ .

The *set of positions* of a run  $\alpha = (v_1, q_1)(v_2, q_2) \dots$  is the set of positions  $\{v_1, v_2, \dots\}$  of its configurations. The *sequence of positions* of a run  $\alpha$  is the sequence of positions  $v_1 v_2 \dots$  of its configurations.

*Multi-Robot Systems.*

We extend the previous definitions to  $k$ -many of robots.

A *k*-robot ensemble is a sequence  $\langle R_1, \dots, R_k \rangle$  where, writing  $R_i = (Q_i, \delta_i)$ , each  $Q_i$  is a finite set of *states*, and each  $\delta_i \subset Q_i \times \text{INS}_{\Sigma, k} \times Q_i$  is a finite *transition* relation, where the *instruction set* is  $\text{INS}_{\Sigma, k} := \{\uparrow_{\sigma} : \sigma \in \Sigma\} \cup \text{MSOL}_k(\Sigma)$ . For a sequence  $ins \in ((\text{INS}_{\Sigma, k})^k)^*$  define<sup>6</sup>  $[[ins]]_G \subseteq V^{2k}$  such that  $(\bar{u}, \bar{v}) \in [[ins]]_G$  if and only if, in  $G$ , one can reach  $\bar{v}$  from  $\bar{u}$  by following the instructions in  $ins$ .

Formally,

1. If  $ins = \epsilon$ , then  $(\bar{u}, \bar{v}) \in [[ins]]_G$  if and only if  $\bar{u} = \bar{v}$ ;
  2. If  $ins = (d_1, \dots, d_k) \in \text{INS}_{\Sigma, k}$  then  $(\bar{u}, \bar{v}) \in [[ins]]_G$  if, for each  $i \leq k$ :
    - (a) if  $d_i = \uparrow_\sigma$  then  $\lambda(u_i, v_i) = \sigma$ ,
    - (b) if  $d_i = \tau(x_1, \dots, x_k)$  then  $u_i = v_i$  and  $G \models \tau(u_1, \dots, u_k)$ .

<sup>6</sup>To improve readability, the notation  $[[\cdot]]_G$  does not mention  $k$ .

3. If  $ins = d \cdot e$  then  $(\overline{u}, \overline{v}) \in [[ins]]_G$  if and only if there exists  $\overline{z} \in V$  such that  $(\overline{u}, \overline{z}) \in [[d]]_G$  and  $(\overline{z}, \overline{v}) \in [[e]]_G$ .

Fix a  $\Sigma$ -graph  $G$ . A *configuration*  $c$  of  $\langle R_1, \dots, R_k \rangle$  on graph  $G$  is a pair  $\langle \overline{v}, \overline{q} \rangle \in V^k \times \prod_{i \leq k} Q_i$ . A configuration is *initial* if  $q_i$  is the initial state of robot  $R_i$  (for all  $i \leq k$ ). The following definition expresses that one configuration results from another after the robots simultaneously execute their own next instruction (which may be to move along an edge labeled  $\sigma$ , or to test the current position of all the robots):<sup>7</sup> configuration  $\langle \overline{w}, \overline{q} \rangle$  *results from*  $\langle \overline{v}, \overline{p} \rangle$  iff there are transitions  $(p_i, ins_i, q_i) \in \delta_i$  (for  $i \leq k$ ) such that  $(\overline{v}, \overline{w}) \in [[(ins_1, \dots, ins_k)]]_G$ . *Runs* and their *sets of positions and sequences* are defined as before.

## *Robot Tasks.*

Robots should achieve some task in their environment: a *k*-robot task, or simply a *task*,  $T$ , is a function that maps a graph  $G$  to a set of sequences of positions of  $G$ , i.e.,  $T(G) \subseteq (V^k)^\omega$ . A robot-ensemble  $\bar{R}$  achieves  $T$  on  $G$  if for every run  $\alpha$  of  $\bar{R}$  on  $G$  it holds that  $\alpha' \in T(G)$ , where  $\alpha'$  is the sequence of positions of the run  $\alpha$ .

We give some examples of foundational robot tasks [34]:

- RT1. A robot *explores and halts* if, no matter where it starts, it a) eventually halts, and b) visits every vertex of the graph at least once.
  - RT2. A robot *perpetually explores* if, no matter where it starts, it visits every vertex of  $G$  infinitely often.
  - RT3. A robot *explores and returns* if, no matter where it starts, it a) eventually stops where it started, and b) visits every vertex of the graph at least once.
  - RT4. An ensemble of robots *collaboratively explores* a graph if, no matter where they start, every node is eventually visited by at least one robot.
  - RT5. A  $k$ -robot ensemble *gathers* if, no matter where each robot starts, there is a vertex  $z$ , such that eventually every robot is in  $z$ .

#### 4. REASONING ABOUT ROBOT SYSTEMS

We formalise the parameterised verification problem for robot protocols, and then describe our solution to it (Theorem 2) which shows how to reduce parameterised verification in the case of a single robot to the logical validity problem of certain logics.

#### 4.1 The Parameterised Verification Problem

The parameterised verification problem depends on a set of graphs  $\mathcal{G}$ , a set of robot ensembles  $\mathcal{R}$ , and a robot task  $T$ . Note that  $\mathcal{G}$  is typically infinite.

DEFINITION 1. The *parameterised verification problem*  $\text{PVP}_T(\mathcal{G}, \mathcal{R})$  states: given a robot ensemble  $\bar{R}$  from  $\mathcal{R}$ , decide whether or not  $\bar{R}$  achieves the task  $T$  on every graph  $G \in \mathcal{G}$ .

Unfortunately, this problem is easily undecidable for the types of systems we have:

<sup>7</sup>This is where we model the assumption that robots act synchronously.

**THEOREM 1.** *There exists a task  $T$  and computable sets  $\mathcal{G}$  and  $\mathcal{R}$  such that  $\text{PVP}_T(\mathcal{G}, \mathcal{R})$  is undecidable.*

*In particular, we can choose  $k = 1$ ,  $T$  to be the task “never halt”,  $\mathcal{G}$  to be the set of unlabeled-grids, and  $\mathcal{R}$  to be all robots; or we can choose  $k = 2$ ,  $T$  to be the task “no robot ever halts”,  $\mathcal{G}$  to be the set of unlabeled-lines, and  $\mathcal{R}$  to be all robots.*

**PROOF SKETCH.** Since the proof technique is a standard, we merely sketch it. We reduce the non-halting problem for (Turing-powerful) two-counter machines to  $\text{PVP}_T(\mathcal{G}, \mathcal{R})$ , i.e., given machine  $M$  build robot(s)  $\bar{R}$  such that  $M$  accepts no input if and only if  $\bar{R}$  achieves task  $T$  on all graphs in  $\mathcal{G}$ .

Case  $k = 1$ : As observed by [8] for their “2-dimensional automata”, one robot on a grid can simulate a two-counter machine: counter values  $(n, m) \in \mathbb{N}^2$  are encoded by the robot being at position  $(n, m)$  of the grid. The only tests that are needed are to check whether or not the robot is on the boundary (this is to simulate the counter machine’s “test for zero”, as well as to alert about a counter overflow).

Case  $k = 2$ : The idea is that two distinguishable robots on a line can simulate one robot on a square grid; the position of the  $i$ th robot on the line gives the  $i$ th co-ordinate of the single robot on the grid. The robots only need to be able to test if they are at the end points of the line. Alternatively, one can directly code computations of Turing machines, as was done by [40] to show that universality of 2-head one-way finite-state automata is undecidable.  $\square$

In light of this negative result, our main task is to understand in what way we can restrict the systems to get decidability.

## 4.2 Reducing Parameterised Verification to Validity

We first describe, at a high level, the approach we use to solve (restricted cases of) the parameterised verification problem  $\text{PVP}_T(\mathcal{G}, \mathcal{R})$ . Suppose we can build, for every  $k$ -ensemble  $\bar{R}$  of robots, a formula  $\phi_{\bar{R}, T}$  such that for all graphs  $G$  the following are equivalent:

- $G \models \phi_{\bar{R}, T}$
- $\bar{R}$  achieves task  $T$  on  $G$ .

Then, for every  $\mathcal{R}$  and  $\mathcal{G}$ , we would have reduced the parameterised verification problem  $\text{PVP}_T(\mathcal{G}, \mathcal{R})$  to the  $\Phi_{\bar{R}, T}$ -validity problem for  $\mathcal{G}$  where  $\Phi_{\bar{R}, T}$  is the set of formulas  $\{\phi_{\bar{R}, T} : \bar{R} \in \mathcal{R}\}$ .<sup>8</sup>

We now detail this approach in the case of a single robot, i.e.,  $k = 1$ .

**LEMMA 1.** *Let  $R = (Q, \delta)$  be a robot over instruction set  $\text{INS}_{\Sigma, 1}$ , and let  $p, q \in Q$ .*

*We can build  $\text{MSOL}(\Sigma)$  formulas  $\psi_{R, p, q}(X, x, y)$  so that for every graph  $G$ :  $G \models \psi_{R, p, q}(X, x, y)$  if and only if there exists a run of  $R$  on  $G$  starting from configuration  $\langle x, p \rangle$  that has a prefix that reaches the configuration  $\langle y, q \rangle$  and the set of positions on the prefix is exactly  $X$ .*

*We can build  $\text{MSOL}(\Sigma)$  formulas  $\psi_{R, p, q}^\infty(X, x, y)$  so that for every graph  $G$ :  $G \models \psi_{R, p, q}^\infty(X, x, y)$  if and only if there is a run of  $R$  on  $G$  starting from configuration  $\langle x, p \rangle$  that reaches the configuration  $\langle y, q \rangle$  infinitely often and the set of positions on the run is exactly  $X$ .*

<sup>8</sup>Note that in our approach  $\phi_{\bar{R}, T}$  does not depend on  $\mathcal{R}$  or on  $\mathcal{G}$ .

**PROOF.** A robot  $R = (Q, \delta)$  is a finite automaton (without initial or final states) over a finite alphabet  $\text{Alph}$  of instructions, i.e.,  $\text{Alph} \subset \{\uparrow_\sigma : \sigma \in \Sigma\} \cup \text{MSOL}_1(\Sigma)$ . By Kleene’s theorem — which states that every finite-state automaton can be translated into a regular expression — we can build a regular expression  $\text{exp}$  (that depends on  $R, p, q$ ) over alphabet  $\text{Alph}$  for the language of the automaton  $R$  with initial state  $p$  and final state  $q$ .

By induction on the structure of regular expressions over alphabet  $\text{Alph}$  we build  $\text{MSOL}$  formulas:

- $\varphi_\emptyset := \text{false}$
- $\varphi_\epsilon(X, x, y) := x = y \wedge x \in X$
- $\varphi_{\uparrow_\sigma}(X, x, y) := \text{edg}_\sigma(x, y) \wedge x, y \in X$
- $\varphi_\tau(X, x, y) := x = y \wedge \tau(x) \wedge x \in X$
- $\varphi_{r+s} := \varphi_r \vee \varphi_s$
- $\varphi_{r \cdot s}(X, x, y) := \exists z [\varphi_r(X, x, z) \wedge \varphi_s(X, z, y)]$
- $\varphi_{r^*}(X, x, y) := \forall Z[(\text{cl}_{\phi_r}(X, Z) \wedge x \in Z) \rightarrow y \in Z]$   
where
- $\text{cl}_\phi(X, Z) := \forall a, b [(a \in Z \wedge \phi(X, a, b)) \rightarrow b \in Z]$ .

An easy induction shows that  $G \models \varphi_r(X, x, y)$  if and only if there is a sequence of instructions  $\text{ins} \in \text{Alph}^*$  accepted by the regular expression  $r$  and a path from  $x$  to  $y$  that follows instructions  $\text{ins}$ , and that only visits vertices in  $X$  (but not necessarily all of  $X$ ). Finally, define the  $\text{MSOL}$  formula  $\psi_{R, p, q}(X, x, y)$  to state that  $\phi_{\text{exp}}(X, x, y)$  holds and  $X$  is minimal:  $\phi_{\text{exp}}(X, x, y) \wedge \neg \exists Y (\phi_{\text{exp}}(Y, x, y) \wedge Y \subset X)$ .

Similarly, by a variation of Kleene’s Theorem, we can build an  $\omega$ -regular expression  $\text{exp}$  over alphabet  $\text{Alph}$  for the language consisting of all infinite sequences that label infinite paths in  $R$  that start in  $p$  and see  $q$  infinitely often. Then, inductively build  $\varphi_{\text{exp}}$ , as before, with the following additional rule:

$$\varphi_{r^\omega}(X, x, y) := \exists z [\varphi_{r^*}(X, x, y) \wedge \varphi_r(X, y, z) \wedge \varphi_{r^*}(X, z, y)]$$

As before, define the  $\text{MSOL}$  formula  $\psi_{R, p, q}^\infty(X, x, y)$  to state that  $X$  is minimal such that  $\phi_{\text{exp}}(X, x, y)$  holds.  $\square$

The idea of the proof of Lemma 1 follows [7] who built a formula expressing that there is a run starting in configuration  $\langle x, p \rangle$  that reaches configuration  $\langle y, q \rangle$ . Our Lemma extends this in two ways, i.e., recording the visited states  $X$ , and expressing that a configuration occurs infinitely often.

**NOTE 1.** *The case of multiple robots ( $k > 1$ ) is more subtle. For instance, the analogue of Lemma 1 does not hold. Indeed, for  $k = 2$ , let  $R_1 = R_2$  be robots that have one state  $p$  and that non-deterministically move in every direction. Then the statement “there is a run of the robots from configuration  $\langle x_1, x_2, p, p \rangle$  to configuration  $\langle y_1, y_2, p, p \rangle$ ” is equivalent to the statement that the  $k$ -ary transitive closure of the edge relation in graph  $G$  contains the tuple  $\langle x_1, x_2, y_1, y_2 \rangle$ . However, the  $k$ -ary transitive closure is not expressible in  $\text{MSOL}$  (as was pointed out in Example MF5 in Section 2).*

### 4.3 Robot Task Logic — RTL

We now define a logic, called **RTL**, for expressing robot tasks in the case of one robot ( $k = 1$ ).

**Syntax.** Formulas of **RTL** are built, as in the definition of **MSOL** from Section 2, from the following atomic formulas:  $x = y$ ,  $\text{Reach}(X, x, y)$ ,  $\text{Halt}(X, x, y)$ ,  $\text{Infty}(X, x, y)$ , and  $\text{Rept}(X, x, y)$ .

**Semantics.** Formulas of **RTL** are interpreted with respect to graphs and robots. Thus, given a graph  $G$  and a robot  $R$  with state set  $Q$ , initial-state set  $I$ , repeating-state set  $A$ , and halting-state set  $H$ , define the satisfaction relation  $\models_{\text{RTL}}$ :

$$\begin{aligned} \langle G, R \rangle \models_{\text{RTL}} \text{Reach}(X, x, y) &\quad \text{iff } G \models \bigwedge_{p \in I} \bigvee_{q \in Q} \psi_{R,p,q}(X, x, y) \\ \langle G, R \rangle \models_{\text{RTL}} \text{Halt}(X, x, y) &\quad \text{iff } G \models \bigwedge_{p \in I} \bigvee_{q \in H} \psi_{R,p,q}(X, x, y) \\ \langle G, R \rangle \models_{\text{RTL}} \text{Infty}(X, x, y) &\quad \text{iff } G \models \bigwedge_{p \in I} \bigvee_{q \in Q} \psi_{R,p,q}^\infty(X, x, y) \\ \langle G, R \rangle \models_{\text{RTL}} \text{Rept}(X, x, y) &\quad \text{iff } G \models \bigwedge_{p \in I} \bigvee_{q \in A} \psi_{R,p,q}^\infty(X, x, y) \end{aligned}$$

The formulas  $\psi_{R,p,q}$  and  $\psi_{R,p,q}^\infty$  are from Lemma 1. Extend the satisfaction relation to all formulas of **RTL** in the natural way.

**Examples.** Here are some example **RTL** formulas and their meanings.

1. The atomic formula  $\text{Reach}(X, x, y)$  expresses that the robot, starting in position  $x$ , reaches position  $y$ , and the set of visited vertices is  $X$ . The **RTL** formula  $\forall x \exists y \text{Reach}(V, x, y)$  expresses that the robot explores the graph, no matter where it starts.
2. The **RTL** formula  $\forall x \exists y \text{Halt}(V, x, y)$  expresses “explore and stop”.
3. The **RTL** formula  $\forall x \text{Halt}(V, x, x)$  expresses “explore and return”.
4. The atomic formula  $\text{Infty}(X, x, y)$  expresses that the robot, starting in position  $x$ , visits position  $y$  infinitely often, and the set of vertices the robot visits along this run is exactly  $X$ . Thus  $\forall x \text{Infty}(V, x, x)$  is an **RTL** formula expressing that the robot “perpetually explores” the graph.

A task  $T$  is *captured* by an **RTL** formula  $\vartheta$  if  $\langle G, R \rangle \models_{\text{RTL}} \vartheta$  is equivalent to the statement that every run of  $R$  on  $G$  is in  $T(G)$ . The example formulas show that the first three tasks from Section 3.1 are captured by **RTL** formulas.

Here is the main theorem that solves the parameterised verification problem in the case of a single robot ( $k = 1$ ). It reduces the **PVP** to **MSOL**-validity of the set of environments.

**THEOREM 2.** Fix an edge-label set  $\Sigma$ . Let  $\mathcal{R}$  be the set of all robots over  $\text{INS}_{\Sigma,1}$ , let  $T$  be a task that is captured by an **RTL** formula, and let  $\mathcal{G}$  be a set of  $\Sigma$ -graphs with decidable **MSOL**-validity problem. Then  $\text{PVP}_T(\mathcal{G}, \mathcal{R})$  is decidable.

**PROOF.** Say **RTL** formula  $\vartheta$  captures  $T$ . Given  $R \in \mathcal{R}$ , build the formula  $\phi_{R,T}$  by replacing every atomic formula in

$\vartheta$  by its definition with respect to  $R$ , e.g.,  $\text{Reach}(X, x, y)$  is replaced by  $\bigwedge_{p \in I} \bigvee_{q \in Q} \psi_{R,p,q}(X, x, y)$ . A routine induction on the structure of the formula  $\vartheta$  gives:  $\langle G, R \rangle \models_{\text{RTL}} \vartheta$  if and only if  $G \models \phi_{R,T}$ . But by Lemma 1  $\phi_{R,T}$  is a formula in **MSOL**( $\Sigma$ ). Thus we can apply the fact that the **MSOL**-validity problem for  $\mathcal{G}$  is decidable to decide whether or not  $G \models \phi_{R,T}$  for all  $G \in \mathcal{G}$ .  $\square$

**COROLLARY 1.** If  $\mathcal{R}$  and  $T$  are as in Theorem 2 and  $\mathcal{G}$  is a context-free set of graphs, then  $\text{PVP}_T(\mathcal{G}, \mathcal{R})$  is decidable.

**NOTE 2.** The case of multiple robots ( $k > 1$ ) is harder to analyse. Indeed, Theorem 1 states that **PVP** is undecidable already for  $k = 2$  on lines (which is a very basic context-free set of graphs) and simple reachability tasks. It is a challenging problem to find natural and useful restrictions on robots which allow the analogue of Corollary 1 (or Lemma 1) to hold in the case of multiple robots.

### 5. ILLUSTRATION: DISTRIBUTED COMPUTING

We now instantiate our framework to a popular model of autonomous mobile-agent from the distributed computing literature, i.e., [27, 21, 16, 33, 28, 18]. To distinguish their agents from ours, we call theirs *DC-robots*. Here is their model: environments are modeled as undirected graphs, and each vertex is annotated with a local port numbering i.e., for every vertex  $v$  the set of labels of the edges of  $v$  are in bijection with  $\{1, 2, \dots, \deg(v)\}$ . A DC-robot finding itself at vertex  $v$  can decide to exit via local port-number  $d$  and update its local state based on a) its current state, b) the identification of the robots at the same vertex  $v$ , c) the degree of  $v$ , and d) the local-port number of  $v$  of the edge it used when arriving at  $v$ . Thus graphs are assumed to have bounded degree  $\Delta$ . Typical tasks from this literature are “perpetual exploration”, “exploration with return”, and “gathering”. The main twist is that the robots should perform their task on a graph no matter the local port numbering.

Such robot systems are easily expressible in our framework. The edge-label set  $\Sigma$  is defined to be  $[\Delta]$ , the edge-relation  $E$  is assumed to be symmetric, and  $\lambda(v, w) = i$  codes that  $i$  is  $v$ 's port number for the undirected edge  $\{v, w\}$ . Note that  $\lambda(v, w)$  need not equal  $\lambda(w, v)$ . The interesting aspect is how to simulate d) above. Our robot must store in its state both the state of the DC-robot, as well as the local-port number with which it entered the current vertex. It does this as follows: when our robot is at vertex  $v$ , and the DC-robot says to take exit  $i$ , our robot first determines the  $w$  such that  $\lambda(v, w) = i$ , and then it determines the  $j \in \Delta$  such that  $\lambda(w, v) = j$ . It can do this with test-instructions in **FOL**( $\Sigma$ ).

**Rotor-Robot** We now give a simple but important example that can be analysed in our framework. The *rotor robot* operates as follows: when it enters a vertex  $v$  by port  $i$  it leaves by port  $i + 1$  modulo  $\deg(v)$ . This robot is known by various other names: Abelian mobile robot, rotor walk, ant walk, Eulerian walker, and Propp machine. It is important because it is a viable alternative to probabilistic robots that perform random walks [11]. It has the property that it explores all trees — a result that seems to be folklore, and that could be proved, for fixed degree  $\Delta$ , using Theorem 2.

**Label-Guided Tasks.** The graphs in this paper are edge-labeled, but our results hold allowing vertex-labels. Moreover, our framework can incorporate questions of the form: given a robot and a task, decide if for every graph  $G \in \mathcal{G}$  there exists a labeling of  $G$  such that the robot achieves the task on the graph with the help of the labeling. For instance, although there is no DC-robot that perpetually explores all graphs, there is a DC-robot and an algorithm that colours vertices by three colours, such that the robot can perpetually explore every such coloured graph [16]. Since the set of all graphs does not have decidable-validity, we might only hope to verify variations of this fact for restricted classes of graphs, e.g., fix a context-free set of graphs  $\mathcal{G}$ ; then one can decide, given a DC-robot, whether or not for every  $G \in \mathcal{G}$  the robot achieves the task “there is a colouring of  $G$  using three colours that the robot can use to perpetually explore  $G$ ”. The reason this fits into Theorem 2 is that this task is captured by an RTL formula — indeed, the property that  $X_1, X_2, X_3 \subseteq V$  colour a graph is easily expressed in MSOL (just say that  $\wedge_{i \neq j} X_i \cap X_j = \emptyset$  and  $X_1 \cup X_2 \cup X_3 = V$ ).

## 6. COMPLEXITY CONSIDERATIONS

As discussed in Section 4.2, the framework reduces the parameterised verification problem  $\text{PVP}_T(\mathcal{G}, \mathcal{R})$  to the  $\Phi_{\bar{R}, T}$ -validity problem for  $\mathcal{G}$  where  $\Phi_{\bar{R}, T}$  is the set of formulas  $\{\phi_{\bar{R}, T} : \bar{R} \in \mathcal{R}\}$ . Unfortunately, the complexity of the decision procedure for  $\text{PVP}_T(\mathcal{G}, \mathcal{R})$  in Corollary 1 may be non-elementary, i.e., not bounded by any tower of exponentials in the size of the input robot  $R$ . Indeed: although the size of the computed formula  $\phi_{R, T}$  is exponential in the size of the robot  $R$  and linear in the size of the RTL formula capturing  $T$ , but the complexity of the MSOL-validity problem for  $\mathcal{G}$  is non-elementary even taking  $\mathcal{G}$  to be the set of binary-labeled lines [42].

Consequently, we illustrate that improved decision procedures can be found for interesting tasks and sets of graphs.

A robot is *deterministic* if for all  $p \in Q$ , either a) there exists  $q, q' \in Q$  and  $\tau \in \text{MSOL}(\Sigma)$  and the only transitions out of  $p$  are  $(p, \tau, q)$  and  $(p, \neg\tau, q')$ , or b) there exists  $\sigma \in \Sigma, q, q' \in Q$  and the only transitions out of  $p$  are  $(p, \uparrow_\sigma, q)$  and  $(p, \neg\exists z(\text{edg}_\sigma(x, z)), q')$ . In other words, a) says that if test  $\tau$  holds goto state  $q$  else goto state  $q'$ , and b) says that if there is an edge in the graph labeled  $\sigma$  then traverse it (in case of many such edges, pick one nondeterministically) and go to state  $q$ , otherwise goto state  $q'$ . A  $\Sigma$ -graph is *deterministic* if  $\lambda(v, w) = \lambda(v, w')$  implies  $w = w'$ . For instance,  $\Delta$ -ary trees are deterministic. Note that deterministic robots have at most one run on deterministic graphs.

**THEOREM 3.** Fix  $\Delta \in \mathbb{N}$ . Let  $\mathcal{G}$  be the  $\Delta$ -ary trees, let  $\mathcal{R}$  be all deterministic robots, and let  $T$  be the “explore and halt” task. Then  $\text{PVP}_T(\mathcal{G}, \mathcal{R})$  is EXPTIME-COMPLETE.

**PROOF SKETCH.** The idea is to inter-reduce the parameterised verification problem with the universality problem for deterministic tree-walking automata (DTWA). A DTWA is a deterministic machine that can recognise sets of trees: the automaton starts at the root, at any given time the automaton sits on a node of the input tree, can test if the current node is a leaf, the root, or the  $i$ th child (for  $i \leq \Delta$ ), and based on these tests the machine updates its internal state and executes one of the commands: “accept the tree”, “reject the tree”, “go to the parent” or “go the  $i$ th child”. The universality of DTWA is EXPTIME-COMPLETE. Indeed, from

a DTWA  $A$  build a DTWA  $B$  that simulates  $A$  and rejects whenever  $A$  runs forever (this causes a quadratic blowup in the number of states [36]); then complement  $B$  by swapping accept and reject states to get a DTWA  $C$ ; then convert  $C$  into an ordinary frontier-to-root tree automaton  $D$  using a subset construction that calculates loops of  $C$  [9, Fact 1] (this causes an exponential blowup); then test  $D$  for emptiness (which can be done in P).

Here is the reduction that gives the upper bound: given a deterministic robot  $R$  build a DTWA  $A_R$  that operates on trees  $t$  with marked nodes  $s$  and  $v$ , written  $(t, s, v)$ , as follows: first it does a DFS from the root, and when it reaches  $s$  it begins the simulation of  $R$ ; after the simulation begins,  $A_R$  remembers if  $v$  is visited; if  $R$  enters a halt state and  $v$  was visited then  $A_R$  accepts  $(t, s, v)$ , and if  $R$  enters a halt state and  $v$  was not visited then  $A_R$  rejects input  $(t, s, v)$ . Thus:  $R$  “explores and halts” iff for all  $(t, s, v)$  the run of  $R$  on  $t$  starting at  $s$  visits  $v$  and later enters a halt state iff  $A_R$  accepts all inputs of the form  $(t, s, v)$ . The size of  $A_R$  is linear in the size of  $R$ .

Here is the reduction that gives the lower bound: given a DTWA  $A$ , build a robot  $R_A$  which first explores the input tree  $t$  (doing, say, a DFS), then simulates  $A$  from the root, and halts iff  $A$  accepts (thus  $R_A$  runs forever if  $A$  rejects). Since  $R_A$  always explores its input,  $A$  accepts  $t$  iff  $R_A$  explores and halts on  $t$ . The size of  $R_A$  is linear in the size of  $A$ .  $\square$

**NOTE 3.** For every  $\Delta \geq 2$ , there is no robot that “explores and halts” on all local port-numbered  $\Delta$ -ary trees [21]. The proof above is easily adapted to yield an algorithm that, given a robot  $R$  and  $\Delta \geq 2$ , returns a local-port numbered tree on which the robot does not succeed to “explore and halt”.

## 7. COMPARISON WITH RELATED WORK

Robot systems are considered distributed if they involve autonomous agents with no central control. In light of this, we first describe the relationship of our work to formal verification of distributed systems generally, and then to formal verification of robot protocols in particular.

### Formal verification of distributed systems.

Typical parameters that arise in the study of distributed systems are the number of agents, the number of assumed faulty agents, etc. Since parameterised problems of distributed systems are, in general, undecidable [3, 43], the formal methods community developed sound but incomplete methods that require human intervention, e.g., counter- and predicate-abstractions, inductive invariants, regular model checking and acceleration techniques. See for instance [39, references on pages 2-3].

On the other hand, by simplifying the systems one can get decidable PVP. These use techniques from games, automata theory and logic, notably finite-model properties/cutoffs, reduction to Petri nets, and the theory of well-structured transition systems [24, 26, 23, 15, 31, 19, 1, 2].

Of all these models, token-passing systems (i.e., the models in [24, 15, 1, 2]) are the closest to robots — both tokens and robots move along the vertices of graphs. However, we now argue that the model in these cited papers is incomparable with our model. First, the translation of their token-passing systems into robot systems would require that robots can read and write to variables at the vertices (this

is to model the fact that processes have states). However, we assumed environments are static (because otherwise the PVP is quickly undecidable). Conversely, translating robot-systems into the cited token-passing systems requires that the robot-systems satisfy the following unrealistic assumptions (that are used in their decidability proofs): when a robot decides to move, an adversary decides which edge it takes, as well as to which of its internal states to transition (i.e., even the robot’s memory may be scrambled). Not even the simplest robots from the distributed computing literature (e.g., the rotor robot, or the DFS robot) satisfy this double restriction.

#### *Formal verification of robot systems.*

The formal methods community has only recently [30, 20, 12, 6, 4, 35] begun explicitly verifying and synthesising robot protocols (rather than distributed systems in general). However, half of these papers only treat small values of the parameters. For instance, one such paper concludes [6, page 11]:

*While our method is parameterised by both  $k$  [the number of robots] and  $n$  [the size of the graph], it does not permit to verify whether a [robot] protocol is valid for every  $k$  and  $n$  satisfying a particular predicate.*

The papers that do treat parameterised robot protocols do not give sound and complete decision procedures for their systems, as we do. Indeed, [4] uses a proof assistant to provide certificates (formal proofs) of impossibility results about robot networks; [35] uses the theory of games on graphs to synthesise a robot protocol for gathering  $k$  robots on a ring of size  $n$  (for small values of  $k, n$ ), and relies on a hand-proven induction to prove that the synthesised robot protocol works for all values of the parameters  $k, n$ ; and [32] presents a sound technique that may identify cutoffs using a counter-abstraction in order to draw conclusions about certain swarm algorithms, independently of the number of swarming agents.

The quotation above continues:

*Adapting recent advances in parameterised model checking [citation elided] would be a nice way to obtain such results.*

Both the methodology (reducing parameterised verification of robot systems to validity problems in logic) and the results of this paper (algorithms for automatic parameterised verification of robot systems consisting of a single robot in an unknown environment) are novel and have succeeded where other methods and “recent advances” (discussed in the previous subsection) have not.

#### *Comparison with graph-walking automata.*

**Graph-walking automata.** There is no canonical definition of automaton on graphs. Our model of a single robot is equivalent to graph-walking automata with MSOL-tests [7]. The proof idea of Lemma 1 (which compiles robots into formulas over graphs) is borrowed from [7, 25]. Other classical notions of automata on graphs (e.g., [8]) are often too expressive and lead to undecidable parameterised verification problems (see the proof of Theorem 1).

**Multi-head automata.** If agents are modeled as finite-state machines, then multi-agent systems are instances of

multi-head automata, and the parameterised verification problem for reachability tasks is equivalent to the universality problem of such automata. A technical difference between our robot-ensembles and multi-head automata is that the  $k$ -many heads of a multi-head automaton are operated from a central control. In the language of robots this would mean that the robots can communicate their current local states to each other. Such communication is disallowed as it quickly results in undecidability. However, the proof of Theorem 1 shows that, similarly, even simple vision-based communication leads to undecidability.

**Tree-walking automata (TWA)** are a natural generalisation to trees of two-way automata on words. Tree-walking automata with a single head, and their corresponding regular expressions, have been studied for their own sake [9], in formal verification, e.g., [46, 10, 37], and as tree pattern-matching tools [13, 41]. TWA are used in Theorem 3 to reason about a single robot walking on unknown trees.

## 8. FUTURE CHALLENGES

This paper takes a step in the following general research agenda: find natural robot systems that have decidable or tractable PVP.

Regarding decidability, much work remains to be done. An important problem is to extend the methodology or results of this paper to multiple robots. Notes 1 and 2 discuss the challenges facing such an investigation.

Similarly, in light of the fact that PVP is quickly undecidable in a dynamic environment (e.g., this is the case for simple reachability tasks of a single robot that can read and write to every vertex on a line, cf. [43, 24]), what restrictions on the robot or the dynamic environment will result in decidable PVP?

On the other hand, we believe it is feasible to extend our framework to quantitative tasks, such as minimising the number of steps to complete a task.

Regarding complexity, we have seen that our general algorithm for solving the PVP (in Theorem 2) has high computational complexity, and we have seen (in Theorem 3) that a certain problem on trees is EXPTIME-COMPLETE. For which systems (tasks, classes of graphs, and classes of robots) is the PVP solvable in P, NP, or PSPACE?

We believe that tackling these questions will open avenues both in automata theory and in the verification of mobile multi-agent systems in unknown environments.

## Acknowledgements

I thank the reviewers for their careful reading and provocative questions which helped improve the content and presentation of this paper.

## REFERENCES

- [1] B. Aminof, S. Jacobs, A. Khalimov, and S. Rubin. Parameterized model checking of token-passing systems. In *VMCAI*, pages 262–281, 2014.
- [2] B. Aminof, T. Kotek, F. Spegni, S. Rubin, and H. Veith. Parameterized model checking of rendezvous systems. In *CONCUR*, pages 109–124, 2014.
- [3] K. Apt and D. Kozen. Limits for automatic verification of finite-state concurrent systems. *Inf. Process. Lett.*, 15:307–309, 1986.

- [4] C. Auger, Z. Bouzid, P. Courtieu, S. Tixeuil, and X. Urbain. Certified impossibility results for byzantine-tolerant mobile robots. In *SSS*, pages 178–190, 2013.
- [5] M. A. Bender, A. Fernandez, D. Ron, A. Sahai, and S. Vadhan. The power of a pebble: Exploring and mapping directed graphs. *Information and Computation*, 176(1):1–21, 2002.
- [6] B. Berard, L. Millet, M. Potop-Butucaru, Y. Thierry-Mieg, and S. Tixeuil. Formal verification of mobile robot protocols. Report <hal-00834061>, 2013.
- [7] R. Bloem and J. Engelfriet. Monadic second order logic and node relations on graphs and trees. In *Structures in Logic and Computer Science*, pages 144–161, 1997.
- [8] M. Blum and C. Hewitt. Automata on a 2-dimensional tape. *SWAT (FOCS)*, pages 155–160, 1967.
- [9] M. Bojańczyk. Language and automata theory and applications. *Lecture Notes in Computer Science Volume 5196*, pages 1–2, 2008.
- [10] P. A. Bonatti, C. Lutz, A. Murano, and M. Y. Vardi. The complexity of enriched  $\mu$ -calculi. *Logical Methods in Computer Science*, 4(3:11):1–27, 2008.
- [11] B. Bond and L. Levine. Abelian networks: foundations and examples. *CoRR*, abs/1309.3445, 2013.
- [12] F. Bonnet, X. Defago, F. Petit, M. Potop-Butucaru, and S. Tixeuil. Brief announcement: Discovering and assessing fine-grained metrics in robot networks protocols. In *SSS*, pages 282–284. 2012.
- [13] A. Brüggemann-Klein and D. Wood. Caterpillars: A context specification technique. *Markup Languages*, 2:81–106, 2000.
- [14] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
- [15] E. Clarke, M. Talupur, T. Touili, and H. Veith. Verification by network decomposition. In *CONCUR 2004*, volume 3170, pages 276–291, 2004.
- [16] R. Cohen, P. Fraigniaud, D. Ilcinkas, A. Korman, and D. Peleg. Label-guided graph exploration by a finite automaton. In *ICALP*, pages 335–346. 2005.
- [17] B. Courcelle and J. Engelfriet. Book: Graph structure and monadic second-order logic. a language-theoretic approach. *Bull. EATCS*, 108:179, 2012.
- [18] S. Das. Mobile agents in distributed computing: Network exploration. *Bull. EATCS*, 109:54–69, 2013.
- [19] G. Delzanno. Parameterized verification and model checking for distributed broadcast protocols. In *ICGT*, pages 1–16, 2014.
- [20] S. Devismes, A. Lamani, F. Petit, P. Raymond, and S. Tixeuil. Optimal grid exploration by asynchronous oblivious robots. *CoRR*, abs/1105.2461, 2011.
- [21] K. Diks, P. Fraigniaud, E. Kranakis, and A. Pelc. Tree exploration with little memory. *Journal of Algorithms*, 51(1):38–63, 2004.
- [22] H. Ebbinghaus and J. Flum. *Finite model theory*. Perspectives in Mathematical Logic. Springer, 1995.
- [23] E. Emerson and V. Kahlon. Model checking guarded protocols. In *LICS*, pages 361–370, 2003.
- [24] E. Emerson and K. Namjoshi. On reasoning about rings. *Int. J. Found. Comput. Sci.*, 14(4):527–550, 2003.
- [25] J. Engelfriet and H. J. Hoogeboom. Nested pebbles and transitive closure. In *STACS*, pages 477–488, 2006.
- [26] J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *LICS*, pages 352–359, 1999.
- [27] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. *Theoretical Computer Science*, 345:331 – 344, 2005.
- [28] L. Gasieniec and T. Radzik. Memory efficient anonymous graph exploration. In H. Broersma, T. Erlebach, T. Friedetzky, and D. Paulusma, editors, *Graph-Theoretic Concepts in Computer Science*, volume 5344 of *LNCS*, pages 14–29. 2008.
- [29] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *LNCS*. Springer, 2002.
- [30] X. Huang, P. Maupin, and R. Van Der Meyden. Model checking knowledge in pursuit evasion games. In *IJCAI*, pages 240–245, 2011.
- [31] P. Kouvaros and A. Lomuscio. Automatic verification of parameterised multi-agent systems. In *AAMAS*, pages 861–868, 2013.
- [32] P. Kouvaros and A. Lomuscio. A counter abstraction technique for the verification of robot swarms. To appear in *AAAI*, 2015.
- [33] E. Kranakis, D. Krizanc, and S. Rajsbaum. Mobile agent rendezvous: A survey. *SIROCCO*, pages 1–9, 2006.
- [34] E. Kranakis, D. Krizanc, and S. Rajsbaum. Computing with mobile agents in distributed networks. In S. Rajasekaran and J. Reif, editors, *Handbook of Parallel Computing: Models, Algorithms, and Applications*, pages 8–1 to 8–20. Chapman Hall/CRC Computer and Inf. Sci. Series, 2007.
- [35] L. Millet, M. Potop-Butucaru, N. Sznajder, and S. Tixeuil. On the synthesis of mobile robots algorithms: The case of ring gathering. In *SSS*, pages 237–251, 2014.
- [36] A. Muscholl, M. Samuelides, and L. Segoufin. Complementing deterministic tree-walking automata. *Inf. Process. Lett.*, 99(1):33–39, 2006.
- [37] J. Obdržálek. Fast mu-calculus model checking when tree-width is bounded. In *CAV*, pages 80–92, 2003.
- [38] A. Pelc. Disc 2011 invited lecture: Deterministic rendezvous in networks: Survey of models and results. In *DISC*, pages 1–15, 2011.
- [39] A. Pnueli, J. Xu, and L. Zuck. Liveness with  $(0,1,\infty)$ -counter abstraction. In *CAV*, pages 93–111, 2002.
- [40] A. Rosenberg. On multi-head finite automata. *IBM Journal of Research and Development*, 10(5):388–394, Sep 1966.
- [41] T. Schwentick. Foundations of xml based on logic and automata: A snapshot. In *FoIKS*, pages 23–33, 2012.
- [42] L. J. Stockmeyer. *The complexity of decision problems in automata theory and logic*. PhD thesis, MIT, 1974.
- [43] I. Suzuki. Proving properties of a ring of finite-state machines. *Inf. Process. Lett.*, 28(4):213–214, July 1988.
- [44] W. Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 133–192, 1990.

- [45] W. Thomas. Languages, automata, and logic. In *Handbook of Formal Languages*, pages 389–455, 1996.
- [46] M. Y. Vardi. Reasoning about the past with two-way automata. In *ICALP*, pages 628–641, 1998.

## **4.6 Publication**

The rest of this page is intentionally left blank

# Graded Strategy Logic: Reasoning about Uniqueness of Nash Equilibria

Benjamin Aminof  
Technische Universität Wien, Austria  
benj@forsyte.tuwien.ac.at

Vadim Malvone, Aniello Murano, Sasha Rubin  
Università degli Studi di Napoli Federico II, Italy  
{vadim.malvone, anielo.murano,  
sasha.rubin}@unina.it

## ABSTRACT

Strategy Logic (SL) is a well established formalism for strategic reasoning in multi-agent systems. In a nutshell, SL is built over LTL and treats strategies as first-order objects that can be associated with agents by means of a binding operator. In this work we introduce Graded Strategy Logic (GRADED-SL), an extension of SL by graded quantifiers over tuples of strategy variables such as “there exist at least  $g$  different tuples  $(x_1, \dots, x_n)$  of strategies”. We study the model-checking problem of GRADED-SL and prove that it is no harder than for SL, i.e., it is non-elementary in the quantifier rank.

We show that GRADED-SL allows one to count the number of different strategy profiles that are Nash equilibria (NE), or subgame-perfect equilibria (SPE). By analyzing the structure of the specific formulas involved, we conclude that the important problems of checking for the existence of a unique NE or SPE can both be solved in 2EXPTIME, which is not harder than merely checking for the existence of such equilibria.

## Categories and Subject Descriptors

I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence — *Multiagent Systems*

## General Terms

Theory, Verification

## Keywords

Strategic logics; Graded modalities; Nash equilibrium

## 1. INTRODUCTION

Strategy Logic (SL) is a powerful formalism for reasoning about strategies in multi-agent systems [43, 44]. Strategies tell an agent what to do — they are functions that prescribe an action based on the history. The fundamental idea in SL is to treat strategies as first-order object. A strategy  $x$  can be quantified existentially  $\langle\langle x \rangle\rangle$  (read: there exists a strategy  $x$ ) and universally  $\llbracket x \rrbracket$  (read: for all strategies  $x$ ). Furthermore, strategies are not intrinsically glued to specific agents: the *binding* operator  $(\alpha, x)$  allows one to bind an

**Appears in:** *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016)*, J. Thangarajah, K. Tuyls, C. Jonker, S. Marsella (eds.), May 9–13, 2016, Singapore.

Copyright © 2016, International Foundation for Autonomous Agents and Multiagent Systems ([www.ifaamas.org](http://www.ifaamas.org)). All rights reserved.

agent  $\alpha$  to the strategy  $x$ . SL strictly subsumes several other logics for strategic reasoning including the well known ATL and ATL\* [3]. Being a very powerful logic, SL can directly express many solution concepts [9, 19, 32, 35, 43] among which that a strategy profile  $\bar{x}$  is a Nash equilibrium, and thus also the existence of a Nash equilibrium (NE).

Nash equilibrium is one of the most important concepts in game theory, forming the basis of much of the recent fundamental work in multi-agent decision making. A challenging and important aspect is to establish whether a game admits a *unique* NE [2, 20, 48]. This problem impacts on the predictive power of NE since, in case there are multiple equilibria, the outcome of the game cannot be uniquely pinned down [21, 49, 57]. Unfortunately, uniqueness has mainly been established either for special cost functions [2], or for very restrictive game topologies [47]. More specifically, uniqueness of NE in game theory is proved with procedures that are separated from the ones that check for its existence [2]; these procedures require various transformations of the best response functions of individual contributors [28, 29, 53], and there is no general theory that can be applied to different application areas [2].

In this paper, we address and solve the problem of expressing the uniqueness of certain solution concepts (and NE in particular) in a principled and elegant way. We introduce an extension of SL called GRADED-SL and study its model-checking problem. More specifically, we extend SL by replacing the quantification  $\langle\langle x \rangle\rangle$  and  $\llbracket x \rrbracket$  over strategy variables with *graded quantification over tuples of strategy variables*:  $\langle\langle x_1, \dots, x_n \rangle\rangle^{\geq g}$  (read  $\langle\langle x_1, \dots, x_n \rangle\rangle^{\geq g}$  as “there exist at least  $g$  different tuples  $(x_1, \dots, x_n)$  of strategies”) and its dual  $\llbracket x_1, \dots, x_n \rrbracket^{< g}$  ( $g \in \mathbb{N}$ ). Here, two strategies are different if they disagree on some history. That is, we count strategies syntactically as is usual in graded extensions of modal and description logics [6, 12, 15, 30, 36]. The key for expressing uniqueness of NE is the combination of quantifying over tuples (instead of singleton variables), and adding counting (in the form of graded modalities).

We address the model-checking problem for GRADED-SL and prove that it has the same complexity as SL, i.e., it is non-elementary in the nesting depth of quantifiers. In particular, we show that model checking GRADED-SL formulas with a nesting depth  $k > 0$  of blocks of quantifiers (a block of quantifiers is a maximally-consecutive sequence of quantifiers of the same type, i.e., either all existential, or all universal) is in  $(k+1)\text{EXPTIME}$ , and that for the special case where the formula starts with a block of quantifiers, it is in  $k\text{EXPTIME}$ . Since many natural formulas contain a very small number of

quantifiers, the complexity of the model-checking problem is not as bad as it seems. Specifically, several solution concepts can be expressed as SL formulas with a small number of quantifiers [9, 19, 32, 35, 43]. Since the existence of a NE, and the fact that there is at most one NE, can be expressed in GRADED-SL using simple formulas we are able to conclude that the problem of checking the uniqueness of a NE can be solved in 2EXPTime. Previously, it was only known that existence of NE can be checked in 2EXPTime [32, 43], and indeed it is 2EXPTime-complete [4, 32, 50]. Thus, GRADED-SL is the first logic that can solve the existence and uniqueness of NE (as well as many other solution concepts) in a uniform way.

SL has a few natural syntactic fragments, the most powerful of which is called Nested-Goal SL which can express NE [43]. Similarly, we define GRADED-SL *Nested-Goal*, a syntactic fragment of GRADED-SL. The Nested-Goal restriction encompasses formulas in a special prenex normal form with a particular nested temporal structure that restricts the application of both strategy quantifiers and agent bindings (see Section 2 for details). We show that the graded extension of Nested-Goal SL has the same model-checking complexity, i.e., non-elementary in the *alternation number* of the quantifiers appearing in the formula (the alternation number is, roughly speaking, the maximum number of existential/universal quantifier switches [43]). Since uniqueness of NE can be expressed in Nested-Goal GRADED-SL using alternation one, we get an alternative proof for checking the NE uniqueness in 2EXPTime.

We exemplify our definition and our automata-theoretic algorithm for the model-checking problem with a large number of applications. In particular, we use it for reasoning about repeated one-shot games such as the *iterated prisoner’s dilemma* (IPD) [51]. The prisoner’s dilemma game is a popular metaphor for the problem of stable cooperation and has been widely used in several application domains [8]. In the classic definition, it consists of two players, each of them has an option to defect or collaborate. More involved is the IPD in which the process is repeated and one can model reactive strategies in continuous play. The IPD has become a standard model for the evolution of cooperative behaviour within a community of egoistic agents, frequently cited for implications in both sociology and biology (see [8]). Evaluating the existence of NE in an IPD and, even more, its uniqueness, is a very challenging and complicated question due to the rich set of strategies such a game can admit [8, 16, 17]. In particular, such infinite-duration games need to be supported by more complex solution concepts such as *subgame-perfect* equilibrium [27, 41, 55]. Thanks to GRADED-SL and the related model-checking result, we get that checking the uniqueness of a NE in an IPD can be solved in 2EXPTime.

**Related work.** The importance of solution concepts, verifying a unique equilibrium, and the relationship with logics for strategic reasoning is discussed above. We now give some highlights from the long and active investigation of graded modalities in the formal verification community.

*Graded modalities* were first studied in modal logic [26] and then exported to the field of *knowledge representation* to allow quantitative bounds on the set of individuals satisfying a given property. Specifically, they were considered as *counting quantifiers* in first-order logics [31] and *number restrictions* in *description logics* [34]. *Graded  $\mu$ -calculus*, in which immediate-successor accessible worlds are counted, was intro-

duced to reason about graded modal logic with fixed-point operators [36]. Recently, the notion of graded modalities was extended to count the number of paths in the branching-time temporal logic formulas CTL and CTL\* [7, 10]. In the verification of reactive systems, we mention two orthogonal approaches: module checking for graded  $\mu$ -calculus [6, 25] and an extension of ATL by graded path modalities [24].

The work closest to ours is [40]: also motivated by counting NE, it introduces a graded extension of SL, called GSL. In contrast with our work, GSL has a very intricate way of counting strategies: it makes use of a semantic definition of strategies being equivalent, and counting in which equivalent strategies are counted as a single strategy. While this approach has been proved to be sound and general, it heavily complicates the model-checking problem. Indeed, only a very weak fragment of GSL has been solved in [40] by exploiting an *ad hoc* solution that does not seem to be easily scalable to (all of) GSL. Precisely, the fragment investigated there is the vanilla restriction of the graded version of one-goal SL [42]. There is a common belief that the one-goal fragment is not powerful enough to express the existence of a Nash Equilibrium in concurrent games. The smallest fragment that is known to be able to represent this is the so called Boolean-goal Strategy Logic, whose graded extension (in the GSL sense) has no known solution.<sup>1</sup>

**Outline.** The sequel of the paper is structured as follows. In Section 2 we introduce GRADED-SL and provide some preliminary related concepts. In Section 3 we address the model-checking problem for GRADED-SL and its fragments. We conclude with Section 4 in which we have a discussion and suggestions for future work.

## 2. GRADED STRATEGY LOGIC

In this section we introduce Graded Strategy Logic, which we call GRADED-SL for short.

### 2.1 Models

Sentences of GRADED-SL are interpreted over *concurrent game structures*, just as for ATL and SL [3, 43].

**DEFINITION 2.1.** A concurrent game structure (CGS) is a tuple  $\mathcal{G} \triangleq \langle AP, Ag, Ac, St, s_I, ap, tr \rangle$ , where AP, Ag, Ac, and St are the sets of atomic propositions, agents, actions and states, respectively,  $s_I \in St$  is the initial state, and  $ap : St \rightarrow 2^{AP}$  is the labeling function mapping each state to the set of atomic propositions true in that state. Let  $Dc \triangleq Ag \rightarrow Ac$  be the set of decisions, i.e., functions describing the choice of an action by every agent. Then,  $tr : Dc \rightarrow (St \rightarrow St)$  denotes the transition function mapping every decision  $\delta \in Dc$  to a function  $tr(\delta) : St \rightarrow St$ .

We will usually take the set Ag of agents to be  $\{\alpha_1, \dots, \alpha_n\}$ .

A path (from  $s$ ) is a finite or infinite non-empty sequence of states  $s_1 s_2 \dots$  such that  $s = s_1$  and for every  $i$  there exists a decision  $\delta$  with  $tr(\delta)(s_i) = s_{i+1}$ . The set of paths starting

<sup>1</sup>In [33] it has been shown that, in the restricted case of turn-based structures it is possible to express the existence of Nash equilibria in  $m^-$ ATL\* [45], a memory-full variant of ATL\* (hence included in one-goal SL), but exponentially more succinct — and thus with a much more expensive model-checking algorithm. As also the authors in [33] state, it is not clear how to extend this result to the concurrent setting, even in the two player case.

with  $s$  is denoted  $\text{Pth}(s)$ . The set of finite paths from  $s$ , called the *histories (from s)*, is denoted  $\text{Hst}(s)$ . A *strategy (from s)* is a function  $\sigma \in \text{Str}(s) \triangleq \text{Hst}(s) \rightarrow \text{Ac}$  that prescribes which action has to be performed given a certain history. We write  $\text{Pth}, \text{Hst}, \text{Str}$  for the set of all paths, histories, and strategies (no matter where they start).

We use the standard notion of equality between strategies, [38], i.e.,  $\sigma_1 = \sigma_2$  iff for all  $\rho \in \text{Hst}$ ,  $\sigma_1(\rho) = \sigma_2(\rho)$ . This extends to equality between two  $n$ -tuples of strategies in the natural way, i.e., coordinate-wise.

## 2.2 Syntax

We describe the syntax as well as related basic concepts. GRADED-SL extends SL by replacing the classic singleton strategy quantifiers  $\langle\langle x \rangle\rangle$  and  $\llbracket x \rrbracket$  with the graded (tupled) quantifiers  $\langle\langle x_1, \dots, x_n \rangle\rangle^{\geq g}$  and  $\llbracket x_1, \dots, x_n \rrbracket^{< g}$ , respectively, where each  $x_i$  belongs to a countable set of variables  $\text{Vr}$  and  $g \in \mathbb{N}$  is called the *degree* of the quantifier. Intuitively, these are read as “there exist at least  $g$  tuples of strategies  $(x_1, \dots, x_n)$ ” and “all but less than  $g$  many tuples of strategies”, respectively. The syntax  $(\alpha, x)$  denotes a *binding* of the agent  $\alpha$  to the strategy  $x$ . The syntax of GRADED-SL is:

**DEFINITION 2.2.** GRADED-SL formulas are built inductively by means of the following grammar, where  $p \in \text{AP}$ ,  $\alpha \in \text{Ag}$ ,  $x, x_1, \dots, x_n \in \text{Vr}$  such that  $x_i \neq x_j$  for  $i \neq j$ , and  $g, n \in \mathbb{N}$ :

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X} \varphi \mid \varphi \mathbf{U} \varphi \mid \langle\langle x_1, \dots, x_n \rangle\rangle^{\geq g} \varphi \mid (\alpha, x) \varphi.$$

**Notation.** For the rest of the paper, whenever we write  $\langle\langle x_1, \dots, x_n \rangle\rangle^{\geq g}$  we also mean that  $x_i \neq x_j$  for  $i \neq j$ .

Shorthands are derived as usual. Specifically,  $\mathbf{true} \triangleq p \vee \neg p$ ,  $\mathbf{false} \triangleq \neg \mathbf{true}$ ,  $\mathbf{F} \varphi \triangleq \mathbf{true} \mathbf{U} \varphi$ , and  $\mathbf{G} \varphi \triangleq \neg \mathbf{F} \neg \varphi$ . Moreover, the graded universal operator corresponds to the dual of the existential one, i.e.,  $\llbracket x_1, \dots, x_n \rrbracket^{< g} \varphi \triangleq \neg \langle\langle x_1, \dots, x_n \rangle\rangle^{\geq g} \neg \varphi$ . A *placeholder* refers to an agent or a variable. In order to define the semantics, we first define the concept of free placeholders in a formula. Intuitively, an agent or variable is free in  $\varphi$  if it does not have a strategy associated with it (either by quantification or binding) but one is required in order to ascertain if  $\varphi$  is true or not. The definition mimics that for SL [43]. It is included here both for completeness and because it is important for defining the model-checking procedure, in particular for the encoding of strategies as trees (Definition 3.2).

**DEFINITION 2.3.** The set of free agents and free variables of a GRADED-SL formula  $\varphi$  is given by the function  $\text{free} : \text{GRADED-SL} \rightarrow 2^{\text{Ag} \cup \text{Vr}}$  defined as follows:

- $\text{free}(p) \triangleq \emptyset$ , where  $p \in \text{AP}$ ;
- $\text{free}(\neg\varphi) \triangleq \text{free}(\varphi)$ ;
- $\text{free}(\varphi_1 \vee \varphi_2) \triangleq \text{free}(\varphi_1) \cup \text{free}(\varphi_2)$ ;
- $\text{free}(\mathbf{X} \varphi) \triangleq \text{Ag} \cup \text{free}(\varphi)$ ;
- $\text{free}(\varphi_1 \mathbf{U} \varphi_2) \triangleq \text{Ag} \cup \text{free}(\varphi_1) \cup \text{free}(\varphi_2)$ ;
- $\text{free}(\langle\langle x_1, \dots, x_n \rangle\rangle^{\geq g} \varphi) \triangleq \text{free}(\varphi) \setminus \{x_1, \dots, x_n\}$ ;
- $\text{free}((\alpha, x) \varphi) \triangleq \text{free}(\varphi)$ , if  $\alpha \notin \text{free}(\varphi)$ , where  $\alpha \in \text{Ag}$  and  $x \in \text{Vr}$ ;
- $\text{free}((\alpha, x) \varphi) \triangleq (\text{free}(\varphi) \setminus \{\alpha\}) \cup \{x\}$ , if  $\alpha \in \text{free}(\varphi)$ , where  $\alpha \in \text{Ag}$  and  $x \in \text{Vr}$ .

A formula  $\varphi$  without free agents (resp., variables), i.e., with  $\text{free}(\varphi) \cap \text{Ag} = \emptyset$  (resp.,  $\text{free}(\varphi) \cap \text{Vr} = \emptyset$ ), is called agent-closed (resp., variable-closed). If  $\varphi$  is both agent- and variable-closed, it is called a sentence.

Another important concept that characterizes the syntax of GRADED-SL is the *alternation number* of quantifiers, i.e., the maximum number of quantifier switches  $\langle\langle \cdot \rangle\rangle \llbracket \cdot \rrbracket$ ,  $\llbracket \cdot \rrbracket \langle\langle \cdot \rangle\rangle$ ,  $\langle\langle \cdot \rangle\rangle \neg \langle\langle \cdot \rangle\rangle$ , or  $\llbracket \cdot \rrbracket \neg \llbracket \cdot \rrbracket$  that binds a tuple of variables in a subformula that is not a sentence. We denote by  $\text{alt}(\varphi)$  the alternation number of a GRADED-SL formula  $\varphi$ . The *quantifier rank* of  $\varphi$  is the maximum nesting of quantifiers in  $\varphi$ , e.g.,  $\langle\langle x_1, \dots, x_n \rangle\rangle^{\geq g} (\alpha_1, x_1) \dots (\alpha_n, x_n) \wedge_{i=1}^n (\langle\langle y \rangle\rangle (\alpha_i, y) \psi_i) \rightarrow \psi_i$  has quantifier rank 2 if each  $\psi_i$  is quantifier free. Moreover, a *quantifier-block* of  $\varphi$  is a maximally-consecutive sequence of quantifiers in  $\varphi$  of the same type (i.e., either all existential, or all universal). The *quantifier-block rank* of  $\varphi$  is exactly like the quantifier rank except that a quantifier block of  $j$  quantifiers contributes 1 instead of  $j$  to the count.

SL has a few natural syntactic fragments, the most powerful of which is called Nested-Goal SL. Similarly, we define GRADED-SL *Nested-Goal* (abbreviated GRADED-SL[NG]), as a syntactic fragment of GRADED-SL. As in Nested-Goal SL, in GRADED-SL[NG] we require that bindings and quantifications appear in exhaustive blocks. I.e., whenever there is a quantification over a variable in a formula  $\psi$  it is part of a consecutive sequence of quantifiers that covers all of the free variables that appear in  $\psi$ , and whenever an agent is bound to a strategy then it is part of a consecutive sequence of bindings of all agents to strategies. Finally, formulas with free agents are not allowed. To formalize GRADED-SL[NG] we first introduce some notions. A *quantification prefix* over a finite set  $V \subseteq \text{Vr}$  of variables is a sequence  $\wp \in \{\langle\langle x_1, \dots, x_n \rangle\rangle^{\geq g}, \llbracket x_1, \dots, x_n \rrbracket^{< g} : x_1, \dots, x_n \in V \wedge g \in \mathbb{N}\}^*$  such that each  $x \in V$  occurs exactly once in  $\wp$ . A *binding prefix* is a sequence  $b \in \{(a, x) : \alpha \in \text{Ag} \wedge x \in \text{Vr}\}^{|\text{Ag}|}$  such that each  $\alpha \in \text{Ag}$  occurs exactly once in  $b$ . We denote the set of binding prefixes by  $\text{Bn}$ , and the set of quantification prefixes over  $V$  by  $\text{Qn}(V)$ .

**DEFINITION 2.4.** GRADED-SL[NG] formulas are built inductively using the following grammar, with  $p \in \text{AP}$ ,  $\wp \in \text{Qn}(V)$  ( $V \subseteq \text{Vr}$ ), and  $b \in \text{Bn}$ :

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X} \varphi \mid \varphi \mathbf{U} \varphi \mid \wp \varphi \mid b \varphi,$$

where in the rule  $\wp \varphi$  we require that  $\varphi$  is agent-closed and  $\wp \in \text{Qn}(\text{free}(\varphi))$ .

We conclude this subsection by introducing GRADED-SL[1c], another graded extension of fragment of SL, namely the one-goal sub-logic. As the name says, this fragment is obtained by restricting GRADED-SL[NG] to encompass formulas with just one nested goal. The importance of this fragment in SL stems from the fact that it strictly includes ATL\* while maintaining the same complexity for both the model checking and the satisfiability problems, i.e. 2EXPTIME-COMPLETE [42, 43]. However, it is commonly believed that Nash Equilibrium cannot be expressed in this fragment. The definition of GRADED-SL[1c] follows.

**DEFINITION 2.5.** GRADED-SL[1c] formulas are built inductively using the following grammar, with  $p \in \text{AP}$ ,  $\wp \in \text{Qn}(V)$  ( $V \subseteq \text{Vr}$ ), and  $b \in \text{Bn}$ :

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X} \varphi \mid \varphi \mathbf{U} \varphi \mid \wp b \varphi,$$

with  $\wp$  quantification prefix over  $\text{free}(b\varphi)$ .

### 2.3 Semantics

As for SL, the interpretation of a GRADED-SL formula requires a valuation of its free placeholders. This is done via *assignments* (*from*  $s$ ), i.e., functions  $\chi \in \text{Asg}(s) \triangleq (\text{Vr} \cup \text{Ag}) \rightarrow \text{Str}(s)$  mapping variables/agents to strategies. We denote by  $\chi[e \mapsto \sigma]$ , with  $e \in \text{Vr} \cup \text{Ag}$  and  $\sigma \in \text{Str}(s)$ , the assignment that differs from  $\chi$  only in the fact that  $e$  maps to  $\sigma$ . Extend this definition to tuples: for  $\bar{e} = (e_1, \dots, e_n)$  with  $e_i \neq e_j$  for  $i \neq j$ , define  $\chi[\bar{e} \mapsto \bar{\sigma}]$  to be the assignment that differs from  $\chi$  only in the fact that  $e_i$  maps to  $\sigma_i$  (for each  $i$ ).

Since assignments are total functions, each assignment from  $s$  determines a unique path from  $s$ , called a play:

**DEFINITION 2.6.** For an assignment  $\chi \in \text{Asg}(s)$  the  $(\chi, s)$ -play denotes the path  $\pi \in \text{Pth}(s)$  such that for all  $i \in \mathbb{N}$ , it holds that  $\pi_{i+1} = \text{tr}(\text{dc})(\pi_i)$ , where  $\text{dc}(\alpha) \triangleq \chi(\alpha)(\pi_{\leq i})$  for  $\alpha \in \text{Ag}$ . The function  $\text{play} : \text{Asg} \times \text{St} \rightarrow \text{Pth}$ , with  $\text{dom}(\text{play}) \triangleq \{(\chi, s) : \chi \in \text{Asg}(s)\}$ , maps  $(\chi, s)$  to the  $(\chi, s)$ -play  $\text{play}(\chi, s) \in \text{Pth}(s)$ .

The notation  $\pi_{\leq i}$  (resp.  $\pi_{< i}$ ) denotes the prefix of the sequence  $\pi$  of length  $i$  (resp.  $i - 1$ ). Similarly, the notation  $\pi_i$  denotes the  $i$ th symbol of  $\pi$ . Thus,  $\text{play}(\chi, s)_i$  is the  $i$ th state on the play determined by  $\chi$  from  $s$ .

The following definition of  $\chi_i$  says how to interpret an assignment  $\chi$  starting from a point  $i$  along the play, i.e., for each placeholder  $e$ , take the action the strategy  $\chi(e)$  would do if it were given the prefix of the play up to  $i$  followed by the current history.

**DEFINITION 2.7.** For  $\chi \in \text{Asg}(s)$  and  $i \in \mathbb{N}$ , writing  $\rho \triangleq \text{play}(\chi, s)_{\leq i}$  (the prefix of the play up to  $i$ ) and  $t \triangleq \text{play}(\chi, s)_i$  (the last state of  $\rho$ ) define  $\chi_i \in \text{Asg}(t)$  to be the assignment from  $t$  that maps  $e \in \text{Vr} \cup \text{Ag}$  to the strategy that maps  $h \in \text{Hst}(t)$  to the action  $\chi(e)(\rho_{< i} \cdot h)$ .

We now define the semantics of GRADED-SL. In particular, we define  $\mathcal{G}, \chi, s \models \varphi$  and say that  $\varphi$  holds at  $s$  in  $\mathcal{G}$  under  $\chi$ .

**DEFINITION 2.8.** Fix a CGS  $\mathcal{G}$ . For all states  $s \in \text{St}$  and assignments  $\chi \in \text{Asg}(s)$ , the relation  $\mathcal{G}, \chi, s \models \varphi$  is defined inductively on the structure of  $\varphi$ :

- $\mathcal{G}, \chi, s \models p$  iff  $p \in \text{ap}(s)$ ;
- $\mathcal{G}, \chi, s \models \neg \varphi$  iff  $\mathcal{G}, \chi, s \not\models \varphi$ ;
- $\mathcal{G}, \chi, s \models \varphi_1 \vee \varphi_2$  iff  $\mathcal{G}, \chi, s \models \varphi_1$  or  $\mathcal{G}, \chi, s \models \varphi_2$ ;
- $\mathcal{G}, \chi, s \models X \varphi$  iff  $\mathcal{G}, \chi_1, \text{play}(\chi, s)_1 \models \varphi$ ;
- $\mathcal{G}, \chi, s \models \varphi_1 \cup \varphi_2$  iff there is an index  $i \in \mathbb{N}$  such that  $\mathcal{G}, \chi_i, \text{play}(\chi, s)_i \models \varphi_2$  and, for all indexes  $j \in \mathbb{N}$  with  $j < i$ , it holds that  $\mathcal{G}, \chi_j, \text{play}(\chi, s)_j \models \varphi_1$ ;
- $\mathcal{G}, \chi, s \models (\alpha, x)\varphi$  iff  $\mathcal{G}, \chi[\alpha \mapsto \chi(x)], s \models \varphi$ ;
- $\mathcal{G}, \chi, s \models \langle\!\langle x_1, \dots, x_n \rangle\!\rangle^{\geq g} \varphi$  iff there exist  $g$  many tuples  $\bar{\sigma}_1, \dots, \bar{\sigma}_g$  of strategies such that:
  - $\bar{\sigma}_i \neq \bar{\sigma}_j$  for  $i \neq j$ , and
  - $\mathcal{G}, \chi[\bar{x} \mapsto \bar{\sigma}_i], s \models \varphi$  for  $1 \leq i \leq g$ .

Intuitively, the existential quantifier  $\langle\!\langle x_1, \dots, x_n \rangle\!\rangle^{\geq g} \varphi$  allows us to count the number of distinct tuples of strategies that satisfy  $\varphi$ .

As usual, if  $\chi$  and  $\chi'$  agree on  $\text{free}(\varphi)$ , then  $\mathcal{G}, \chi, s \models \varphi$  if and only if  $\mathcal{G}, \chi', s \models \varphi$ , i.e., the truth of  $\varphi$  does not depend on the values the assignment takes on placeholders that are not free. Thus, for a sentence  $\varphi$ , we write  $\mathcal{G} \models \varphi$  to mean that  $\mathcal{G}, \chi, s_I \models \varphi$  for some (equivalently, for all) assignments  $\chi$ , and where  $s_I$  is the initial state of  $\mathcal{G}$ .

**Comparison with other logics.** In the following we give the main intuitions relating GRADED-SL with SL and a *naive* fragment of GRADED-SL in which quantifiers are over single variables (and not tuple of variables).

GRADED-SL extends SL by replacing universal and existential strategy quantifiers  $\langle\!\langle x \rangle\!\rangle$  and  $\llbracket x \rrbracket$  with their graded versions over tuples of variables  $\langle\!\langle x_1, \dots, x_n \rangle\!\rangle^{\geq g}$  and  $\llbracket x_1, \dots, x_n \rrbracket^{\leq g}$ . Grades allow one to count, which is not possible, a priori, in SL. On the other hand, every formula of SL has an equivalent formula of GRADED-SL: formed by replacing every quantifier  $\langle\!\langle x \rangle\!\rangle$  with  $\langle\!\langle x \rangle\!\rangle^{\geq 1}$ .

An important power of GRADED-SL is that it can quantify over tuples of strategy variables. Consider the formula  $\varphi = \langle\!\langle x, y \rangle\!\rangle^{\geq 2}(a, x)(b, y)\psi$  in GRADED-SL that represents the property in which there exist two tuples of strategies that satisfy  $\psi$ , and compare it to the following attempt at expressing  $\varphi$  only quantifying over single strategies:  $\varphi' \triangleq \langle\!\langle x \rangle\!\rangle^{\geq 2}\langle\!\langle y \rangle\!\rangle^{\geq 2}(a, x)(b, y)\psi$ . Observe that  $\varphi'$  says that there are two different strategies  $\sigma_1$  and  $\sigma_2$  for agent  $\alpha_1$ , and for each  $\sigma_i$  there are two different strategies  $\delta_1^i$  and  $\delta_2^i$  for the agent  $\alpha_2$  that satisfy  $\psi$ . Thus, there are four tuples of strategies that satisfy  $\psi$ , i.e.,  $\{\sigma_1, \delta_1^1\}, \{\sigma_1, \delta_2^1\}, \{\sigma_2, \delta_1^2\}, \{\sigma_2, \delta_2^2\}$ . Other attempts (such as  $\langle\!\langle x \rangle\!\rangle^{\geq 2}\langle\!\langle y \rangle\!\rangle^{\geq 1}(a, x)(b, y)\psi$ ), also fail to capture  $\varphi$  as they restrict one of the agents to a single strategy. This demonstrates the inadequacy of quantifying over single strategies for counting solution concepts.

### 2.4 Games with temporal objectives

In game theory, players have objectives that are summarised in a payoff function they receive depending on the resulting play. In order to specify such payoffs we follow a formalisation from [35] called *objective* LTL. This will allow us to model the Prisoner’s Dilemma (PD), probably the most famous game in game-theory, as well as an iterated version (IPD). We then discuss appropriate solution concepts, and show how to express these in GRADED-SL.

Let  $\mathcal{G}$  be a CGS with  $n$  agents. Let  $m \in \mathbb{N}$  and fix, for each agent  $\alpha_i \in \text{Ag}$ , an *objective* tuple  $S_i \triangleq \langle f_i, \varphi_i^1, \dots, \varphi_i^m \rangle$ , where  $f_i : \{0, 1\}^m \rightarrow \mathbb{N}$ , and each  $\varphi_i^j$  is an LTL formula over AP. If  $\pi$  is a play, then agent  $\alpha_i$  receives payoff  $f_i(\bar{h}) \in \mathbb{N}$  where the  $j$ ’th bit  $\bar{h}_j$  of  $\bar{h}$  is 1 if and only if  $\pi \models \varphi_i^j$ . For instance,  $f$  may count the number of formulas that are true.

#### Prisoner’s Dilemma – One shot and Iterated

Two people have been arrested for robbing a bank and placed in separate isolation cells. Each has two possible choices, remaining silent or confessing. If a robber confesses and the other remains silent, the former is released and the latter stays in prison for a long time. If both confess they get two convictions, but they will get early parole. If both remain silent, they get a lighter sentence (e.g., on firearms possession charges). The dilemma faced by the prisoners is that, whatever the choice of the other prisoner, each is better off confessing than remaining silent. But the result obtained when both confess is worse than if they both remain silent.

We describe this (one-shot) scenario with the CGS in Figure 1 and, for agent  $\alpha_i$ , the objective  $S_i \triangleq \langle f_i, \varphi_i^1, \varphi_i^2, \varphi_i^3, \varphi_i^4 \rangle$

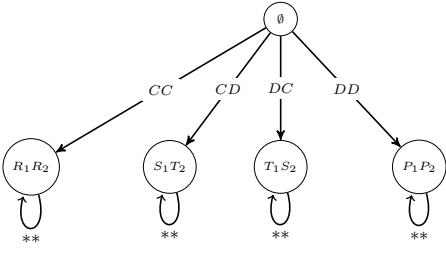


Figure 1: Prisoner’s dilemma.

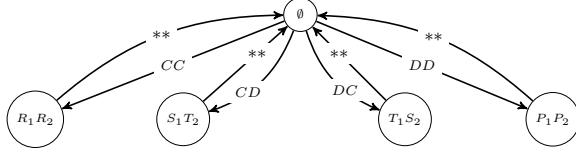


Figure 2: Iterated Prisoner’s dilemma.

where  $\varphi_i^1 \triangleq \mathbf{X} \mathbf{S}_i$ ,  $\varphi_i^2 \triangleq \mathbf{X} \mathbf{P}_i$ ,  $\varphi_i^3 \triangleq \mathbf{X} \mathbf{R}_i$ , and  $\varphi_i^4 \triangleq \mathbf{X} \mathbf{T}_i$  and  $f_i$  returns the value of its input vector interpreted as a binary number, e.g.,  $f_i(0100) = 4$ .

In words, we have two agents  $\alpha_1$  and  $\alpha_2$ . Each agent has two actions: cooperate ( $C$ ) and defect ( $D$ ), corresponding, respectively, to the options of remaining silent or confessing. For each possible pair of moves, the game goes in a state whose atomic propositions represent the payoffs:  $\mathbf{R}_i$  represents the *reward* payoff that  $\alpha_i$  receives if both cooperate;  $\mathbf{P}_i$  is the *punishment* that  $\alpha_i$  receives if both defect;  $\mathbf{T}_i$  is the *temptation* that  $\alpha_i$  receives as a sole defector, and  $\mathbf{S}_i$  is the *sucker* payoff that  $\alpha_i$  receives as a sole cooperator. The payoffs satisfy the following chain of inequalities:  $\mathbf{T}_i > \mathbf{R}_i > \mathbf{P}_i > \mathbf{S}_i$ .

The Iterated Prisoner’s Dilemma (IPD) is used to model a series of interactions. This is like PD except that after the first choice, both agents have another choice and so on. This is modeled in Figure 2. The main difference between the one-shot and iterated PD is that in the latter the agents’ actions (may) depend on the past behaviour. Convenient payoff functions for the IPD are the mean-average and discounted-payoff [11]. Such quantitative payoffs can be replaced, in a first approximation, by LTL payoffs such as “maximise the largest payoff I receive infinitely often”. This is formalised, for agent  $\alpha_i$ , by the objective  $S_i \triangleq \langle f_i, \varphi_i^1, \varphi_i^2, \varphi_i^3, \varphi_i^4 \rangle$  where  $\varphi_i^1 \triangleq \mathbf{G} \mathbf{F} \mathbf{S}_i$ ,  $\varphi_i^2 \triangleq \mathbf{G} \mathbf{F} \mathbf{P}_i$ ,  $\varphi_i^3 \triangleq \mathbf{G} \mathbf{F} \mathbf{R}_i$ , and  $\varphi_i^4 \triangleq \mathbf{G} \mathbf{F} \mathbf{T}_i$ , and  $f_i$  is as in the one-shot PD.

**Solution Concepts.** Solution concepts are criteria by which one captures what rational agents would do. This is especially relevant in case each agent has its own objective. The central solution concept in game theory is the Nash Equilibrium.

A tuple of strategies, one for each player, is called a *strategy profile*. A strategy profile is a *Nash equilibrium (NE)* if no agent can increase his payoff by unilaterally choosing a different strategy. A game may have zero, one, or many NE.

Consider first a CGS  $\mathcal{G}$  with  $n$  agents, where the objective of the agent  $\alpha_i \in \text{Ag}$  contains a single LTL formula  $\varphi_i$  (with a larger payoff if it holds than if it doesn’t). It is not hard to see that the following formula of SL expresses that

$\bar{x} \triangleq (x_1 \dots x_n)$  is a Nash Equilibrium:

$$\psi_{NE}^1(\bar{x}) \triangleq (\alpha_1, x_1) \dots (\alpha_n, x_n) \bigwedge_{i=1}^n (\langle\langle y \rangle\rangle(\alpha_i, y) \varphi_i) \rightarrow \varphi_i$$

An alternative (which we will later use to obtain a formula in the fragment GRADED-SL[NC] that expresses the existence of a unique NE) is the following:

$$\phi_{NE}^1(\bar{x}) \triangleq [\![y_1]\!] \dots [\![y_n]\!] \bigwedge_{i=1}^n (b_i \varphi_i) \rightarrow b \varphi_i$$

where  $b = (\alpha_1, x_1) \dots (\alpha_n, x_n)$ , and  $b_i = (\alpha_1, x_1) \dots (\alpha_{i-1}, x_{i-1})(\alpha_i, y_i)(\alpha_{i+1}, x_{i+1}) \dots (\alpha_n, x_n)$ .

Consider now the general case, where each agent  $\alpha_i$  has an objective tuple  $S_i \triangleq \langle f_i, \varphi_i^1, \dots, \varphi_i^m \rangle$ . Given a vector  $\bar{h} \in \{0, 1\}^m$ , let  $gd_i(\bar{h}) \triangleq \{\bar{t} \in \{0, 1\}^m \mid f_i(\bar{t}) \geq f_i(\bar{h})\}$  be the set of vectors  $\bar{t}$  for which the payoff for agent  $\alpha_i$  is not worse than for  $\bar{h}$ . Also, let  $\eta_i^{\bar{h}}$  be the formula obtained by taking a conjunction of the formulas  $\varphi_i^1, \dots, \varphi_i^m$  or their negations according to  $\bar{h}$ , i.e., by taking  $\varphi_i^j$  if the  $j$ ’th bit in  $\bar{h}$  is 1, and otherwise taking  $\neg \varphi_i^j$ . Formally,  $\eta_i^{\bar{h}} \triangleq \bigwedge_{j \in \{1 \leq j \leq m \mid h_j = 1\}} \varphi_i^j \wedge \bigwedge_{j \in \{1 \leq j \leq m \mid h_j = 0\}} \neg \varphi_i^j$ . Observe that the following formula says that  $\bar{x} \triangleq (x_1 \dots x_n)$  is a Nash Equilibrium:

$$\psi_{NE}(\bar{x}) \triangleq (\alpha_1, x_1) \dots (\alpha_n, x_n)$$

$$\bigwedge_{i=1}^n \bigwedge_{\bar{h} \in \{0, 1\}^m} (\langle\langle y \rangle\rangle(\alpha_i, y) \eta_i^{\bar{h}}) \rightarrow \bigvee_{\bar{t} \in gd_i(\bar{h})} \eta_i^{\bar{t}}$$

Alike, one can modify  $\phi_{NE}^1(\bar{x})$  to obtain a similar formula  $\phi_{NE}(\bar{x})$  expressing that  $\bar{x} \triangleq (x_1 \dots x_n)$  is a Nash Equilibrium:

$$\phi_{NE}(\bar{x}) \triangleq [\![y_1]\!] \dots [\![y_n]\!]$$

$$\bigwedge_{i=1}^n \bigwedge_{\bar{h} \in \{0, 1\}^m} (b_i \eta_i^{\bar{h}}) \rightarrow \bigvee_{\bar{t} \in gd_i(\bar{h})} b \eta_i^{\bar{t}}$$

Going back to the PD example, due to the simplicity of the payoff functions, the formula  $\psi_{NE}$  collapses to become:

$$\psi_{PD}(\bar{x}) \triangleq (\alpha_1, x_1) \dots (\alpha_n, x_n) \bigwedge_{i=1}^n \bigwedge_{j=1}^4 (\langle\langle y \rangle\rangle(\alpha_i, y) \varphi_i^j) \Rightarrow (\bigvee_{r \geq j} \varphi_i^r)$$

As it turns out (again due to the simplicity of the payoff functions), the formula above is also correct for the IPD.

It has been argued (in [35, 55]) that NE may be implausible when used for sequential games (of which iterated one shot games are central examples), and that a more robust notion is subgame-perfect equilibrium [52]. Given a game  $\mathcal{G}$ , a strategy profile is a *subgame-perfect equilibrium (SPE)* if for every possible history of the game, the strategies are an NE. The following formula expresses that  $\bar{x} \triangleq (\alpha_1, x_1) \dots (\alpha_n, x_n)$  is an SPE:

$$\phi_{SPE}(\bar{x}) \triangleq [\![z_1, \dots, z_n]\!](\alpha_1, z_1) \dots (\alpha_n, z_n) \mathsf{G} \phi_{NE}(\bar{x})$$

Using graded modalities, we can thus express the uniqueness of a NE using the following GRADED-SL formula:

$$\langle\langle x_1, \dots, x_n \rangle\rangle^{\geq 1} \psi_{NE}(\bar{x}) \wedge \neg \langle\langle x_1, \dots, x_n \rangle\rangle^{\geq 2} \psi_{NE}(\bar{x})$$

By replacing  $\psi_{NE}$  with  $\phi_{NE}$  (resp. by  $\phi_{SPE}$ ) in the formula above, we can express the uniqueness of a NE (resp. SPE) in GRADED-SL[NC].

### 3. THE MODEL-CHECKING PROCEDURE

In this section we study the model-checking problem for GRADED-SL and show that it is decidable with a time-complexity that is non-elementary (i.e., not bounded by any fixed tower of exponentials). However, it is elementary if the number of blocks of quantifiers is fixed. For the algorithmic procedures, we follow an *automata-theoretic approach* [37], reducing the decision problem for the logic to the emptiness problem of an automaton. The procedure we propose here extends that used for SL in [43]. The only case that is different is the new graded quantifier over tuples of strategies.

We start with the central notions of automata theory, and then show how to convert a GRADED-SL sentence  $\varphi$  into an automaton that accepts exactly the (tree encodings) of the concurrent game structures that satisfy  $\varphi$ . This is used to prove the main result about GRADED-SL model checking.

#### 3.1 Automata Theory

A  $\Sigma$ -labeled  $\Upsilon$ -tree  $\mathsf{T}$  is a pair  $\langle T, V \rangle$  where  $T \subseteq \Upsilon^+$  is prefix-closed (i.e., if  $t \in T$  and  $s \in \Upsilon^+$  is a prefix of  $t$  then also  $s \in T$ ), and  $V : T \rightarrow \Sigma$  is a labeling function. Note that every word  $w \in \Upsilon^+ \cup \Upsilon^\omega$  with the property that every prefix of  $w$  is in  $T$ , can be thought of as a path in  $\mathsf{T}$ . Infinite paths are called *branches*.

*Nondeterministic tree automata* (NTA) are a generalization to infinite trees of the classical automata on words [54]. *Alternating tree automata* (ATA) are a further generalization of nondeterministic tree automata [23]. Intuitively, on visiting a node of the input tree, while an NTA sends exactly one copy of itself to each of the successors of the node, an ATA can send several copies to the same successor. We use the parity acceptance condition [37].

For a set  $X$ , let  $\mathbf{B}^+(X)$  be the set of positive Boolean formulas over  $X$ , including the constants **true** and **false**. A set  $Y \subseteq X$  satisfies a formula  $\theta \in \mathbf{B}^+(X)$ , written  $Y \models \theta$ , if assigning **true** to elements in  $Y$  and **false** to elements in  $X \setminus Y$  makes  $\theta$  true.

**DEFINITION 3.1.** An Alternating Parity Tree-Automaton (APT) is a tuple  $\mathcal{A} \triangleq \langle \Sigma, \Delta, Q, \delta, q_0, \aleph \rangle$ , where  $\Sigma$  is the input alphabet,  $\Delta$  is a set of directions,  $Q$  is a finite set of states,  $q_0 \in Q$  is an initial state,  $\delta : Q \times \Sigma \rightarrow \mathbf{B}^+(\Delta \times Q)$  is an alternating transition function, and  $\aleph$ , an acceptance condition, is of the form  $(F_1, \dots, F_k) \in (2^Q)^+$  where  $F_1 \subseteq F_2 \dots \subseteq F_k = Q$ .

The set  $\Delta \times Q$  is called the set of *moves*. An NTA is an ATA in which each conjunction in the transition function  $\delta$  has exactly one move  $(d, q)$  associated with each direction  $d$ .

An *input tree* for an APT is a  $\Sigma$ -labeled  $\Delta$ -tree  $\mathsf{T} = \langle T, v \rangle$ . A *run* of an APT on an input tree  $\mathsf{T} = \langle T, v \rangle$  is a  $(\Delta \times Q)$ -tree  $R$  such that, for all nodes  $x \in R$ , where  $x = (d_1, q_1) \dots (d_n, q_n)$  (for some  $n \in \mathbb{N}$ ), it holds that (i)  $y \triangleq (d_1, \dots, d_n) \in T$  and (ii) there is a set of moves  $S \subseteq \Delta \times Q$  with  $S \models \delta(q_n, v(y))$  such that  $x \cdot (d, q) \in R$  for all  $(d, q) \in S$ .

The acceptance condition allows us to say when a run is successful. Let  $R$  be a run of an APT  $\mathcal{A}$  on an input tree  $\mathsf{T}$  and  $u \in (\Delta \times Q)^\omega$  one of its branches. Let  $\text{inf}(u) \subseteq Q$  denote the set of states that occur in infinitely many moves of  $u$ . Say that  $u$  satisfies the parity acceptance condition  $\aleph = (F_1, \dots, F_k)$  if the least index  $i \in [1, k]$  for which  $\text{inf}(u) \cap F_i \neq \emptyset$  is even. An APT *accepts* an input tree  $\mathsf{T}$  iff there exists a run  $R$  of  $\mathcal{A}$  on  $\mathsf{T}$  such that all its branches satisfy the acceptance condition

$\aleph$ . The *language*  $L(\mathcal{A})$  of the APT  $\mathcal{A}$  is the set of trees  $\mathsf{T}$  accepted by  $\mathcal{A}$ . Two automata are *equivalent* if they have the same language. The *emptiness problem* for alternating parity tree-automata is to decide, given  $\mathcal{A}$ , whether  $L(\mathcal{A}) = \emptyset$ . The *universality problem* is to decide whether  $\mathcal{A}$  accepts all trees.

#### 3.2 From Logic to Automata

Following an automata-theoretic approach, we reduce the model-checking problem of GRADED-SL to the emptiness problem for alternating parity tree automata [43]. The main step is to translate every GRADED-SL formula  $\varphi$  (i.e.,  $\varphi$  may have free placeholders), concurrent-game structure  $\mathcal{G}$ , and state  $s$ , into an APT that accepts a tree if and only if the tree encodes an assignment  $\chi$  such that  $\mathcal{G}, \chi, s \models \varphi$ .

We first describe the encoding, following [43]. Informally, the CGS  $\mathcal{G}$  is encoded by its “tree-unwinding starting from  $s$ ” whose nodes represent histories, i.e., the St-labeled St-tree  $\mathsf{T} \triangleq \langle \text{Hst}(s), u \rangle$  such that  $u(h)$  is the last symbol of  $h$ . Then, every strategy  $\chi(e)$  with  $e \in \text{free}(\varphi)$  is encoded as an Ac-labelled tree over the unwinding. The unwinding and these strategies  $\chi(e)$  are viewed as a single  $(\text{VAL} \times \text{St})$ -labeled tree where  $\text{VAL} \triangleq \text{free}(\varphi) \rightarrow \text{Ac}$ .

**DEFINITION 3.2.** The encoding of  $\chi$  (w.r.t.  $\varphi, \mathcal{G}, s$ ) is the  $(\text{VAL} \times \text{St})$ -labeled St-tree  $\mathsf{T} \triangleq \langle T, u \rangle$  such that  $T$  is the set of histories  $h$  of  $\mathcal{G}$  starting with  $s$  and  $u(h) \triangleq (f, q)$  where  $q$  is the last symbol in  $h$  and  $f : \text{free}(\varphi) \rightarrow \text{Ac}$  is defined by  $f(e) \triangleq \chi(e)(h)$  for all  $e \in \text{free}(\varphi)$ .<sup>2</sup>

**LEMMA 3.1.** For every GRADED-SL formula  $\varphi$ , CGS  $\mathcal{G}$ , and state  $s \in \text{St}$ , there exists an APT  $\mathcal{A}_\varphi$  such that for all assignments  $\chi$ , if  $\mathsf{T}$  is the encoding of  $\chi$  (w.r.t.  $\varphi, \mathcal{G}, s$ ), then  $\mathcal{G}, \chi, s \models \varphi$  iff  $\mathsf{T} \in L(\mathcal{A}_\varphi)$ .

**PROOF.** As in [43] we induct on the structure of the formula  $\varphi$  to construct the corresponding automaton  $\mathcal{A}_\varphi$ . The Boolean operations are easily dealt with using the fact that disjunction corresponds to non-determinism, and negation corresponds to dualising the automaton. Note ( $\dagger$ ) that thus also conjunction is dealt with due to De Morgan’s laws. The temporal operators are dealt with by following the unique play (determined by the given assignment) and verifying the required subformulas, e.g., for  $\mathbf{X}\psi$  the automaton, after taking one step along the play, launches a copy of the automaton for  $\psi$ . All of these operations incur a linear blowup in the size of the automaton. The only case that differs from [43] is the quantification, i.e., we need to handle the case that  $\varphi = \langle \langle x_1, \dots, x_n \rangle \rangle^{\geq g} \psi$ . Recall that  $\mathcal{G}, \chi, s \models \langle \langle x_1, \dots, x_n \rangle \rangle^{\geq g} \psi$  iff there exists  $g$  many tuples  $\overline{x_1}, \dots, \overline{x_g}$  of strategies such that:  $\overline{x_a} \neq \overline{x_b}$  for  $a \neq b$ , and  $\mathcal{G}, \chi[\overline{x} \mapsto \overline{x_i}], s \models \psi$  for  $1 \leq i \leq g$ . We show how to build an NPT for  $\varphi$  that mimics this definition: it will be a projection of an APT  $\mathcal{D}_\psi$ , which itself is the intersection of two automata, one checking that each of the  $g$  tuples of strategies satisfies  $\psi$ , and the other checking that each pair of the  $g$  tuples of strategies is distinct.

In more detail, introduce a set of fresh variables  $X \triangleq \{x_i^j \in \text{Vr} : i \leq n, j \leq g\}$ , and consider the formulas  $\psi^j$  (for  $j \leq g$ ) formed from  $\psi$  by renaming  $x_i$  (for  $i \leq n$ ) to  $x_i^j$ . Define  $\psi' \triangleq \bigwedge_{j \leq g} \psi^j$ . Note that, by induction, each  $\psi^j$  has a corresponding APT, and thus, using the conjunction-case

<sup>2</sup>In case  $\text{free}(\varphi) = \emptyset$ , then  $f$  is the (unique) empty function. In this case, the encoding of every  $\chi$  may be viewed as the tree-unwinding from  $s$ .

(†) above, there is an APT  $\mathcal{B}$  for  $\psi'$ . Note that the input alphabet for  $\mathcal{B}$  is  $(\text{free}(\psi') \rightarrow \text{Ac}) \times \text{St}$  and that  $X \subseteq \text{free}(\psi')$ .

On the other hand, let  $\mathcal{C}$  be an APT with input alphabet  $(\text{free}(\psi') \rightarrow \text{Ac}) \times \text{St}$  that accepts a tree  $T = \langle T, v \rangle$  if and only if for every  $a \neq b \leq g$  there exists  $i \leq n$  and  $h \in T$  such that  $v(h) = (f, q)$  and  $f(x_i^a) \neq f(x_i^b)$ .

Form the APT  $\mathcal{D}_\psi$  for the intersection of  $\mathcal{B}$  and  $\mathcal{C}$ .

Now, using the classic transformation [46], we remove alternation from the APT  $\mathcal{D}_\psi$  to get an equivalent NPT  $\mathcal{N}$  (note that this step costs an exponential). Finally, use the fact that NPTs are closed under projection (with no blowup) to get an NPT for the language  $\text{proj}_X(L(\mathcal{N}))$  of trees that encode assignments  $\chi$  satisfying  $\varphi$ .

For completeness we recall this last step. If  $L$  is a language of  $\Sigma$ -labeled trees with  $\Sigma \triangleq A \rightarrow B$ , and  $X \subset A$ , then the  $X$ -projection of  $L$ , written  $\text{proj}_X(L)$ , is the language of  $\Sigma'$ -labeled trees with  $\Sigma' \triangleq A \setminus X \rightarrow B$  such that  $T \triangleq \langle T, v \rangle \in \text{proj}_X(L)$  if and only if there exists an  $X$ -labeled tree  $\langle T, w \rangle$  such that the language  $L$  contains the tree  $\langle T, u \rangle$  where  $u : T \rightarrow (A \rightarrow B)$  maps  $t \in T$  to  $v(t) \cup w(t)$ . Now, if  $\mathcal{N}$  is an NPT with input alphabet  $\Sigma \triangleq A \rightarrow B$ , and if  $X \subset A$ , then there is an NPT with input alphabet  $\Sigma' \triangleq A \setminus X \rightarrow B$  with language  $\text{proj}_X(L(\mathcal{N}))$ .

The proof that the construction is correct is immediate.  $\square$

We make some remarks about the proof. First, all the cases in the induction incur a linear blowup except for the quantification case (recall that the translation from an APT to an NPT results in an exponentially larger automaton [37]). Thus, the size of the APT for  $\varphi$  is non-elementary in the quantifier-rank of  $\varphi$ . However, we can say a little more. Note that a block of  $k$  identical quantifiers only costs, in the worst case, a single exponential (and not  $k$  many exponentials) because we can extend the proof above to deal with a block of quantifiers at once. Thus, we get that the size of the APT for  $\varphi$  is non-elementary in the quantifier-block rank of  $\varphi$ .

Here is the main decidability result.

**THEOREM 3.1.** *The model-checking problem for GRADED-SL is PTIME-COMPLETE w.r.t. the size of the model and  $(k+1)\text{EXPTIME}$  if  $k \geq 1$  is the quantifier-block rank of  $\varphi$ . Moreover, if  $\varphi$  is the form  $\wp\psi$ , where  $\wp$  is a quantifier-block, and  $\psi$  is of quantifier-block rank  $k-1$ , then the complexity is  $k\text{EXPTIME}$ .*

**PROOF.** The lower-bound w.r.t the size of the model already holds for SL [43]. For the upper bound, use Lemma 3.1 to transform the CGS and  $\varphi$  into an APT and test its emptiness. The complexity of checking emptiness (or indeed, universality) of an APT is in EXPTIME [37]. As discussed after the proof of the Lemma, the size of the APT is a tower of exponentials whose height is the quantifier-block rank of  $\varphi$ . This gives the  $(k+1)\text{EXPTIME}$  upper bound.

Moreover, suppose that  $\varphi = \wp\psi$  where  $\wp$  consists of, say,  $n$  existential quantifiers (resp. universal quantifiers). The quantifier-block rank of  $\psi$  is  $k-1$ . Moreover, in the proof of Lemma 3.1, the APT  $\mathcal{D}_\psi$ , whose size is non-elementary in  $k-1$ , has the property that it is non-empty (resp. universal) if and only if the CGS satisfies  $\wp\psi$ . Conclude that model checking  $\wp\psi$  can be done in  $k\text{EXPTIME}$ .  $\square$

**THEOREM 3.2.** *The model-checking problem for GRADED-SL<sub>[NG]</sub> is PTIME-COMPLETE w.r.t. the size of the model and  $(k+1)\text{-EXPTIME}$  when restricted to formulas of maximum alternation number  $k$ .*

**PROOF.** The lower bound already holds for SL<sub>[NG]</sub> [43], and the upper bound is obtained by following the same reasoning for SL<sub>[NG]</sub> of the singleton existential quantifier [43] but using the automaton construction as in Theorem 3.1.  $\square$

Directly from the statements reported above, we get the following results:

**THEOREM 3.3.** *Checking the uniqueness of NE, and checking the uniqueness of SPE, can be done in 2EXPTIME.*

**PROOF.** For NE: by Section 2.4, we need to check that  $\langle x_1, \dots, x_n \rangle^{\geq 1} \psi_{NE}(\bar{x})$  holds but  $\langle x_1, \dots, x_n \rangle^{\geq 2} \psi_{NE}(\bar{x})$  does not; by the second part of Theorem 3.1, each of these two model-checking problems can be decided in 2EXPTIME.

For SPE: apply Theorem 3.2 and use the fact that the formula for SPE in Section 2.4 is in GRADED-SL Nested-Goal and has alternation number 1.  $\square$

We conclude this section with the complexity of the model checking problem for GRADED-SL[1c]. Also in this case one can derive the lower bound from the one holding for the corresponding sub-logic in SL (SL[1c]) and the upper bound by using the same algorithm for SL[1c] but plugging a (yet no more complex) different automata construction for the new existential quantifier modality. Indeed the model checking problem for GRADED-SL[1c] is 2EXPTIME-COMPLETE. It is worth recalling that SL[1c] strictly subsumes ATL\* [43]. It is quite immediate to see that this also holds in the graded setting (note that ATL\* already allows quantifying over tuples of agents' (bound) strategies). As the model checking for ATL\* is already 2EXPTIME-HARD, we get that also for the graded extension for this logic, which we name GATL\*, the model checking problem is 2EXPTIME-COMPLETE. The model checking results for both GATL\* and GRADED-SL[1c] are reported in the following theorem.

**THEOREM 3.4.** *The model-checking problem for GATL\* and GRADED-SL[1c] is PTIME-COMPLETE w.r.t. the size of the model and 2-EXPTIME-COMPLETE in the size of formula.*

## 4. CONCLUSION

The Nash equilibrium is the foundational solution concept in game theory. The last twenty years have witnessed the introduction of many logical formalisms for modeling and reasoning about solution concepts, and NE in particular [9, 14, 19, 32, 39, 43, 44]. These formalisms are useful for addressing qualitative questions such as “does the game admit a Nash equilibrium?”. Among others, Strategy Logic (SL) has come to the fore as a general formalism that can express and solve this question, for LTL objectives, in 2EXPTIME. Contrast this with the fact that this question is 2EXPTIME-complete even for two player zero-sum LTL games [4].

One of the most important questions about NE in computational game theory is “does the game admit more than one NE?” [20, 48] — the unique NE problem. This problem is deeply investigated in game theory and is shown to be very challenging [2, 21, 28, 29, 47, 49, 53, 57]. Prior to this work, no logic-based technique, as far as we know, solved this problem.<sup>3</sup> In this paper we introduced GRADED-SL to address

<sup>3</sup>In the related work section we discussed the logic GSL that, although motivated by the need to address the unique NE problem, only supplies a model-checking algorithm for a very small fragment of GSL that, it is assumed, is not able to express the existence of NE.

and solve the unique NE problem. We have demonstrated that GRADED-SL is elegant, simple, and very powerful, and can solve the unique NE problem for LTL objectives in 2EXPTIME, and thus at the same complexity that is required to merely decide if a NE exists. We also instantiate our formalism by considering the well-known prisoner’s dilemma and its iterated version. We have also shown that using the same approach one can express (and solve) the uniqueness of other standard solution concepts, e.g., subgame-perfect equilibria, again in 2EXPTIME. Finally, our work gives the first algorithmic solution to the model-checking problem of a graded variant of ATL\*, and proves it to be 2EXPTIME-COMPLETE.

The positive results presented in this paper open several directions for future work. We are most excited about extending LTL objectives to quantitative objectives such as mean-payoff or discounted-payoff. These naturally extend classic games with quantitative aspects. That is, the result of a play is a real-valued payoff for each player [58]. In a mean-payoff game, one is interested in the *long-run average* of the edge-weights along a play, called the *value* of the play. In the basic setting, there are two players, one wishing to minimize this value, and the other to maximize it. In the discounted version, the weights associated with edges are “*discounted*” with time. In other words, an edge chosen at time  $t$  adds a weight to the long-run average that is greater than the value the same edge would contribute if chosen later on. Because of their applicability to economics these games have been studied from an algorithmic perspective for some time [58]. Also, the connection of mean-payoff and discounted-payoff objectives with NE has been recently investigated in the multi-agent setting (see [13] for a recent work). However, extending our results to the weighted setting may prove challenging since, in this setting, the automata-theoretic approach gives rise to weighted-automata, for which many problems are much harder or undecidable (though not in all cases) [1, 5].

In the multi-agent setting, reasoning about epistemic alternatives plays a key role. Thus, an important extension would be to combine the knowledge operators in SLK [18] with the graded quantifiers we introduced for GRADED-SL. Since strategic reasoning under imperfect information has an undecidable model-checking problem [22], one may restrict to memoryless strategies as was done for SLK. More involved, would be to add grades to the knowledge operators, thus being able to express “there exists at least  $g$  equivalent worlds” [56].

Last but not least, another direction is to consider implementing GRADED-SL and its model-checking procedure in a formal verification tool. A reasonable approach would be, for example, to extend the tool SLK-MCMAS [18].

## Acknowledgments

We thank Michael Wooldridge for suggesting uniqueness of Nash Equilibria as an application of graded strategy logic. Benjamin Aminof is supported by the Austrian National Research Network S11403-N23 (RiSE) of the Austrian Science Fund (FWF) and by the Vienna Science and Technology Fund (WWTF) through grant ICT12-059. Sasha Rubin is a Marie Curie fellow of the Istituto Nazionale di Alta Matematica. Aniello Murano is partially supported by the GNCS 2016 project: Logica, Automi e Giochi per Sistemi Auto-adattivi.

## REFERENCES

- [1] S. Almagor, U. Boker, and O. Kupferman. What’s decidable about weighted automata? In *ATVA’11*, LNCS 6996, pages 482–491, 2011.
- [2] E. Altman, H. Kameda, and Y. Hosokawa. Nash equilibria in load balancing in distributed computer systems. *IGTR*, 4(2):91–100, 2002.
- [3] R. Alur, T. Henzinger, and O. Kupferman. Alternating-Time Temporal Logic. *JACM*, 49(5):672–713, 2002.
- [4] R. Alur, S. La Torre, and P. Madhusudan. Playing games with boxes and diamonds. In *CONCUR’03*, LNCS 2761, pages 127–141. Springer, 2003.
- [5] B. Aminof, O. Kupferman, and R. Lampert. Rigorous approximated determinization of weighted automata. *Theor. Comput. Sci.*, 480:104–117, 2013.
- [6] B. Aminof, A. Legay, A. Murano, and O. Serre.  $\mu$ -calculus pushdown module checking with imperfect state information. In *IFIP-TCS’08*, IFIP 273, pages 333–348. Springer, 2008.
- [7] B. Aminof, A. Murano, and S. Rubin. On CTL\* with graded path modalities. In *LPAR-20’15*, LNCS 9450, pages 281–296, 2015.
- [8] R. Axelrod. The evolution of strategies in the iterated prisoners dilemma. *The dynamics of norms*, pages 1–16, 1987.
- [9] F. Belardinelli. A logic of knowledge and strategies with imperfect information. In *LAMAS’15*, 2015.
- [10] A. Bianco, F. Mogavero, and A. Murano. Graded Computation Tree Logic. *TOCL*, 13(3):25:1–53, 2012.
- [11] K. G. Binmore. *Fun and Games: A Text on Game Theory*. D.C. Heath, 1992.
- [12] P. Bonatti, C. Lutz, A. Murano, and M. Vardi. The Complexity of Enriched muCalculi. *LMCS*, 4(3):1–27, 2008.
- [13] E. Boros, K. M. Elbassioni, V. Gurvich, and K. Makino. Nested family of cyclic games with k-total effective rewards. *CoRR*, abs/1412.6072, 2014.
- [14] T. Brihaye, A. D. C. Lopes, F. Laroussinie, and N. Markey. ATL with Strategy Contexts and Bounded Memory. In *LFCS’09*, LNCS 5407, pages 92–106, 2009.
- [15] D. Calvanese, G. De Giacomo, and M. Lenzerini. Reasoning in expressive description logics with fixpoints based on automata on infinite trees. In *IJCAI’99*, pages 84–89, 1999.
- [16] V. Capraro. A model of human cooperation in social dilemmas. *CoRR*, abs/1307.4228, 2013.
- [17] V. Capraro, M. Venanzi, M. Polukarov, and N. R. Jennings. Cooperative equilibria in iterated social dilemmas. In *SAGT’13*, LNCS 8146, pages 146–158, 2013.
- [18] P. Čermák, A. Lomuscio, F. Mogavero, and A. Murano. MCMAS-SLK: A Model Checker for the Verification of Strategy Logic Specifications. In *CAV’14*, LNCS 8559, pages 524–531. Springer, 2014.
- [19] K. Chatterjee, T. Henzinger, and N. Piterman. Strategy Logic. *IC*, 208(6):677–693, 2010.
- [20] R. Cornes, R. Hartley, and T. Sandler. An elementary proof via contraction. *Journal of Public Economic Theory*, 1(4):499–509, 1999.
- [21] J. B. D. Simchi-Levi, X. Chen. *The Logic of Logistics*:

- Theory, Algorithms, and Applications for Logistics Management.* Science and Business Media. 2013.
- [22] C. Dima and F. Tiplea. Model-checking ATL under Imperfect Information and Perfect Recall Semantics is Undecidable. Technical report, arXiv, 2011.
  - [23] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *ASFC'91*, pages 368–377, 1991.
  - [24] M. Faella, M. Napoli, and M. Parente. Graded Alternating-Time Temporal Logic. *FI*, 105(1-2):189–210, 2010.
  - [25] A. Ferrante, A. Murano, and M. Parente. Enriched Mu-Calculi Module Checking. *LMCS*, 4(3):1–21, 2008.
  - [26] K. Fine. In So Many Possible Worlds. *NDJFL*, 13:516–520, 1972.
  - [27] D. Fisman, O. Kupferman, and Y. Lustig. Rational Synthesis. In *TACAS'10*, LNCS 6015, pages 190–204. Springer, 2010.
  - [28] C. D. Fraser. The uniqueness of nash equilibrium in the private provision of public goods: an alternative proof. *Journal of Public Economics*, 49(3):389–390, 1992.
  - [29] A. Glazer and K. A. Konrad. Private provision of public goods, limited tax deducibility, and crowding out. *FinanzArchiv / Public Finance Analysis*, 50(2):203–216, 1993.
  - [30] E. Grädel. On the restraining power of guards. *J. Symb. Log.*, 64(4):1719–1742, 1999.
  - [31] E. Grädel, M. Otto, and E. Rosen. Two-Variable Logic with Counting is Decidable. In *LICS'97*, pages 306–317. IEEE Computer Society, 1997.
  - [32] J. Gutierrez, P. Harrenstein, and M. Wooldridge. Reasoning about equilibria in game-like concurrent systems. In *KR'14*, 2014.
  - [33] J. Gutierrez, P. Harrenstein, and M. Wooldridge. Expressiveness and complexity results for strategic reasoning. In *CONCUR'15*, LIPIcs 42, pages 268–282, 2015.
  - [34] B. Hollunder and F. Baader. Qualifying Number Restrictions in Concept Languages. In *KR'91*, pages 335–346. Kaufmann, 1991.
  - [35] O. Kupferman, G. Perelli, and M. Y. Vardi. Synthesis with rational environments. In *EUMAS'14*, LNCS 8953, pages 219–235, 2014.
  - [36] O. Kupferman, U. Sattler, and M. Vardi. The Complexity of the Graded muCalculus. In *CADE'02*, LNCS 2392, pages 423–437. Springer, 2002.
  - [37] O. Kupferman, M. Vardi, and P. Wolper. An Automata Theoretic Approach to Branching-Time Model Checking. *JACM*, 47(2):312–360, 2000.
  - [38] K. Leyton-Brown and Y. Shoham. *Essentials of Game Theory: A Concise, Multidisciplinary Introduction (Synthesis Lectures on Artificial Intelligence and Machine Learning)*. M&C, 2008.
  - [39] A. Lopes, F. Laroussinie, and N. Markey. ATL with Strategy Contexts: Expressiveness and Model Checking. In *FSTTCS'10*, LIPIcs 8, pages 120–132, 2010.
  - [40] V. Malvone, F. Mogavero, A. Murano, and L. Sorrentino. On the counting of strategies. In *TIME'15*, pages 170–179, 2015.
  - [41] J. H. Miller. The coevolution of automata in the repeated prisoner’s dilemma. *Journal of Economic Behavior & Organization*, 29(1):87–112, 1996.
  - [42] F. Mogavero, A. Murano, G. Perelli, and M. Vardi. What Makes ATL\* Decidable? A Decidable Fragment of Strategy Logic. In *CONCUR'12*, LNCS 7454, pages 193–208. Springer, 2012.
  - [43] F. Mogavero, A. Murano, G. Perelli, and M. Vardi. Reasoning About Strategies: On the Model-Checking Problem. *TOCL*, 15(4):34:1–42, 2014.
  - [44] F. Mogavero, A. Murano, and M. Vardi. Reasoning About Strategies. In *FSTTCS'10*, LIPIcs 8, pages 133–144. Leibniz-Zentrum fuer Informatik, 2010.
  - [45] F. Mogavero, A. Murano, and M. Vardi. Relentful Strategic Reasoning in Alternating-Time Temporal Logic. In *LPAR'10*, LNAI 6355, pages 371–387, 2010.
  - [46] D. E. Muller and P. E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of rabin, mcnaughton and safra. *Theor. Comput. Sci.*, 141(1&2):69–107, 1995.
  - [47] A. Orda, R. Rom, and N. Shimkin. Competitive routing in multiuser communication networks. *IEEE/ACM Trans. Netw.*, 1(5):510–521, 1993.
  - [48] G. Papavassiliopoulos and J. B. Cruz. On the uniqueness of nash strategies for a class of analytic differential games. *Journal of Optimization Theory and Applications*, 27(2):309–314, 1979.
  - [49] L. Pavel. *Game Theory for Control of Optical Networks*. Science and Business Media. Springer, 2012.
  - [50] A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *FOCS'90*, pages 746–757, 1990.
  - [51] A. Rubinstein. Finite automata play the repeated prisoner’s dilemma. *Journal of Economic Theory*, 39(1):83–96, 1986.
  - [52] R. Selten. Spieltheoretische behandlung eines oligopolmodells mit nachfragetrageheit. *Zeitschrift fur die gesamte Staatswissenschaft*, 121:301–324, 1965.
  - [53] H. R. V. T. C. Bergstrom, L. E. Blume. On the private provision of public goods. *Journal of Public Economics*, 29(1):25–49, 1986.
  - [54] W. Thomas. Infinite trees and automaton definable relations over omega-words. In *STACS'90*, pages 263–277, 1990.
  - [55] M. Ummels. Rational behaviour and strategy construction in infinite multiplayer games. In *FSTTCS'06*, LNCS 4337, pages 212–223, 2006.
  - [56] W. van der Hoek and J.-J. Meyer. Graded modalities in epistemic logic. In *LFCS'92*, LNCS 620, pages 503–514. 1992.
  - [57] Y. Zhang and M. Guizani. *Game Theory for Wireless Communications and Networking*. CRC Press, 2011.
  - [58] U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theor. Comput. Sci.*, 158(1&2):343–359, 1996.

## **4.7 Publication**

The rest of this page is intentionally left blank

# Automatic Verification of Multi-Agent Systems in Parameterised Grid-Environments

Benjamin Aminof  
Technische Universität Wien,  
Austria  
benj@forsyte.at

Aniello Murano, Sasha Rubin  
Università di Napoli Federico II, Italy  
aniello.murano@unina.it  
sasharubin@unina.it

Florian Zuleger  
Technische Universität Wien,  
Austria  
zuleger@forsyte.at

## ABSTRACT

We present a framework for modeling and analysing multiple mobile agents on grid-environments such as finite mazes and labyrinths. Agents are modeled as automata, and the grid-environments are parameterised by their size and the relative positions of the obstacles. We study the verification problem, i.e., whether given agents complete a given task on a given (possibly infinite) set of grid-environments. We identify restrictions on the agents and on the environments for which the verification problem is decidable (and in PSPACE). These assumptions are: i) there are a bounded number of obstacles, and ii) the agents are not allowed to issue commands like “increase my  $x$ -coordinate by 1” but can only issue commands that change their relative positions, e.g., “increase my  $x$ -coordinate until I go past this wall”.

We prove PSPACE-hardness already for the verification problem of a single agent on singleton parameterised environments with no obstacles. It is therefore remarkable that the PSPACE-upper bound also holds for the verification problem with multiple agents, parameterised environments and multiple obstacles. We prove that weakening either of restrictions i) or ii) results in undecidability. The importance of this work is that it is the first to give a sound and complete decision procedure for the verification problem on parameterised grid-like environments. Previous work either involved only a single grid, restricted the scheduling of the agents, or excluded grids altogether.

## General Terms

Theory, Verification

## Keywords

Computational Models; Autonomous Mobile Agents; Distributed Robot Systems; Grids; Parameterised Verification

## 1. INTRODUCTION

Physical multiagent systems are designed to move in space. Thus 2D or 3D space, or their abstractions into grids [41, 7, 25, 26, 3], are the canonical environments for modeling multiagent systems. In some cases, e.g., if the environment

**Appears in:** *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016), J. Thangarajah, K. Tuyls, C. Jonker, S. Marsella (eds.), May 9–13, 2016, Singapore.*

Copyright © 2016, International Foundation for Autonomous Agents and Multiagent Systems ([www.ifaamas.org](http://www.ifaamas.org)). All rights reserved.

is too dangerous or expensive to be reached by humans, we may assume that agents initially only have partial information about the environment. For instance, they may not know the exact size of the space or the extent of the obstacles, but they may know the relative positions of obstacles. Thus, agents should be designed so that they operate correctly on all possible grids that are consistent with their information [44, 43, 7, 31, 19, 25, 3]. In this paper we study the verification problem for such multiagent systems that have partial information about the environment.

Model checking is one of the main paradigms for automatic verification: the system is modeled as a finite structure  $M$ , the specification is expressed as a formula  $\phi$  in a suitable logical formalism, and a model checking algorithm is used to verify if the structure  $M$  satisfies the formula  $\phi$  [13]. The main obstacle to using this paradigm in our setting is that, since the environment is not completely-known, we have to model check many (potentially infinitely many) structures  $M$ . There are two main approaches to overcoming this problem, abstraction [13] and parameterised model-checking [6]. We pursue the second approach.

**Motivating Scenario: Navigating Manhattan.** Suppose you are arrive at Grand Central Station (GCS) in midtown Manhattan and want to get to a certain landmark, say Carnegie Hall (CH), by walking along the streets and avenues. Perhaps you already know (because you have a crude map drawn on a piece of paper) the relative positions of some of the major landmarks,<sup>1</sup> e.g., Times Square (TS) is west of GCS, somewhere between TS and GCS is a point, North of which is the Museum of Modern Art (MOMA), and CH is North of MOMA. Can you use this information to reach CH? Any algorithm that you use to navigate will probably have primitives of the form “walk east until you reach landmark X”, or “walk east for a while, and stop at some point to look at your map again”. Now, suppose you have a generic algorithm for navigating, given the relative positions of landmarks. Is your algorithm always guaranteed to succeed? This is the parameterised verification problem. It is parameterised because you do not want to test your algorithm only on this particular part of Manhattan, but on all, potentially infinitely many, such environments.

We believe that there are many scenarios of multi-agent systems where navigating using exact positioning is not possible or desirable. On the one hand, precise positioning requires complicated hardware (such as GPS receivers, servo

<sup>1</sup>Even if one has a map which is to scale, one probably does not navigate by measuring or calculating the distance one travels.

mechanisms, etc.), and on the other hand, even if the agent knows its exact location, many times it does not have this information for other objects in the environment, e.g., if a complete to-scale map of the environment is not available, and only a partial schematic exists (such as a “map” drawn on a napkin), or if there is no map at all.

**Aims and Contributions.** In order to capture the motivating example above, as well as many other scenarios, the aim of this work is to provide a formal framework for reasoning mathematically and computationally about multiple mobile-agents in grid-environments, having some tasks to perform. We model agents as finite-state machines that walk on grids containing obstructions. The agents can move along the axes, e.g., in two dimensions they can move horizontally and vertically but not diagonally. Agents maintain a fixed number of beacons which they can place on the grid to remember positions they have been to, and can be used for navigation or coordination with other agents. Grid-environments are parameterised by their size and the relative positions of the obstacles on the grid. Tasks are formalised in a suitable linear-time logic that extends LTL by adding the ability to talk about the relative positions of the agents and obstacles at different points in time. In particular, it can express tasks like gathering, border patrol, and line-formation. Agents move asynchronously according to an optional scheduling policy; this is motivated by the assumption that agents have no common notion of time since they have different internal clocks [40, 2, 6].

We prove that the verification problem is decidable (and in PSPACE) under two restrictions: i) there is a bounded number of obstacles, and ii) the agents are not allowed to issue commands like “increase my  $x$ -coordinate by 1” but can only issue commands that change their relative positions, e.g., “increase my  $x$ -coordinate until I go past this wall”. We prove PSPACE-hardness already for the verification problem of a single agent on singleton parameterised environments with no obstacles and no beacons. We therefore find it remarkable that the PSPACE-upper bound also holds for the verification problem with multiple agents, beacons, parameterised environments and multiple obstacles.

In stark contrast, we prove that verification is undecidable, even for very simple tasks (such as reachability), if either of these two conditions is relaxed. E.g., even on empty grids, thus i) holds, if agents can issue commands with absolute distances then verification is undecidable; and, if ii) holds, but there is no bound on the number of obstacles, then verification is undecidable even on simple “Avenues-and-Street” grids.<sup>2</sup>

The importance of this work is that it is the first to give a sound and complete decision procedure for the verification problem on parameterised grid-like environments. Previous work either involved abstractions [37], only a single environment [36], parameterised environments but with scheduling that restricted the number of turns [2], or parameterised environments but excluded grids altogether [45]. Thus, completely new ideas are needed.

Our work is based on the insight that the ability to move precisely in a coordinate system results in undecidability, and that methods from formal methods can shed light on

---

<sup>2</sup>Moreover, we get undecidability even if the absolute placement of the obstacles is not fixed, e.g., the width of the avenues may vary.

weakened agents that still retain the ability of moving relative to other objects. In this way we obtain a powerful new paradigm for modelling multi agent systems.

**Related work.** The parameterised model-checking problem has been considered in the verification community for models of (typically immobile) processes — see [6] for a survey. On the other hand, the distributed-computing literature consideration of mobile robots (i.e., automata walking on graphs) is mostly mathematical rather than algorithmic, but naturally (and interestingly) provides theorems that are parameterised (in, e.g., the number of robots, the number of internal states of the robots, etc.) [8, 38, 28, 20, 15, 14, 27].

The main tool we exploit for decidability is a spatio temporal logic called constraint LTL that was originally defined for the verification of counter machines [17, 18].

Our paper considers qualitative tasks. In contrast, [43] studies the shortest path problem in a 2D model that bears some similarity to ours: obstacles are non-intersecting rectangles with sides parallel to the axes, and obstacles that are in a straight line of sight can be seen, and also measured.

Our beacons are reminiscent of pebble automata [21, 4] which themselves are related to extensions of first-order logic with transitive closure; we remark that in our model there is no stack-like restriction on the orders that beacons can placed or retrieved, and yet we still achieve decidability.

Our paper considers verification. A related problem is synthesis with partially known environments, also known as generalised planning, which typically considers one dimensional environments [16, 32, 34, 35].

Besides navigation in known environments, the AAMAS community has studied distributed solutions to navigation tasks in unknown environments, and in contrast to our work, most studies were experimentally verified: [33] contains a technique for computationally sophisticated robots to solve the multi-robot coverage task problem using Voronoi partitioning; In [12], computationally and memory limited agents are studied using coalition logic to manage team-formation; in [29], the agents communicate by exchanging messages with a predefined set of other agents, whose task is to reach a goal state in weighted planar graphs where robots discover adjacent nodes to visited nodes; [24] contains a variation of the  $A^*$  algorithm for finding optimal paths in partially known graphs, i.e., where robots discover nodes adjacent to visited ones; and the authors of [11], motivated by reducing deployment time, consider intelligent cameras and autonomous robots for a museum guiding service.

## 2. NOTATION AND BACKGROUND

**Notation.** Let  $\mathbb{N}$  denote the integers, and  $\mathbb{N}_0$  be  $\mathbb{N} \cup \{0\}$ . For  $N \in \mathbb{N}$ , write  $[N]$  for the set  $\{1, 2, \dots, N\}$ . Fix an infinite set  $\text{var}$  of variables (that vary over  $\mathbb{N}$ ). Fix, once and for all, a dimension  $D \in \mathbb{N}$  for the grids (e.g., if  $D = 1$  then the agents move on a line, if  $D = 2$  then they move on planar grids, etc). Write  $\epsilon_i$  for the  $D$ -tuple  $(0, \dots, 0, 1, 0, \dots, 0)$  that is 0 everywhere except for the  $i$ th co-ordinate which is 1. Write  $\bar{0}$  for the  $D$ -tuple  $(0, \dots, 0)$ . Write  $\bar{x}[i \leftarrow a]$  for the tuple  $\bar{x}$  with the value of the  $i$ th co-ordinate replaced by the  $a$ , e.g.,  $(0, 0, 1)[2 \leftarrow 1]$  is  $(0, 1, 1)$ . Throughout, we freely interchange between functional notation  $f(i) = b$  and vector notation  $(\dots, b, \dots)$ . For an infinite sequence  $\sigma$ , write  $\sigma_i$  for the  $i$ th element of  $\sigma$ .

**Two-counter Machines.** Our undecidability proofs proceed by reducing the halting problem of two counter machines to the verification problem. An *input-free 2-counter machine* (2CM) [42] is a deterministic program manipulating two nonnegative integer counters using commands that can increment a counter by 1, decrement a counter by 1, and check whether a counter is equal to zero. We refer to the “line numbers” of the program code as the “states” of the machine. One of these states is called the *halting state*, and once it is entered the machine *halts*. The *non-halting problem for 2CMs*, which is known to be undecidable [42], is to decide given a 2CM  $M$  whether it does not halt.

**Constraint linear-temporal logic (C-LTL).** Linear temporal logic (LTL) is nowadays a well-established logic for reasoning about reactive systems. We recall the definition of constraint linear-temporal logic (C-LTL) introduced in [18]. C-LTL is a spatio-temporal logic that has been introduced for the verification of constraint automata, a class of automata that can be viewed as abstractions of counter-automata. It is the main tool that allows us to achieve decidability of the verification problem for multi-agent systems on grids.

A *term*  $t$  is either an element of  $\mathbb{N}_0$  or an expression of the form  $O^i(x)$  ( $x \in \text{var}$  and  $i \in \mathbb{N}_0$ ). We identify  $O^0(x)$  with  $x$ . An *atomic constraint*  $R$  is an expression of the form  $t \bowtie t'$  where  $t, t'$  are terms and  $\bowtie$  is one of  $=, <, \equiv_a^b$  ( $0 \leq a < b \in \mathbb{N}_0$ ). Here  $=, <$  are interpreted as usual on  $\mathbb{N}_0$ , and  $\equiv_a^b$  is the modulo unary relation, i.e.,  $\equiv_a^b(x)$  iff  $x = a \bmod b$ . E.g.,  $O^2(x) < O^1(y)$  is an atomic constraint. Informally, it means that the value of  $x$  two time steps ahead is less than the value of  $y$  one time step ahead. A *constraint* is a Boolean combination of atomic constraints. A *number constraint* is a constraint that only involves elements of  $\mathbb{N}_0$  and variables, i.e., no terms of the form  $O^i(x)$  for  $i > 0$ .

The syntax of C-LTL is given by:  $\varphi ::= R \mid \neg\varphi \mid (\varphi \vee \varphi) \mid O\varphi \mid \varphi U \varphi$  where  $R$  is an atomic constraint. A *valuation* is a function  $\text{var} \rightarrow \mathbb{N}_0$ . Models of C-LTL formulas are infinite sequences of valuations, i.e.,  $\sigma = \sigma_0 \sigma_1 \dots$  where each  $\sigma_j$  is a valuation. The semantics of C-LTL follows:

$$\begin{aligned} (\sigma, i) \models R &\quad \text{iff } \sigma_{i+l}(x) \bowtie \sigma_{i+m}(y) \text{ and} \\ &\quad R = O^l(x) \bowtie O^m(y) \\ (\sigma, i) \models \neg\varphi &\quad \text{iff } (\sigma, i) \not\models \varphi \\ (\sigma, i) \models \varphi \vee \psi &\quad \text{iff } (\sigma, i) \models \varphi \text{ or } (\sigma, i) \models \psi \\ (\sigma, i) \models O\varphi &\quad \text{iff } (\sigma, i+1) \models \varphi \\ (\sigma, i) \models \varphi U \psi &\quad \text{iff } \exists j \geq i : (\sigma, j) \models \psi \text{ and} \\ &\quad \forall l \in [i, j) : (\sigma, l) \models \varphi \end{aligned}$$

Write  $\sigma \models \varphi$  if  $(\sigma, 0) \models \varphi$ . We use the usual derived operators  $F\varphi := \text{true} U \varphi$ , its dual  $G\varphi := \neg F \neg\varphi$ , and  $\varphi_1 R \varphi_2$  which is the dual of  $U$ , i.e.,  $\neg(\neg\varphi_1 U \neg\varphi_2)$ , as well as the short-hands  $\wedge$  and  $\vee$ . For example,  $G(x < O^1(x))$  means that “globally the value of  $x$  at the current point of time is smaller than the value of  $x$  at the next point of time”, which simply means that  $x$  is always strictly increasing.

We mention the following important facts about C-LTL, see [18]. First, recall that linear-temporal logic (LTL) has the same syntax as C-LTL except that instead of atomic constraints it has atomic propositions such as  $p$  and  $q$ ; and LTL formulas are interpreted over infinite sequences of sets of atomic propositions. Now, C-LTL generalizes LTL (the idea is to associate to each atom  $p$  a unique variable  $x$ , and

replace  $p$  by the constraint  $x = 1$  and replace  $\neg p$  by the constraint  $x = 0$ ). Second, the satisfiability problem (and thus also the validity problem) for C-LTL (i.e., given a C-LTL formula  $\varphi$ , decide if there exists  $\sigma$  such that  $\sigma \models \varphi$ ) is decidable, and in fact has the same complexity as for LTL, i.e., it is PSPACE-complete.

### 3. ENVIRONMENTS AND AGENTS

**Environments.** We model environments as grids of arbitrary, but fixed, dimension containing obstacles whose faces are parallel to the axes. A (finite or infinite) set of environments of interest is called a *parameterised environment*. Comparisons between points of  $\mathbb{N}^D$  are taken point-wise, e.g.,  $\bar{v} \leq \bar{w}$  means that  $v_i \leq w_i$  for  $i \leq d$ . We describe various subsets of  $\mathbb{N}^D$ . A *rectangle* is a subset of  $\mathbb{N}^D$  of the form  $\{\bar{a} \in \mathbb{N}^D : \bar{v} \leq \bar{a} \leq \bar{w}\}$ , where  $\bar{v} \leq \bar{w} \in \mathbb{N}^D$ . The points  $\bar{v}$  and  $\bar{w}$  are called the *SW-* and *NE-corners* of the rectangle, respectively.<sup>3</sup> The *interior* of a rectangle  $R$ , whose SW and NE corners are  $\bar{v}$  and  $\bar{w}$ , is the set  $R^{int} := \{\bar{a} \in \mathbb{N}^D : \bar{v} < \bar{a} < \bar{w}\}$ .

A *K-obstruction*  $M = \langle M_1, \dots, M_k \rangle$  is a tuple of  $K$  many rectangles. Given  $L \in \mathbb{N}$ , and a *K-obstruction*  $M$  such that all rectangles in  $M$  are contained in  $[L]^D$ , the  $(L, M)$ -environment is the graph  $G = (V, E)$  whose vertex set  $V = [L]^D \setminus (\bigcup_i M^{int})$  (i.e., it is the rectangle with corners  $(0, \dots, 0)$  and  $(L, \dots, L)$ , and the interiors of the rectangles in  $M$  removed), and whose edge relation  $E \subset V \times V$  consists of pairs  $(\bar{v}, \bar{w})$  such that for some  $i$  we have  $|v_i - w_i| = 1$  and for  $j \neq i$  we have  $v_i = w_i$  (i.e., horizontally- or vertically-adjacent vertices are joined with an edge). Environments are also called finite labyrinths [30]. A *parameterised environment* is some finite or infinite set of environments.

Since only the interiors of the obstructing rectangles are removed, agents are allowed to move along the “walls” of an obstacle. Observe that this has the effect that if, for example, two rectangles (or a rectangle and a boundary of the grid) have a common edge/face, agents can still walk between them “along the adjoining wall”. If one wants to ensure, that agents cannot walk between two rectangles, one can specify that the rectangles overlap. We add that overlapping rectangles can be used to build obstacles of different shapes than rectangles. For example, one can build a non-rectangular obstacle such as an “L” by using two rectangles and specifying that they have the same SW-corner, and that one rectangle is thinner and higher and the other rectangle is broader and more shallow. We finally add, that if one wants to remove the points common to a rectangle and a boundary of the grid<sup>4</sup> this could be accomplished by adding a second type of rectangle with the desired behaviour. For ease of exposition we refrain from doing so.

**Agents.** Agents are modeled as finite-state machines that can move along the axes of the grid. E.g., in a 2-dimensional grid, each single move is in direction north, south, east or west, but not diagonally. More formally, an agent  $A$  is tuple  $(Q, I, \delta)$  where  $Q$  is a finite set of *states* (we usually assume

<sup>3</sup>Note that even though our grids are not necessarily 2-dimensional this is perhaps the most common case, and we find it assists in imagining things if we use 2-dimensional terms like “NE”.

<sup>4</sup>This can be used also to simulate a grid with sides of different length by positioning suitable obstacles along the boundaries.

w.l.o.g. that  $Q = [|Q|]$ ;  $I \subseteq Q$  is a set of *initial states*;  $\delta \subseteq Q \times GC \times Q$  is a finite *transition relation*. Elements of  $GC$  are called *guarded commands* and are of the form  $(d, guard)$  where  $d \in [D]$  is the axis along which the agent should move, and *guard* is a condition specifying some restriction that the path taken by the agent during this move must satisfy for the move to be possible. For example, the guard may specify that the agent moves exactly one step in the positive direction of the axis. There are many possible definitions for the types of guards that can be used, and these choices determine the “power” the agents have, and we will consider a few possible definitions of  $GC$  in later sections. In all cases, we assume that the guard can test whether the agent is at a boundary of the grid. Observe that we implicitly assume<sup>5</sup> that i) the path taken by an agent in a single command does not collide with any other agent (in more detail, an agent may start and end in the same position as another agent, but it may not “pass-through” an agent in a single move); and ii) the path taken does not use positions occupied by an obstruction. Note that the guard may be such that it does not fully specify the end position of a move, in which case the system nondeterministically chooses it (in a way which satisfies the guard).

## 4. UNDECIDABILITY RESULTS

In this section we give two simple undecidability results that direct our choice of model for agents in parameterized grid environments, presented in the next section.

A common assumption is that agents have the ability to exactly know and/or control their position [41, 27, 38, 39, 36]. This assumption abstracts real-world situations of agents that, for example, are equipped with a global positioning system (GPS). Unfortunately, as demonstrated by the following theorem, it is very easy to show that making this assumption leads to undecidability of the verification problem with respect to even the simplest tasks and parameterized grid environments. It is worth noting that this assumption can be made, without leading to undecidability, in many cases where the environments of interest are not grids, e.g., environments with bounded tree-width [45, 2].

Say that an agent has *precise positioning* if  $GC$  contains (directly or indirectly) commands of the form  $(d, x_d = x'_d + j)$ , with  $j \in \{0, -1, 1\}$ , where  $x_d, x'_d$  are the positions of the agent along the  $d$  axis at the start and end of the move, and guards that allow the agent to test its position, in particular whether it is on the bottom or left boundary of the grid.

**THEOREM 1.** *The following problem is undecidable: given a single agent with precise positioning, and a state  $h$  of the agent, verify that for all obstacle-free finite 2-dimensional grids no run of the agent ever enters state  $h$ .*

**PROOF.** The proof is by reduction from the non-halting problem of 2-counter machines. Given a 2CM  $M$ , we build an agent  $A$  that has the same states as the 2CM plus an additional *overflow* state. The agent simulates the 2CM by using its position  $(x, y)$  to encode a configuration of  $M$  where the first counter has value  $x$  and the second counter has value  $y$ . The agent begins by going to the origin of the grid (south as far as possible then west as far as possible). Incrementing (resp. decrementing) a counter is done by simply moving one

<sup>5</sup>In other words, an instruction can be taken only if these extra conditions are satisfied.

step in the correct direction. A test for zero amounts to the agent checking that it has collided with the bottom or left boundary of the grid. If the agent wants to go right or up (to simulate an increment), but hits the boundary, it enters the special *overflow* state and discontinues the simulation.

It is easy to see that, on an  $L \times L$  grid, the agent can simulate any prefix of the computation of the 2CM in which the values of both counters never go above  $n$ . Thus,  $M$  does not halt iff, for all  $L \in \mathbb{N}$  the agent  $A$  doesn’t enter the halting state in the  $L \times L$  grid.  $\square$

Observe that it is simple to modify Theorem 1 and obtain that, if one wishes to consider grids of unbounded size, verifying any non-trivial task for an agent with precise positioning is undecidable. Hence, if we wish to be able to automatically verify that the agent performs its task on a families of grids of unbounded size we must limit the agent’s ability to precisely control/know its location.

In our quest for a model for describing agents in parameterized grid environments we are motivated by the example from the introduction of a person navigating in an unknown city using a schematic map, or instructions from a GPS unit. Note that one is usually *not* using the precise positioning information available to the GPS, but one is simply following instructions such as “turn left at the next corner”. All that is needed to follow such an instruction is the ability to detect the next corner. Let us consider then a model for the agent where the guarded command can specify a guard that expresses (directly or indirectly) that the end position  $\bar{x}'$  of the move is a vertex of some obstacle in the environment, and no point strictly between the start point and the end point is a vertex of an obstacle.<sup>6</sup> This is arguably a very basic way of detecting a corner, as one is doing it “by feel” (reminiscent of how a blind person uses a white cane). We call agents with this ability agents with *next corner detection*.

For the following theorem, given  $L \in \mathbb{N}$ , let the  $L$ -regular-city be the 2-dimensional grid environment of length  $2L+1$ , in which for every  $i, j \in \{0, \dots, L\}$  there is a  $1 \times 1$  obstacle whose SW corner is located at  $(2i, 2j)$ .

**THEOREM 2.** *The following problem is undecidable: given a single agent with next corner detection, and a state  $h$  of the agent, verify that for all  $L \in \mathbb{N}$ , no run of the agent ever enters state  $h$  while navigating the  $L$ -regular-city.*

**PROOF.** The proof is very similar to that of Theorem 1. Unlike Theorem 1, we do not assume that the agent has precise positioning, and thus it can not encode the counters of the 2CM directly by its position. However, it can do so indirectly as follows: it encodes the value of the first counter by the number of obstacles that are strictly to the west of it (at the same  $y$  coordinate as it is), and the value of the second counter by the number of obstacles that are strictly to the south of it (at the same  $x$  coordinate). The main difference with the agent in the proof of Theorem 1, is that incrementing (resp. decrementing) a counter is done by moving to the SW corner of the next obstacle in the correct direction, which takes two moves. For example, if  $q$  is an increment of the first counter, the agent moves twice to the next corner to the east: the first move takes it from the SW corner of the current obstacle to its SE corner (and to state

<sup>6</sup>If one wishes, one can further assume that if no such  $\bar{x}'$  exists then the agent walks as far as it can.

$q'$ ), and the next move takes it to the SW corner of the next obstacle (and the next state of the 2CM).

It follows that, in an  $L$ -regular-city, the agent can simulate any prefix of the computation of  $M$  in which the counter values don't exceed  $L$ . Thus,  $M$  does not halt iff, for all  $L \in \mathbb{N}$ , the agent never enters the halting state of  $M$ .  $\square$

As was the case with Theorem 1, one can modify the proof above to obtain undecidability for any non-trivial task. The proof goes through unchanged even if the definition of an  $L$ -regular-city is modified to allow the obstacles and the gaps between them to have different sizes, as long as the general structure of aligned streets and avenues is maintained.

## 5. A MODEL OF MAS ON GRIDS

In this section we present a model of multiple agents operating asynchronously in unbounded parameterized grid environments with a bounded number of obstacles. We also define a way to specify the tasks for these agents.

The theorems in the previous section imply that if we want to be able to decide whether our agents achieve their specified tasks on grid environments of unbounded size, and still maintain the agents' ability to detect corners of obstacles, we should disallow the agents from precisely specifying the distance they travel as well as bound the number of obstacles in a parameterized environment. However, we place no bounds on the sizes and positions of the obstacles.

Informally, the guarded commands of the agents do not allow them to specify absolute positions or step sizes, and only allow one to express relative positioning with respect to other objects in the environment. In order to regain some of the power lost by this relative positioning, agents are also allowed to place some *fixed* number of smart markers, called *beacons* (similar to non-directional radio beacons used in air and marine navigation), in the environment. At any point in time, and from any point in the environment, an agent can test whether its position along any of the  $D$  axes is smaller, equal, or larger than that of any other agent, corner of an obstacle, or any beacon (of any agent). A beacon can also be remotely retrieved by the agent that owns it (i.e., without going back to it) and deposited at the current location of the agent. Agents are also able to query other agents as to their current local state.

Note that the beacons allow the agents to perform certain operations that would otherwise be impossible with relative positioning, e.g., such as marking a position an agent can later return precisely to, or to draw a virtual line in the town square such that the agents later gather in the square with half of them on each side of the line. The beacons can be implemented, for example, by making the following (limited) use of a GPS system and memory registers: placing a beacon number  $i$  of agent  $j$  in the current position is *simulated* by  $j$  recording, in its  $i$ 'th register, the current position as indicated by the GPS. Comparing an agent's position to that of the beacon is done by querying the GPS system as to whether some coordinate  $d$  of the current location is smaller, larger, or equal, to the value in the corresponding register. Retrieving the beacon is done by simply overwriting the value in the register. Obviously, not all the power of this model may be needed or possible to implement by a real system. For example, our agents can see through walls (I.e., they are in a "smart city" where the corners of each obstacle broadcast radio signals), and one is welcome to not

make use of any unnecessary feature (E.g., limit the agents to not see through obstacles).

For the rest of the paper fix the number  $B$  of beacons, the number  $N$  of agents, and the number  $K$  of obstacles.

**Variables.** We define the following variables (i.e., elements of *var*), that will be used later to define other important concepts such as guards, runs, etc.:

- *agent variables* *avar* :=  $\{x_{n,d} : n \in [N], d \in [D]\}$ ;
- *primed agent variables* *avar'* :=  $\{x'_{n,d} : n \in [N], d \in [D]\}$ ;
- *beacon variables* *bvar* :=  $\{b_{n,d}^j : n \in [N], d \in [D], j \in [B]\}$ ;
- *primed beacon variables* *bvar'* :=  $\{b'^j_{n,d} : n \in [N], d \in [D], j \in [B]\}$ ;
- *agent local-state variables* *svar* :=  $\{s_n : n \in [N]\}$ ;
- *obstruction variables* *ovar* :=  $\{u_{k,d}, v_{k,d} : k \in [K], d \in [D]\}$ ;
- the *size variable*  $l$ , and the *turn variable* *turn*.

The values of each variable have the following meanings.  $\bar{x}_n := (x_{n,1}, \dots, x_{n,d})$ , is the current position of agent  $n$ , and the primed version  $\bar{x}'_n$  is the next position of agent  $n$ ; the position of the  $j$ 'th beacon of agent  $n$  is  $\bar{b}_n^j := (b_{n,1}^j, \dots, b_{n,d}^j)$ , and its next position is  $\bar{b}'_n^j := (b'^j_{n,1}, \dots, b'^j_{n,d})$ . The value of  $s_n$  is the current local state of the  $n$ th agent. The variables  $\bar{u}_k$  and  $\bar{v}_k$  are the SW- and NE-corners of the  $k$ th obstacle;  $l$  is the length of the grid; and *turn* is which agent's turn it is (used to define schedules). For simplicity, we assume that the values of  $l$  and variables in *ovar* do not change over time, i.e., that the size of the grid and the positions of obstacles are fixed throughout a run of a system.

**Parameterised environments with a fixed number of obstacles.** An *environment constraint* is a number constraint  $\phi$  over variables *ovar*  $\cup \{l\}$ . It determines a set of environments  $\mathcal{G}_\phi$  as follows. Every  $(L, M)$ -environment  $G$  with  $M = \langle M_1, \dots, M_K \rangle$  determines a valuation  $\sigma_G$  over *ovar*  $\cup \{l\}$  as follows:  $\sigma_G$  maps variable  $l$  to  $L$ , and for each  $i \in [K]$ ,  $\sigma_G$  maps  $\bar{u}_i$  (resp.  $\bar{v}_i$ ) to the SW-corners (resp. NE-corner) of the rectangle  $M_i$ . Thus, a number constraint  $\phi$  determines the set  $\mathcal{G}_\phi$  of environments  $G$  such that the valuation  $\sigma_G$  satisfies  $\phi$ . E.g., for  $d = 2, K = 1$ , and 2-tuples of variables  $(u_1, u_2), (v_1, v_2)$  representing the inner rectangle, the constraint  $l \geq 10 \wedge \bigwedge_{i=1,2} 0 < u_i < v_i < l$  determines the set of rectangular "race-tracks" of size at least 10.

Note that by constraining the variable  $l$ , we may specify environments of a single size (e.g.,  $l = 4$ ), a finite set of sizes (e.g.,  $l = 4 \vee l = 5$  or  $l < 10$ ), or infinitely many sizes (e.g.,  $l > 10$  or  $\equiv_0^2(l)$ ).

**Agents.** Recall from Section 3 that an agent  $A$  is tuple  $(Q, I, \delta)$  where  $Q$  is a finite set of states,  $I \subseteq Q$  is a set of initial states, and  $\delta \subseteq Q \times \text{GC} \times Q$  is a finite transition relation, where the guarded commands *GC* are of the form  $(d, \text{guard})$  where  $d \in [D]$ . It remains to define the possible values of *guard*. We let *guard* be any number constraint over *avar*  $\cup$  *avar'*  $\cup$  *bvar*  $\cup$  *bvar'*  $\cup$  *ovar*  $\cup \{l\}$ . Note that a transition specifies that beacon  $j$  of agent  $n$  is retrieved and dropped in the new location of the agent by specifying in the

guard that  $\bar{x}'_n = \bar{b}'^j_n$ . An *agent ensemble* is a tuple of agents  $\mathcal{A} = \langle A_1, \dots, A_N \rangle$  (such that all the agents' commands use the same  $N, K$  and  $B$ ).

**Configurations, Schedules, Runs.** Let  $G = (V, E)$  be an  $(L, M)$ -environment with  $K$  obstacles, and let  $\mathcal{A}$  be an agent ensemble. A *configuration* of  $\mathcal{A}$  on  $G$  is a tuple  $c := (loc, locb, state)$ , where  $loc : [N] \rightarrow V$  maps an agent to its location;  $locb : [N] \times [B] \rightarrow V$  maps a beacon to its location; and  $state : [N] \rightarrow \cup_j Q_j$ , such that  $state(j) \in Q_j$  (for  $j \in [N]$ ), maps an agent to its current local state. Say that  $c$  is an *initial configuration* if  $state(j) \in I_j$  (for  $j \in [N]$ ).

We can think of  $c$  as a valuation  $val_c$  over variables  $avar \cup bvar \cup ovar \cup svar$  that (for  $n \in [N], j \in [B], k \in [K]$ ): maps  $\bar{x}_n$  to  $loc(n)$ , maps  $\bar{b}_n^j$  to  $locb(n, j)$ , maps  $\bar{u}_k$  (resp.  $\bar{v}_k$ ) to the SW-corner (resp. NE-corner) of the  $k$ th obstruction  $M_k$  of the environment  $G$ ; and maps  $s_n$  to  $state(n)$ . Similarly, if we are given a second configuration  $c'$ , we can further extend  $val_c$  to get a valuation  $val_{c,c'}$  that maps  $\bar{x}'_n$  to  $loc'(n)$  and  $\bar{b}'^j_n$  to  $loc'(n, j)$ .

For configurations  $c = (loc, locb, state)$  and  $c' = (loc', locb', state')$ , and  $n \in [N]$ , write  $c \vdash_n c'$  if agent  $n$  can, by taking one transition, change the configuration from  $c$  to  $c'$ . Formally,  $c \vdash_n c'$  if there exists  $(q, (d, guard), q') \in \delta_n$  and  $\alpha \in \mathbb{Z}$  such that:

1.  $state(n) = q$ , and  $state' = state[n \leftarrow q']$  (i.e., the agent changes state from  $q$  to  $q'$ );
2.  $loc' = loc[n \leftarrow loc(n) + \alpha e_d]$  (i.e., the agent moves some distance  $\alpha \in \mathbb{Z}$  along the  $d$ th axis);
3. For every  $m \in [N], j \in B$ , either  $loc'(m, j) = locb(m, j)$ , or  $m = n$  and  $locb'(m, j) = loc'(n)$ ; (i.e., the agent can transfer any of its beacons to its new location);
4.  $\sigma_{loc, loc'} \models guard$  (i.e., the guard holds);
5. for every  $\bar{v} \in \mathbb{N}_0^D$  strictly between  $loc(n)$  and  $loc'(n)$ :  $\bar{v} \in V$  (i.e., no obstruction) and  $\bar{v} \neq loc(j)$  for all  $j \in [N] \setminus \{n\}$  (i.e., no collision).

An *schedule*  $\kappa$  is an element of  $[N]^\omega$ . A set  $S$  of schedulers is called a *scheduler*. A *scheduler constraint* is a C-LTL formula  $\omega$  over the variable  $\{turn\}$ . It induces the set of schedules  $\kappa$  such that there exists  $\sigma \models \omega$  such that  $\sigma_i \models (turn = \kappa_i)$  for all  $i \in \mathbb{N}$ .<sup>7</sup>

**EXAMPLE 1.** Round-robin of  $N$  agents may be expressed as the  $N$ -scheduler consisting of the schedules  $(\pi(1) \dots \pi(N))^\omega$  such that  $\pi$  is a bijection of  $[N]$ . To express this scheduler in C-LTL one may use the formula  $\bigvee_\pi \mathsf{G} \bigwedge_n (turn = n) \rightarrow \mathsf{O}(turn = \pi(\pi^{-1}(n) + 1))$  where the disjunction is over all bijections  $\pi$  of  $[N]$ .

**EXAMPLE 2.** Fair scheduling of  $N$  agents may be expressed as the set of schedules satisfying  $\bigwedge_n \mathsf{GF}(turn = n)$ .

A *run*  $\rho$  of  $\mathcal{A}$  on  $G$  according to scheduler  $S$  is an infinite sequence  $\rho = \rho_1 \rho_2 \rho_3 \dots$  of configurations such that  $\rho_1$  is initial and there exists  $\kappa \in S$  such that  $\rho_i \vdash_{\kappa_i} \rho_{i+1}$ , for all  $i$ . The *agent locations* of the run is the sequence

<sup>7</sup>It is also possible to define more powerful schedulers by C-LTL formulae over the variables  $\{turn\} \cup svar \cup avar \cup bvar \cup ovar \cup \{l\}$ . For ease of exposition we refrain from doing so.

$loc(\rho_1)loc(\rho_2) \dots$ , the *beacon locations of the run* is the sequence  $locb(\rho_1)locb(\rho_2) \dots$ , and the *states of the run* is the sequence  $state(\rho_1)state(\rho_2) \dots$ . To every run  $\rho = \rho_1 \rho_2 \dots$  we associate  $val(\rho)$ , which is the sequence of valuations  $val_{\rho_1} val_{\rho_2} \dots$ .

**Agent Tasks.** Agents should achieve some task in their environment. A *task* is a C-LTL formula  $\tau$  over variables  $avar \cup svar \cup bvar \cup ovar \cup \{l\}$ . The ensemble  $\mathcal{A}$  *achieve the task  $\tau$  in environment  $G$  according to scheduler  $S$*  if for all runs  $\rho$  of  $\mathcal{A}$  on  $G$  according to scheduler  $S$ , the sequence of valuations  $val(\rho)$ , satisfies  $\tau$ . Note that the task may, if one wishes, restrict the initial states and positions of the agents. For example, agents may be required to start at the corners of the grid, etc.

**EXAMPLE 3.** Agents gather if eventually they arrive at the same, not previously determined, location [39]. This task can be expressed by the C-LTL formula  $\mathsf{F} \bigwedge \{x_{n,d} = x_{m,d} : n, m \in [N], d \in [D]\}$ .

**EXAMPLE 4.** The Line Formation task requires the set of agents to form a line, an example of a pattern-formation task [25]. This can be expressed by the C-LTL formula:  $\mathsf{F} \text{Line}$  where  $\text{Line} := \bigvee_{d \in [D]} \bigwedge \{x_{n,d} = x_{m,d} : n, m \in [N]\}$ . A variation asks that the agents repeatedly form a line, and can be written  $\mathsf{GF} \text{Lin}$ .

The following is inspired by the task of a guard making sure that the doors of all buildings on campus are locked.

### Example.

We consider the task of **border patrol** for a **single agent**. This task consists of moving through the grid such that every border, in our encoding *every edge of every obstacle*, is traversed at least once by the agent. Below we will give a specification for border patrol and describe a protocol for an agent that achieves border patrol. Our main positive result (Theorem 3) means that one can automatically verify this protocol against this specification for every parameterized grid environment satisfying the constraints of Theorem 3. For ease of exposition, we consider the 2-D setting where all obstacles are disjoint (disjointness can be encoded by a C-LTL-formula which is quadratic in the number of obstacles).

**Specification for border patrol.** We consider an obstacle described by SW corner  $(x_1, y_1)$  and NE  $(x_2, y_2)$ . For the edge between  $(x_1, y_1)$  and  $(x_1, y_2)$  (the bottom edge of the rectangle) we can encode by an *eventually formula* the agent arrives at the corner  $(x_1, y_1)$  or  $(x_1, y_2)$ . Then we can describe by an *until formula* that the agent will eventually reach the other corner of the edge while not leaving the edge on the way. Border control is specified by a conjunction of formulae as described above for every edge of every obstacle.

**Protocol for border control.** We assume that the agent starts at the origin (where the security personnel office is located), i.e., the SW corner of the grid. When the agent is at the left side of the grid it places a beacon on its position. Then the agent moves from the left of the grid to the right of the grid circling around every obstacle encountered on its way. After circling around an obstacle the agent uses the beacon to stay on the horizontal line defined by the coordinates of the beacon. When the agent has arrived at the right hand side of the grid it returns to the beacon. Then the agent moves upwards until it is on the same height with

the next obstacle, i.e., the agent and the SW corner of the first obstacle in the whole environment whose SW corner is at a larger  $y$  coordinate than its current position. The agent repeats this behavior until it reached the NE corner of the grid. This completes the description of the agent. It is interesting to note that the described agent can be implemented by a finite automaton with a single beacon whose size only depends linearly on the number of obstacles.

Observe that in the description above, the agent's move up until aligned with the SW corner of the next obstacle cannot be directly implemented in a model where the agent can only use vision to navigate (since that corner may be obstructed by other obstacles that are horizontally between the agent and that corner), and the example illustrates the power of our model. As noted before, one can obviously limit the agents to not use the full power of the model, and one can come up with border patrol protocols for vision/touch limited agents as well.

## 6. PARAMETERISED VERIFICATION

We first formalise the parameterised verification problem (PVP) for agents on parameterized grids with a bounded number of obstacles (as defined in the previous section) and show that it is decidable. The PVP depends on formulas for tasks, environments, and schedulers. Recall that  $\mathcal{G}_\phi$  is the (possibly infinite) set of environments determined by  $\phi$ , and  $S_\omega$  is the set of schedules determined by  $\omega$ .

**DEFINITION 1.** *The parameterised verification problem of Sliding Robots on Grids, written PVP, is the following decision problem: given an agent ensemble  $\mathcal{A}$ , a task  $\tau$  for the agents, an environment constraint  $\phi$  describing a parameterized grid environment, and scheduler constraint  $\omega$ , decide whether for every  $G \in \mathcal{G}_\phi$ , the agents  $\mathcal{A}$  achieve task  $\tau$  on environment  $G$  according to scheduler  $S_\omega$ .*

**THEOREM 3.** *The PVP is in PSPACE.*

**PROOF.** We show this by reducing the PVP to satisfiability of C-LTL which is in PSPACE [18]. I.e., we effectively transform the agent-ensemble  $\mathcal{A} = \langle A_1, \dots, A_N \rangle$ , environment constraint  $\phi$ , and scheduler constraint  $\omega$ , into a C-LTL formula  $\psi_{\mathcal{A}, \phi, \omega}$  whose models code all, and only, the runs of  $\mathcal{A}$  on environments  $G \in \mathcal{G}_\phi$  according to the scheduler  $S_\omega$ . Then we check that the C-LTL formula  $\psi_{\mathcal{A}, \phi, \omega} \rightarrow \tau$  is valid, or equivalently, that  $\neg(\psi_{\mathcal{A}, \phi, \omega} \rightarrow \tau)$  is not satisfiable.

Here are the details of the transformation. Say  $A_n = (Q_n, I_n, \delta_n)$ , and w.l.o.g. assume, for  $n \in [N], j \in Q_n$ , that  $Q_n = [[Q_n]]$ . The formula  $\psi_{\mathcal{A}, \phi}$  has variables  $\text{avar} \cup \text{ovar} \cup \{l\} \cup \{s_i : i \in [N]\} \cup \{\text{turn}\}$  and uses constants from the set  $\{0, 1, \dots, \max_n |Q_n|\}$ . We first define some helper formulas:

- $In(\bar{x}, \bar{u}, \bar{v})$  is  $\bigwedge_d u_d \leq x_d \leq v_d$  (position  $\bar{x}$  is within the rectangle determined by  $\bar{u}$  and  $\bar{v}$ );
- $\text{NotObstructed}$  is  $\bigwedge_n \bigwedge_k \neg In(\bar{x}_n, \bar{u}_k, \bar{v}_k)$  (no agent is inside any rectangle obstruction);
- $Btwn_d(\bar{x}, \bar{y}, \bar{z})$  is  $(x_d < y_d < z_d \vee z_d < y_d < x_d) \wedge \bigwedge_{e \neq d} z_e = y_e = x_e$  (position  $\bar{y}$  is between positions  $\bar{x}$  and  $\bar{z}$  and lie parallel to axis  $d$ );
- $O^1(\bar{z})$  is  $(O^1(z_1), \dots, O^1(z_n))$ .

Define  $\psi_{\mathcal{A}, \phi, \omega}$  as the conjunction of:

- $G \text{ NotObstructed}$  (agents are unobstructed);
- $\phi \wedge G[l = O^1(l) \wedge \bigwedge_k \bar{u}_k = O^1(\bar{u}_k) \wedge \bar{v}_k = O^1(\bar{v}_k)]$  (obstructions satisfy the environment constraint and do not change over time);
- $\bigwedge_n \bigvee_{j \in I_n} s_n = j$  (each agent starts in an initial state);
- $\omega \wedge G \bigwedge_n (\text{turn} \neq n) \rightarrow (\bar{x}_n = O^1(\bar{x}_n) \wedge s_n = O^1(s_n))$  (agents take turns according to a schedule in  $\omega$ );
- $G \bigwedge_n \bigwedge_{i \in B} (\text{turn} \neq n) \rightarrow (\bar{b}_n^i = O^1(\bar{b}_n^i))$  (only beacons of the agent whose turn it is can be moved);
- $\bigwedge_{n,j} G[\text{turn} = n \wedge s_n = j \rightarrow \bigvee_{t \in \delta_n} Next_t]$  (agents follow their protocols),

where  $Next_t$ , for  $t$  of the form  $(j, (d, \text{guard}), j')$ , is

$$\text{guard}' \wedge \text{move} \wedge O^1(s_n) = j' \wedge \text{nocollide} \wedge \text{beacon}$$

where  $\text{guard}'$  is  $\text{guard}$  with every primed agent variable  $x'_{m,e}$  (resp. beacon variable  $b'_{m,e}$ ) replaced by  $O^1(x_{m,e})$  (resp.  $O^1(b_{m,e})$ );  $\text{move}$  is  $\bigwedge_{e \neq d} x_{n,e} = O^1(x_{n,e})$ ;  $\text{nocollide}$  is  $\bigwedge_{m \neq n} \neg Btwn_d(\bar{x}_n, \bar{x}_m, O^1(\bar{x}_n))$ ; and  $\text{beacon}$  is  $\bigwedge_i \bar{b}_n^i \neq O^1(\bar{b}_n^i) \rightarrow O^1(\bar{b}_n^i) = O^1(\bar{x}_n)$  (the agent can move a beacon only to its new location).

It is not hard to check that for every sequence  $\sigma$  of valuations,  $\sigma \models \psi_{\mathcal{A}, \phi, \omega}$  if and only if there exists  $G \in \mathcal{G}_\phi$  and a run  $\rho$  of  $\mathcal{A}$  on  $G$  according to scheduler  $S_\omega$  such that (i)  $\sigma \upharpoonright \text{avar}$  is equal to  $\text{loc}(\rho)$ ; (ii)  $\sigma \upharpoonright \{s_n : n \in [N]\}$  is equal to  $\text{state}(\rho)$ ; (iii)  $\sigma \upharpoonright \text{ovar} \cup \{l\}$  is a sequence of identical valuations, each satisfying  $\phi$ .

Thus,  $\psi_{\mathcal{A}, \phi, \omega} \rightarrow \tau$  is not valid if and only if there exists a sequence  $\sigma$  satisfying  $\psi_{\mathcal{A}, \phi, \omega} \wedge \neg\tau$  if and only if there exists  $G \in \mathcal{G}_\phi$  and a run  $\rho$  of  $\mathcal{A}$  on  $G$  according to scheduler  $S_\omega$  such that the sequence  $\text{val}(\rho)$  of valuations of the run  $\rho$  satisfies  $\neg\tau$  if and only if the agents  $\mathcal{A}$  do not achieve the task  $\tau$  according to scheduler  $S_\omega$  on all environments from  $\mathcal{G}_\phi$ . This completes the proof.  $\square$

Before presenting a PSPACE lower-bound for the PVP, we need some definitions and a lemma.

Let  $\text{AP}$  be a set of atomic propositions, and let  $\text{AP}_{ig} := \text{AP} \cup \{\text{ignore}\}$ , where  $\text{ignore}$  is a new atomic proposition not in  $\text{AP}$ . Let  $\Sigma := 2^{\text{AP}}$  and  $\Sigma_{ig} := 2^{\text{AP}_{ig}}$ . Given a word  $w' \in \Sigma_{ig}^\omega$ , let  $\text{sub}(w)$  be the word obtained by deleting all letters in  $w$  that are not in  $\Sigma$  (i.e., which contain  $\text{ignore}$ ).

**LEMMA 1.** *Let  $\phi$  be an LTL formula over atomic propositions  $\text{AP}$ . One can compute in polynomial time an LTL formula  $\phi'$  over  $\text{AP}_{ig}$ , such that for every  $w' \in \Sigma_{ig}^\omega$  we have that  $w' \models \phi'$  iff  $\text{sub}(w)$  is infinite and models  $\phi$ .*

**PROOF.** W.l.o.g., we assume that  $\phi$  is in positive normal form (i.e., negations are pushed to the atoms). The required formula is the conjunction of two formulas: the first requires that  $\neg\text{ignore}$  holds infinitely often (thus ensuring that if  $w' \models \phi$  then  $\text{sub}(w)$  is infinite), and the second “simulates”  $\phi$  on the points where  $\neg\text{ignore}$  holds (and “skipping” the points in which  $\text{ignore}$  holds). Formally,  $\phi' := \hat{\phi} \wedge \text{GF}(\neg\text{ignore})$ , where  $\hat{\phi}$  is constructed by induction on the structure of  $\phi$  as follows:

- if  $\phi := t$ , where  $t$  is an atomic proposition or its negation, then  $\hat{\phi} := \text{ignore} \cup (\neg\text{ignore} \wedge t)$ ;

- if  $\phi := \phi_1 \bowtie \phi_2$ , where  $\bowtie \in \{\vee, \wedge\}$ , then  $\widehat{\phi} := \widehat{\phi_1} \bowtie \widehat{\phi_2}$ ;
- if  $\phi := \mathbf{O} \phi_1$  then  $\widehat{\phi} := \text{ignore } \mathbf{U}(\neg \text{ignore} \wedge \mathbf{O} \widehat{\phi_1})$ ;
- if  $\phi := \phi_1 \mathbf{U} \phi_2$  then  $\widehat{\phi} := (\text{ignore} \vee \widehat{\phi_1}) \mathbf{U} (\neg \text{ignore} \wedge \widehat{\phi_2})$ ;
- if  $\phi := \phi_1 \mathbf{R} \phi_2$  then  $\widehat{\phi} := (\neg \text{ignore} \wedge \widehat{\phi_1}) \mathbf{R} (\text{ignore} \vee \widehat{\phi_2})$ .

This completes the proof.  $\square$

**THEOREM 4.** *PVP is PSPACE-hard already for the case of a single agent on singleton parameterized environments with no obstacles and no beacons.*

**PROOF.** We reduce the problem of validity of LTL formulas, which is PSPACE-hard, to the PVP. Let  $\phi$  be an LTL formula over the atomic propositions (w.l.o.g.)  $\mathbf{AP} := \{1, \dots, D\}$ . Consider the parameterized environment  $G$  containing the single  $D$ -dimensional grid with sides of length 1 (i.e., the discrete  $d$ -dimensional unit hypercube), with no obstacles. Note that every position on this grid is a vector  $\bar{x} \in \{0, 1\}^D$ . Consider a single agent  $A$  with two states  $\top, \perp$ . The transition relation allows the agent, from each state  $\top, \perp$ , to transition to either  $\top, \perp$  while moving in any direction (or staying in the same place).

A configuration  $c$  of  $A$  on  $G$  encodes a set  $X \in \Sigma_{ig}$  of atomic propositions in  $\mathbf{AP}_{ig}$  as follows:  $i \in [D]$  is in  $X$  iff the  $i$ 'th coordinate of the location of  $A$  is 1, and  $\text{ignore} \in X$  iff the local state of  $A$  is  $\perp$ . Hence, by the definition of  $A$ , for every word  $w' \in \Sigma_{ig}^\omega$  there is some run of  $A$  on  $G$  that encodes  $w'$ , and vice-versa. Also, using the same encoding (of atomic propositions of  $\mathbf{AP}_{ig}$  as constraints on the location/state of the agent) we can write, with a linear blowup, a task  $\tau$  that is a C-LTL formula that encodes the LTL formula  $\phi'$  we obtain from  $\phi$  using Lemma 1.

By Lemma 1,  $\phi$  is valid iff  $\phi'$  is valid. By the reasoning above, the later is true iff the output of the PVP is true on the input: agent  $A$ , a formula describing the hypercube  $G$ , and the task  $\tau$  (and the unrestricted scheduler true).  $\square$

Observe that, in the proof above, the location of the robot encodes the values of all atomic propositions in parallel, which is the reason  $D$  dimensions are used. However, we can make do instead with a single dimension if we encode the values in serial. I.e., if we partition the positions (over time) of the robot into blocks of length  $D$ , and let the  $j$  position inside a block encode the value of the  $j$ 'th atomic proposition at the time corresponding to the block number. I.e., for  $i \in \mathbb{N}_0$ , and  $1 \leq j \leq D$ , the position of the robot at time  $iD + j$  encodes the value of atomic proposition number  $i$  at time  $j$ . By suitably modifying  $\phi'$  and  $\widehat{\phi}$  to this new encoding we can deduce that the problem remains PSPACE-hard also in the interesting cases of 2 and 3-dimensional grids.

## 7. DISCUSSION

Parameterized verification of multi-agent systems is a hard problem, and the case of most interest, i.e., that of 2 and 3-dimensional grids, even more so. Indeed, in [2] it is shown that (for agents with precise positioning) while the problem is decidable for many parameterized environments it is undecidable even for agents working on a 1-D grid with only collision-detection abilities. In this work we gave one way to regain decidability. We have presented a model for describing multiple agents moving in parameterised grid environments of arbitrary size with a fixed number of obstacles

of arbitrary sizes. We have shown that relaxing the requirement that agents can not specify exact absolute positioning or step size, or the requirement of having a bounded number of obstacles, results in undecidability of the verification problem of very simple tasks of even a single agent. Our model of agents is very powerful (without becoming undecidable), and it generalises many other models (such as vision and touch based agents).

Many works in the past reason about environments by using all kinds of abstractions into graphs [9, 10, 44, 46]. Our approach avoids such abstractions (except for discretising space) by directly reasoning about the grid environments. We believe that our choice of using C-LTL as a logic for specifying both agents, parameterised environments, and tasks, allows one to encode things in a direct and natural way, and yields an elegant automatic verification algorithm. We draw an analog to the case of timed-automata, which are extremely popular because they provide users with direct and natural modeling formalism. Indeed, timed-automata can be verified by a sound and complete "region abstraction" to finite graphs [1, 22, 23], but working directly with these graphs is not convenient. A similar abstraction possibly exists for our framework, although it is not immediately clear or obvious how to define an abstraction with multiple agents, especially since they can stop "in the middle" of edges, and one can speak of their relative positions. Many forms of abstraction are not sound/complete. Thus, they do not allow one to map the border between decidability and undecidability as we do. Also, because an abstraction "throws away information", it must be tailor-made depending on the property to be verified.

**Future Work.** As mentioned above, one possible task is to try and come up with a form of "region abstraction" of our framework that is both sound and complete.

It is interesting to note that one can modify Theorem 1 to the case that the agent has access to absolute positioning with some error (i.e.,  $\pm \epsilon$ , as would be provided by a real-world GPS system). However, it is not clear if it can be modified to the case of an agent that has no global positioning but can measure its step size with some error. This suggests that one may come up with a model for such agents that has a decidable parameterised verification problem (with or without relative positioning).

Two other natural directions of future work are the following. First, one can investigate in practice the actual performance of the presented framework by implementing the proposed algorithms. Although the complexity is PSPACE in general, one can hope — as it is with many algorithms in formal verification — that in many natural cases the algorithm may exhibit acceptable performance. Another possible direction is to try to extend this framework to get decidability results (with reasonable complexity) for multi-robot system scenarios that are rich enough to capture protocols found in the distributed computing literature, e.g., [5, 25, 26, 38].

## Acknowledgments

B. Aminof and F. Zuleger are supported by the Austrian National Research Network S11403-N23 (RiSE) of FWF and by WWTF through grant ICT12-059. S. Rubin is a Marie Curie fellow of the Istituto Nazionale di Alta Matematica. A. Murano is partially supported by the GNCS 2016 project: Logica, Automi e Giochi per Sistemi Auto-adattivi.

## REFERENCES

- [1] R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking in dense real-time. *Inf. Comput.*, 104(1):2–34, 1993.
- [2] B. Aminof, A. Murano, S. Rubin, and F. Zuleger. Verification of asynchronous mobile-robots in partially-known environments. In *PRIMA*, LNCS 9387, pages 185–200, 2015.
- [3] E. M. Barrameda, S. Das, and N. Santoro. Uniform dispersal of asynchronous finite-state mobile robots in presence of holes. In *ALGOSENSORS*, LNCS 8243, pages 228–243, 2013.
- [4] M. A. Bender, A. Fernandez, D. Ron, A. Sahai, and S. Vadhan. The power of a pebble: Exploring and mapping directed graphs. *I&C*, 176(1):1–21, 2002.
- [5] M. A. Bender and D. K. Slonim. The power of team exploration: Two robots can learn unlabeled directed graphs. Technical report, MIT, 1995.
- [6] R. Bloem, S. Jacobs, A. Khalimov, I. Konnov, S. Rubin, H. Veith, and J. Widder. *Decidability of Parameterized Verification*, volume 6 (1) of *Synthesis Lectures on Distributed Computing Theory*. 2015.
- [7] A. Blum, P. Raghavan, and B. Schieber. Navigating in unfamiliar geometric terrain. *SIAM J. Comput.*, 26(1):110–137, 1997.
- [8] M. Blum and C. Hewitt. Automata on a 2-dimensional tape. *SWAT (FOCS)*, pages 155–160, 1967.
- [9] R. Brafman, J. Latombe, Y. Moses, and Y. Shoham. Applications of a logic of knowledge to motion planning under uncertainty. *J.ACM*, 44(5):633–668, 1997.
- [10] W. Burgard, M. Moors, C. Stachniss, and F. Schneider. Coordinated multi-robot exploration. *IEEE Transactions on Robotics*, 21(3):376–386, 2005.
- [11] A. Canedo-Rodríguez, V. Alvarez-Santos, C. V Regueiro, X. M. Pardo, and R. Iglesias. Multi-agent system for fast deployment of a guide robot in unknown environments. *Journal of Physical Agents*, 6(1):31–41, 2012.
- [12] K. Cheng and P. Dasgupta. Coalition game-based distributed coverage of unknown environments by robot swarms. In *AAMAS*, pages 1191–1194, 2008.
- [13] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT, 2002.
- [14] R. Cohen, P. Fraigniaud, D. Ilcinkas, A. Korman, and D. Peleg. Label-guided graph exploration by a finite automaton. In *Automata, Languages and Programming*, LNCS 3580, pages 335–346. 2005.
- [15] S. Das. Mobile agents in distributed computing: Network exploration. *Bull. EATCS*, 109:54–69, 2013.
- [16] G. De Giacomo, P. Felli, F. Patrizi, and S. Sardiña. Two-player game structures for generalized planning and agent composition. In *AAAI*, 2010.
- [17] S. Demri and D. D’Souza. An automata-theoretic approach to constraint LTL. *Inform. and Comp.*, 205(3):380 – 415, 2007.
- [18] S. Demri and R. Gascon. Verification of qualitative Z constraints. *Theor. Comput. Sci.*, 409(1):24–40, 2008.
- [19] A. Dessmark and A. Pelc. Optimal graph exploration without good maps. In *ESA*, LNCS 2461, pages 374–386. Springer, 2002.
- [20] K. Diks, P. Fraigniaud, E. Kranakis, and A. Pelc. Tree exploration with little memory. *Journal of Algorithms*, 51(1):38–63, 2004.
- [21] J. Engelfriet and H. J. Hoogeboom. Nested pebbles and transitive closure. In *STACS*, pages 477–488, 2006.
- [22] M. Faella, S. La Torre, and A. Murano. Dense real-time games. In *LICS*, pages 167–176. IEEE Computer Society, 2002.
- [23] M. Faella, S. La Torre, and A. Murano. Automata-theoretic decision of timed games. *Theor. Comput. Sci.*, 515:46–63, 2014.
- [24] A. Felner, R. Stern, and S. Kraus. PHA\*: performing A\* in unknown physical environments. In *AAMAS*, pages 240–247. ACM, 2002.
- [25] P. Flocchini, G. Prencipe, and N. Santoro. Computing by mobile robotic sensors. In *Theoretical Aspects of Distributed Computing in Sensor Networks*, EATCS, pages 655–693. 2011.
- [26] P. Flocchini, G. Prencipe, and N. Santoro. *Distributed Computing by Oblivious Mobile Robots*. Synthesis Lectures on Distributed Computing Theory. 2012.
- [27] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. *Theoretical Computer Science*, 345:331–344, 2005.
- [28] L. Gasieniec and T. Radzik. Memory efficient anonymous graph exploration. In *Graph-Theoretic Concepts in Computer Science*, LNCS 5344, pages 14–29. Springer, 2008.
- [29] A. Gilboa, A. Meisels, and A. Felner. Distributed navigation in an unknown physical environment. In *AAMAS*, pages 553–560, 2006.
- [30] F. Hoffmann. One pebble does not suffice to search plane labyrinths. In *Fundamentals of Computation Theory*, LNCS 117, pages 433–444. 1981.
- [31] T. Hsiang, E. M. Arkin, M. A. Bender, S. P. Fekete, and J. S. B. Mitchell. Algorithms for rapidly dispersing robot swarms in unknown environments. In *WAFR*, Springer Tracts in Advanced Robotics, vol. 7, pages 77–94. Springer, 2002.
- [32] Y. Hu and G. De Giacomo. Generalized planning: Synthesizing plans that work for multiple environments. In *IJCAI*, pages 918–923, 2011.
- [33] K. Hungerford, P. Dasgupta, and K. R. Guruprasad. Distributed, complete, multi-robot coverage of initially unknown environments using repartitioning. In *AAMAS*, pages 1453–1454. IFAAMAS, 2014.
- [34] A. Khalimov, S. Jacobs, and R. Bloem. PARTY parameterized synthesis of token rings. In *CAV*, LNCS 8044, pages 928–933, 2013.
- [35] A. Khalimov, S. Jacobs, and R. Bloem. Towards efficient parameterized synthesis. In *VMCAI*, LNCS 7737, pages 108–127, 2013.
- [36] P. Kouvaros and A. Lomuscio. Automatic verification of parameterised multi-agent systems. In *AAMAS*, pages 861–868, 2013.
- [37] P. Kouvaros and A. Lomuscio. A counter abstraction technique for the verification of robot swarms. In *AAAI*, pages 2081–2088, 2015.
- [38] E. Kranakis, D. Krizanc, and S. Rajsbaum. Mobile agent rendezvous: A survey. In *SIROCCO*, LNCS 4056, pages 1–9, 2006.
- [39] E. Kranakis, D. Krizanc, and S. Rajsbaum.

- Computing with mobile agents in distributed networks. In *Handbook of Parallel Computing: Models, Algorithms, and Applications. Chapter 8*. 2007.
- [40] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
  - [41] D. K. M. Blum. On the power of the compass (or, why mazes are easier to search than graphs). In *FOCS*, pages 132–142, 1978.
  - [42] M. L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., 1967.
  - [43] C. H. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theor. Comput. Sci.*, 84(1):127–150, 1991.
  - [44] N. Rao, S. Karet, W. Shi, and S. Iyengar. *Robot navigation in unknown terrains: Introductory survey of non-heuristic algorithms*. Jul 1993.
  - [45] S. Rubin. Parameterised verification of autonomous mobile-agents in static but unknown environments. In *AAMAS*, pages 199–208, 2015.
  - [46] K. Senthilkumar and K. Bharadwaj. Multi-robot exploration and terrain coverage in an unknown environment. *Robotics and Autonomous Systems*, 60(1):123–132, 2012.

## **4.8 Publication**

The rest of this page is intentionally left blank

# Prompt Alternating-Time Epistemic Logics

**Benjamin Aminof**

Technische Universität Wien

**Aniello Murano and Sasha Rubin**

Università di Napoli "Federico II"

**Florian Zuleger**

Technische Universität Wien

## Abstract

In temporal logics, the operator  $F$  expresses that at some time in the future something happens, e.g., a request is eventually granted. Unfortunately, there is no bound on the time until the eventuality is satisfied which in many cases does not correspond to the intuitive meaning system designers have, namely, that  $F$  abstracts the idea that there is a bound on this time although its magnitude is not known. An elegant way to capture this meaning is through Prompt-LTL, which extends LTL with the operator  $F_P$  ("prompt eventually"). We extend this work by studying alternating-time epistemic temporal logics extended with  $F_P$ .

We study the model-checking problem of the logic Prompt-KATL\*, which is ATL\* extended with epistemic operators and prompt eventually. We also obtain results for the model-checking problem of some of its fragments. Namely, of Prompt-KATL (ATL with epistemic operators and prompt eventually), Prompt-KCTL\* (CTL\* with epistemic operators and prompt eventually), and finally the existential fragments of Prompt-KATL\* and Prompt-KATL.

## Introduction

Alternating-time temporal logics are expressive tools for reasoning about multi-agent systems (Alur, Henzinger, and Kupferman 2002; van der Hoek and Wooldridge 2002; Chatterjee, Henzinger, and Piterman 2010; Mogavero et al. 2014). These powerful logics allow one to express individual or common goals of the agents throughout time, as well as specify the interactions among the agents (cooperation or adversarial). *Model checking* (Clarke and Emerson 1981; Quelle and Sifakis 1981) specifications written in these logics allows one to verify the correct behavior of multi agent systems using recently developed practical automatic tools (Lomuscio, Qu, and Raimondi 2009; Čermák et al. 2014; Čermák, Lomuscio, and Murano 2015).

A pioneering logic in this field is *Alternating-Time Temporal Logic* (ATL\*) and its fragments ATL (Alur, Henzinger, and Kupferman 2002), and CTL\* (Emerson and Halpern 1986). ATL\* formulas are usually interpreted over *concurrent game structures* (CGS), which are labeled-state transition-systems with the ability of modeling the interaction among agents. For example, in a system with multiple

agents and shared resources, the fact that a set of agents  $A$  can ensure that, regardless of the actions of the other agents, every request to access a resource is eventually granted, can be expressed by the ATL\* formula  $\langle\!\langle A \rangle\!\rangle G(req \rightarrow F grant)$ .

A crucial shortcoming in real-life temporal-logic verification, deeply ingrained in the definition of linear-temporal logic (LTL), is that the satisfaction of a formula like  $F grant$  implies no *a priori* bound on when the *grant* occurs. I.e., the system may admit executions with longer and longer delays before the *grant*, and yet still satisfy the formula  $F grant$ . Replacing the above formula with a formula specifying that the *grant* should occur within some fixed number of steps (say 3) is usually not an option, since one usually does not know the maximal delay that should be expected. This fact, that the  $F$  operator of temporal-logics fails to capture the intuitive meaning of "within some bounded amount of time" has motivated the introduction of an extension of LTL, called "prompt-LTL", in (Kupferman, Piterman, and Vardi 2009; Alur et al. 2001) that includes a new operator  $F_P$  called "prompt eventually". The semantics of  $F_P \phi$  is such that it is satisfied only if there is some bound  $k$ , which is shared by all behaviours/computations of the system, such that whenever  $F_P \psi$  should hold at some point along a computation then  $\psi$  holds within at most  $k$  steps. (Kupferman, Piterman, and Vardi 2009) goes on to show that prompt-LTL model checking is not more costly and slightly more complicated than LTL-model checking. It is important to note that prompt-LTL formulas are in positive normal form (i.e., with negations pushed all the way to the atoms), but it does not include the dual operator  $G_P$  of the  $F_P$  operator (however, the operator  $G$  is included). As argued in (Kupferman, Piterman, and Vardi 2009), on the one hand  $G_P$  is less useful than  $F_P$  (its meaning is that there is some global bound  $k$  such that whenever  $G_P \psi$  should hold then  $\psi$  holds for at least  $k$  steps, and we do not care afterwards), and on the other hand adding it to the logic seriously complicates the decision procedures.

Since ATL\* inherits its temporal operators from LTL it is natural to consider extending it with the  $F_P$  operator, thus allowing one to specify that eventualities should not be delayed for an unbounded number of steps<sup>1</sup>. We believe that, in a multi-agent setting, the need for the  $F_P$  operator is ar-

<sup>1</sup>It is an intriguing open question whether one can write in ATL\* (or CTL\*) a formula that is equivalent to  $F_P$ .

guably even more natural than for closed single-agent systems (for which LTL is suited) as it allows one to specify that certain agents should not have the power to unboundedly delay other agents. Hence, we extend ATL\* to include the  $F_P$  operator. We combine this with the extension of ATL\* with epistemic operators that allow one to express what different agents know in a setting with imperfect information.

Reasoning about agents' strategies in open system verification may require to act under partial information about the states of the system (Aminof, Murano, and Vardi 2007; Jamroga and Ågotnes 2007; Aminof et al. 2013; Bulling and Jamroga 2014; Huang and van der Meyden 2014; Jamroga and Murano 2015). In many practical scenarios such as web-banking attacks, card games, market auctions, etc., agents have indeed a limited observability about the state content. One may think of states containing some private information that are visible only to a (possibly empty) subset of players (Reif 1984; Kupferman and Vardi 1997a). Each agent chooses his strategy based on what he can observe. To work with incomplete information systems, the syntax and the semantics of ATL\* has been properly extended with epistemic operators (van der Hoek and Wooldridge 2002). The resulting logic is known as KATL\*, sometimes simply called ATL\* with *imperfect information*.

**Our contribution** We address the question of verifying prompt branching-time specifications in multi-agent systems for the first time. We present an extension of KATL\* with the prompt eventually temporal operator, called Prompt-KATL\*. We study the model-checking problem of this logic and some of its fragments. Namely, of Prompt-KATL (ATL with epistemic operators and prompt eventually), Prompt-KCTL\* (CTL\* with epistemic operators and prompt eventually), and finally the existential fragments of Prompt-KATL\* and Prompt-KATL. We show that, for the case of perfect information, model-checking is decidable and not harder than for these logics without the prompt eventually operator. For the case of imperfect information, note that model checking of ATL\*, and even ATL, over concurrent game structures with more than two agents and imperfect information is undecidable ((Pnueli and Rosner 1989; Dima and Tiplea 2011)). Moreover, by (Vester 2013), this is already the case for the existential fragment of ATL. However, we show that the imperfect information case is always decidable for Prompt-KCTL\*, and for Prompt-KATL\* and Prompt-KATL it is decidable in the following two cases: (i) the players are constrained to memoryless strategies, or (ii) the players use memory-full but cooperative strategies and one restricts to the existential fragments of Prompt-KATL\* and Prompt-KATL. Furthermore, in all cases, the complexity of our procedures is as good as for the non-prompt version of these logics.<sup>2</sup>

**Related work** In (Alur et al. 2001), the authors introduce a parameterised extension of LTL in which the temporal operators are associated with variables in order to count the

<sup>2</sup>Except for the case of Prompt-KATL with memoryless strategies for which we only show membership in PSPACE.

steps between successive occurrences of different events. A fragment of this logic is closely studied in (Kupferman, Piterman, and Vardi 2009), i.e., the extension of LTL by the prompt eventuality operator  $F_P$ , called *Prompt LTL*. In (Almagor, Hirshfeld, and Kupferman 2010) the automata-theoretic counterpart of the  $F_P$  operator has been also introduced and studied.

Only recently has prompt LTL been studied outside the realm of closed systems. (Zimmermann 2013) studies two-player turn-based games of perfect information with respect to prompt LTL. (Chatterjee, Henzinger, and Horn 2009) lift the prompt semantics to  $\omega$ -regular games, under the parity winning condition, by introducing *finitary parity* games. They make use of the concept of “*distance*” between positions in a play that refers to the number of edges traversed in the game arena. The classical parity winning condition is then reformulated to take into consideration only those states occurring with a bounded distance. This idea has also been generalised to deal with more involved prompt parity conditions (Fijalkow and Zimmermann 2012; Mogavero, Murano, and Sorrentino 2013). Finally, from a practical point of view, one can see connections with bounded model checking of open systems. Indeed, a central question there is whether a model satisfies a given formula by looking at the model up to depth  $k$ . We refer to (Huang, Luo, and Van Der Meyden 2011) for an overview.

Due space limitation, most of the proofs are just sketched.

## Definitions

### Basic Notation

We denote the set of integers by  $\mathbb{N}$ , and write  $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$ . For a set  $\Gamma$ , we write  $\Gamma^\omega$  (resp.  $\Gamma^*$ ) for the set of infinite (resp. finite) sequences (also called words) of elements in  $\Gamma$ , and  $\Gamma^+$  for the non-empty finite sequences. We count positions in a sequence starting with 0, and write  $w_i$  for the  $i$ 'th element (called *letter*) of a word  $w$ . The length (in  $\mathbb{N}_0 \cup \{\infty\}$ ) of  $w$  is written  $|w|$ . The suffix  $w_i w_{i+1} \dots$  of  $w$  is written  $w_{\geq i}$  and  $w_{\leq i}$  is the prefix  $w_0 \dots w_i$  of  $w$ . We usually write  $a$  instead of the singleton set  $\{a\}$ . Given  $n \in \mathbb{N}$ , we consider a function  $f$  with domain  $\{1, \dots, n\}$  as a vector with  $n$  coordinates. Thus we may write  $f = (f_1, \dots, f_n)$ , and use  $f(i)$  and  $f_i$  interchangeably.

### Game Structures

As for KATL\*, models of Prompt-KATL\* are *Imperfect Information Concurrent Game Structures (iCGS)*, i.e., structures of the form  $S = \langle Ag, AP, Act, S, \lambda, \delta, \{\sim_a : a \in Ag\} \rangle$  where:

- $Ag$  is a finite non-empty set of *agents* (also called *players*);
- $AP$  is a finite non-empty set of atoms;  $Act$  is a finite non-empty set of *actions* for the agents;
- $S$  is the set of *states* of the game structure;
- $\lambda : S \rightarrow 2^{AP}$  is a *labeling* function that assigns to a state  $s$  the set  $\lambda(s)$  of atoms that hold in that state;
- $\delta : S \times Act^{Ag} \rightarrow S$  is a *transition* function that assigns to every state, and every choice of actions — one for each agent — a successor state;

- and  $\sim_a \subseteq S \times S$  is an equivalence relation representing the imperfect information of agent  $a$ , i.e.,  $s \sim_a s'$  means that agent  $a$  cannot distinguish between  $s$  and  $s'$ .

The equivalence-classes of  $\sim_a$  are called the *observation sets* of agent  $a$ . The set  $Act^{Ag}$  is called the set of *decisions*. An iCGS is called *finite* if the set  $S$  is finite. An iCGS has *perfect information* if for every  $a \in Ag$  we have that  $\sim_a$  is the equality relation, i.e., if  $s \sim_a s'$  implies that  $s = s'$ . In this case it may be written CGS.

Observe that we assume that all agents use the same set of actions  $Act$ . However, it is sometimes convenient to assume that each agent  $a$  uses only some non-empty subset  $Act_a \subseteq Act$  of actions (one can simply define the transition relation to have all actions in  $Act \setminus Act_a$  duplicate the effect of some action in  $Act_a$ )<sup>3</sup>.

**Computations and Strategies.** A path in  $S$  is a finite or infinite sequence  $\pi_0\pi_1\cdots \in S^\omega \cup S^+$  such that for all  $i$  there exists a decision  $d \in Act^{Ag}$  such that  $\pi_{i+1} = \delta(\pi_i, d)$ . We call finite paths *histories*, and infinite ones *computations* or *plays*. The set of computations in  $S$  is written  $cmp(S)$ , and the set of computations in  $S$  that start with  $s$  is written  $cmp(S, s)$ . We define  $hist(S)$  and  $hist(S, s)$  similarly.

A *strategy* (for a single agent) is a function  $\sigma : hist(S) \rightarrow Act$ . For a non-empty set  $A \subseteq Ag$  of agents, and a strategy  $\sigma_a$  for each  $a \in A$ , write  $\Sigma_A := \{\sigma_a : a \in A\}$  for the set of strategies. A path  $\pi$  is *consistent with*  $\Sigma_A$  if it can be obtained by having the agents in  $A$  follow their strategies in  $\Sigma_A$ , i.e., if for every position  $\pi_i$  of  $\pi$  there exists  $d \in Act^{Ag}$  such that: i)  $\pi_{i+1} \in \delta(\pi_i, d)$  and; ii) for every  $a \in A$ ,  $d(a) = \sigma_a(\pi_{\leq i})$ . The set of computations starting with  $s$  that are consistent with  $\Sigma_A$ , written  $out(s, \Sigma_A)$ , is called the set of *outcomes* of  $\Sigma_A$  from  $s$ .

For  $A \in Ag$ , we derive the following equivalence relations:  $\sim_A := \cap_{a \in A} \sim_a$ , and  $\sim_A^C := (\cup_{a \in A} \sim_a)^*$ , where  $*$  denotes the transitive closure (with respect to composition). Note that  $\sim_{\{a\}} = \sim_a$ , and we use these interchangeably.

Extend  $\sim_A$  to histories point-wise: if  $hs$  and  $h's'$  are histories and  $h \sim_A h'$  and  $s \sim_A s'$  then  $hs \sim_A h's'$ . A strategy  $\sigma$  is *observational for agent a* if for all  $h \sim_a h'$ , we have  $\sigma(h) = \sigma(h')$ . A set  $\Sigma_A$  of strategies for the agents in  $A$  is *cooperatively observational* if for all  $h \sim_A h'$  and  $a \in A$ , we have  $\sigma_a(h) = \sigma_a(h')$ .<sup>4</sup>

**2-Player Games.** The proofs of Propositions 3 and 4 use the following notions. A *2-player concurrent game* is a pair  $\langle S, \Upsilon \rangle$  where  $S$  is an iCGS (called an *arena*) with two players (usually  $Ag = \{0, 1\}$ ), and  $\Upsilon \subseteq cmp(S)$  is a *goal*. A play  $\pi \in cmp(S)$  is *won* by Player 0 if  $\pi \in \Upsilon$ , and is won by Player 1 otherwise. Given  $s \in S$ , an observational strategy  $\sigma_i$  for Player  $i \in \{0, 1\}$  is called *winning from s* (and we say that Player  $i$  *wins from s*) if all plays starting in  $s$  that are consistent with  $\sigma_i$  are won by Player  $i$  (i.e., if  $out(s, \sigma_i) \subseteq \Upsilon$ ). An LTL formula  $\psi$  induces a goal  $\Upsilon := \{\pi \in cmp(S) \mid$

<sup>3</sup>More formally, require that for every agent  $a$ , there is some action  $\alpha \in Act_a$ , such that for every  $d \in Act^{Ag}$  and  $s \in S$ , we have: if  $d(a) \in Act \setminus Act_a$  then  $\delta(s, d) = \delta(s, d')$ , where  $d'$  is obtained from  $d$  by letting  $d'(a) = \alpha$ .

<sup>4</sup>Agents using cooperative strategies are sometimes referred to as having *distributive knowledge*.

$\pi \models \psi\}$ , and we usually just say that the game has goal  $\psi$ . If Player 0 wins from  $s$  in the game with goal  $\psi$  we say that he *can enforce*  $\psi$  from  $s$ .

### Syntax of Prompt-KATL\*.

Fix a finite set of *atomic propositions (atoms)* AP, and a finite set of *agents* Ag. The *Prompt-KATL\** state ( $\varphi$ ) and path ( $\psi$ ) formulas over AP and Ag are built using the following context-free grammar:

$$\begin{aligned}\varphi ::= & p \mid \neg p \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \langle\!\langle A \rangle\!\rangle \psi \mid [[A]] \psi \mid \\ & \mathbb{K}_a \varphi \mid \mathbb{D}_A \varphi \mid \mathbb{C}_A \varphi \mid \widetilde{\mathbb{K}}_a \varphi \mid \widetilde{\mathbb{D}}_A \varphi \mid \widetilde{\mathbb{C}}_A \varphi\end{aligned}$$

and

$$\psi ::= \varphi \mid \psi \vee \psi \mid \psi \wedge \psi \mid X \psi \mid \psi U \psi \mid \psi R \psi \mid F_P \psi$$

where  $p$  varies over AP,  $A$  varies over subsets of Ag, and  $a$  over elements of Ag. The class of *Prompt-KATL\** formulas is the set of state formulas generated by the grammar.

The *temporal operators* are  $X$  (next),  $U$  (until),  $R$  (releases, the dual of until) and  $F_P$  (prompt eventually); the *strategy quantifiers* are  $\langle\!\langle A \rangle\!\rangle$ ,  $[[A]]$ , where  $\langle\!\langle A \rangle\!\rangle \psi$  is read “the agents in  $A$  can enforce  $\psi$ ”, and its dual  $[[A]] \psi$  is read “the agents in  $A$  cannot avoid  $\psi$ ”; and the *epistemic operators* are  $\mathbb{K}_a$  (agent  $a$  knows that),  $\mathbb{D}_A$  (the agents in  $A$  distributively know that), and  $\mathbb{C}_A$  (amongst the agents in  $A$  it is common knowledge that), as well as the dual operators  $\widetilde{\mathbb{K}}_a$ ,  $\widetilde{\mathbb{D}}_A$ ,  $\widetilde{\mathbb{C}}_A$ . Observe that, as discussed in the introduction, we do not include a dual operator to  $F_P$ . Also note that if one removes the prompt eventually operator then the grammar above generates exactly all the formulas of KATL\* in positive normal form (i.e., with negations pushed all the way to the atoms).

We have the usual syntactic sugar: we write  $F \varphi$  (eventually) instead of true  $U \varphi$ ; (globally)  $G \varphi$  instead of false  $R \varphi$ ;  $E$  (exists) instead of  $\langle\!\langle Ag \rangle\!\rangle$ , and  $A$  (for all) instead of  $[[Ag]]$ ;  $\mathbb{E}$  (everybody knows) instead of  $\wedge_{a \in Ag} \mathbb{K}_a$  (and its dual  $\widetilde{\mathbb{E}}$  for  $\vee_{a \in Ag} \widetilde{\mathbb{K}}_a$ ). Finally, we use a shorthand for repeated next:  $X^k$  (for  $k \in \mathbb{N}$ ) is defined as:  $X^1 := X$ , and  $X^{k+1} := XX^k$ .

We now define some important syntactic fragments.

1. Prompt-ATL\* formulas consists of the formulas of Prompt-KATL\* in which no epistemic operator occurs.
2. Prompt-ATL formulas consists of the formulas of Prompt-ATL\* in which every temporal operator is immediately preceded by a strategy quantifier.
3. Prompt-KCTL\* is obtained from Prompt-KATL\* by only allowing strategy quantifiers of the form E and A.
4. Prompt-LTL is the class of path formulas generated by the grammar above in which no strategy quantifier or epistemic operator appears.
5. The *existential fragments* of Prompt-KATL\* and Prompt-KATL consist of those formulas in which the  $[[A]]$  quantifier does not occur.

### Semantics of Prompt-KATL\*.

We first define the semantics of KATL\* (i.e., formulas that don't mention the prompt operator  $F_P$ ), and then define the semantics of Prompt-KATL\*.

**Semantics of KATL<sup>\*</sup>.** The satisfaction relation  $\models$  is defined inductively, as usual. Formally, for all  $s \in S, p \in AP$ :

- $(S, s) \models p$  iff  $p \in \lambda(s)$ , and  $(S, s) \models \neg p$  iff  $p \notin \lambda(s)$ .
- $(S, s) \models \varphi_1 \wedge \varphi_2$  iff  $(S, s) \models \varphi_1$  and  $(S, s) \models \varphi_2$ .
- $(S, s) \models \varphi_1 \vee \varphi_2$  iff  $(S, s) \models \varphi_1$  or  $(S, s) \models \varphi_2$ .
- $(S, s) \models \langle\langle A\rangle\rangle\psi$  iff there exists a set of observational strategies  $\Sigma_A$ , one for each agent in  $A$ , s.t.  $(S, \pi) \models \psi$  for all  $\pi \in out(s, \Sigma_A)$ .
- $(S, s) \models [[A]]\psi$  iff for every set of observational strategies  $\Sigma_A$ , one for each agent in  $A$ , there is a computation  $\pi \in out(s, \Sigma_A)$  s.t.  $(S, \pi) \models \psi$ .
- $(S, s) \models \mathbb{K}_a\varphi$  (resp.  $\widetilde{\mathbb{K}}_a\varphi$ ) iff  $(S, s') \models \varphi$  for every  $s'$  (resp. some  $s'$ ) s.t.  $s' \sim_a s$ .
- $(S, s) \models \mathbb{D}_A\varphi$  (resp.  $\widetilde{\mathbb{D}}_A\varphi$ ) iff  $(S, s') \models \varphi$  for every  $s'$  (resp. some  $s'$ ) s.t.  $s' \sim_A s$ .
- $(S, s) \models \mathbb{C}_A\varphi$  (resp.  $\widetilde{\mathbb{C}}_A\varphi$ ) iff  $(S, s') \models \varphi$  for every  $s'$  (resp. some  $s'$ ) s.t.  $s' \sim_A^C s$ .

and for all  $\pi \in cmp(S)$ :

- $(S, \pi) \models \varphi$ , for  $\varphi$  a state formula, iff  $(S, \pi_0) \models \varphi$ .
- $(S, \pi) \models \psi_1 \wedge \psi_2$  iff  $(S, \pi) \models \psi_1$  and  $(S, \pi) \models \psi_2$ .
- $(S, \pi) \models \psi_1 \vee \psi_2$  iff  $(S, \pi) \models \psi_1$  or  $(S, \pi) \models \psi_2$ .
- $(S, \pi) \models X\psi$  iff  $(S, \pi_{\geq 1}) \models \psi$ .
- $(S, \pi) \models \psi_1 U \psi_2$  iff there  $i \in \mathbb{N}$  such that  $(S, \pi_{\geq i}) \models \psi_2$  and for all  $j < i$ ,  $(S, \pi_{\geq j}) \models \psi_1$ .
- $(S, \pi) \models \psi_1 R \psi_2$  iff for all  $i$ , either  $(S, \pi_{\geq i}) \models \psi_2$  or there exists  $j < i$  such that  $(S, \pi_{\geq j}) \models \psi_1$ .

We emphasise two points: strategies have perfect recall, and epistemic operators depend only on the current state and not on the history (the latter appears, e.g., in (Čermák et al. 2014)).

**Semantics of Prompt-KATL<sup>\*</sup>.** For  $k \in \mathbb{N}$ , we use the notation  $(S, s) \models^k \varphi$  and  $(S, \pi) \models^k \psi$  to denote that the formula is satisfied in which every prompt eventuality is fulfilled within at most  $k$  steps. Formally: define  $(S, s) \models^k \varphi$  and  $(S, \pi) \models^k \psi$  inductively, as above,<sup>5</sup> with the following additional rule:

- $(S, \pi) \models^k F_P \psi$  iff there exists  $j \leq k$  such that  $(S, \pi_{\geq j}) \models^k \psi$ . Say that  $\pi$  models  $\psi$  with bound  $k$ .

**Definition 1.** For a Prompt-KATL<sup>\*</sup> state-formula  $\varphi$  and  $s \in S$ , define  $(S, s) \models \varphi$  iff there exists  $k \in \mathbb{N}$  such that  $(S, s) \models^k \varphi$ . Say that  $\varphi$  is satisfied at  $s$ .

## Linearising branching-formulas and Prompt-LTL

Like CTL<sup>\*</sup>, and ATL<sup>\*</sup> after it, one can think of a Prompt-KATL<sup>\*</sup> path formula  $\psi$  over atoms AP as a Prompt-LTL formula  $lin(\psi)$  over atoms which are the maximal state subformulas of  $\psi$ , as follows (Kupferman, Vardi, and Wolper 2000).

<sup>5</sup>Thus, e.g.,  $(S, \pi) \models^k \psi_1 U \psi_2$  iff there  $i \in \mathbb{N}$  such that  $(S, \pi_{\geq i}) \models^k \psi_2$  and for all  $j < i$ ,  $(S, \pi_{\geq j}) \models^k \psi_1$ .

A formula  $\varphi$  is a *state subformula* of  $\psi$  if  $\varphi$  is a state formula as well as a subformula of  $\psi$ . A formula  $\varphi$  is a *maximal state subformula* of  $\psi$  if  $\varphi \neq \psi$ , it is a state subformula of  $\psi$ , and it is not a proper subformula of any other state subformula of  $\psi$ . Let  $max(\psi)$  be the set of maximal state subformulas of  $\psi$ .

Every Prompt-KATL<sup>\*</sup> path formula  $\psi$  can be viewed as a Prompt-LTL formula, call it  $lin(\psi)$ , whose atoms are elements of  $max(\psi)$ . Formally:

**Definition 2.** For a path formula  $\psi$ , define  $lin(\psi)$  as follows — in each case note that  $lin(\psi)$  is a formula over atoms  $max(\psi)$ :

- $lin(p) := p$  and  $lin(\neg p) := \neg p$ ;
- For  $\circ \in \{\langle\langle A\rangle\rangle, [[A]], \mathbb{K}_a, \mathbb{D}_A, \mathbb{C}_A\}$ ,  $lin(\circ\psi) := \circ lin(\psi)$ ;
- If  $\psi = \phi_1 \circ \phi_2$  for  $\circ \in \{\vee, \wedge\}$ , then  $lin(\phi_1 \circ \phi_2)$  is defined to be  $lin(\phi_1) \circ lin(\phi_2)$  (note this is well defined since at least one  $\phi_i$  is not a state formula);
- For  $\circ \in \{X, F_P\}$ ,  $lin(\circ\phi) := \circ lin(\phi)$ ;
- For  $\circ \in \{U, R\}$ ,  $lin(\psi_1 \circ \psi_2) := lin(\psi_1) \circ lin(\psi_2)$ .

For example, if  $\psi = (p U \langle\langle A\rangle\rangle F_P q) \vee X \neg p$ , then its state subformulas are  $\{p, \langle\langle A\rangle\rangle F_P q, q, \neg p\}$ , and  $max(\psi) = \{p, \langle\langle A\rangle\rangle F_P q, \neg p\}$ , and thus  $lin(\psi)$  is the Prompt-LTL formula  $(\boxed{p} U \langle\langle A\rangle\rangle F_P q) \vee X \boxed{\neg p}$  over the atoms  $max(\psi)$  (for illustration we box the subformulas that are treated as atoms).

For an iCGS  $S := \langle Ag, AP, Act, S, \lambda, \delta, \{\sim_a : a \in Ag\}\rangle$ , and a Prompt-KATL<sup>\*</sup> path formula  $\psi$  over AP, define the iCGS  $S_\psi := \langle Ag, max(\psi), Act, S, \lambda, \delta_\psi, \{\sim_a : a \in Ag\}\rangle$  with atoms  $max(\psi)$  and a labeling  $\lambda_\psi : S \rightarrow 2^{max(\psi)}$  defined by letting  $\varphi \in \lambda_\psi(s)$  iff  $(S, s) \models \varphi$ . The next lemma says that a computation  $\pi$  of  $S$  satisfies  $\psi$  iff, when viewed as a computation of  $S_\psi$ , it satisfies  $lin(\psi)$ :

**Lemma 1.** For every iCGS  $S$ , Prompt-KATL<sup>\*</sup> path formula  $\psi$  over atoms AP, and computation  $\pi$  of  $S$ , we have that  $(S, \pi) \models \psi$  if and only if  $(S_\psi, \pi) \models lin(\psi)$ .

*Proof.* The proof is by induction on the structure of  $\psi$ , using the following inductive hypothesis on the subformulas  $\phi$  of  $\psi$  (that are not proper subformulas of any formula in  $max(\psi)$ ): for every position  $i \in \mathbb{N}$  of  $\pi$ , we have that  $(S, \pi_{\geq i}) \models \phi$  iff  $(S_\psi, \pi_{\geq i}) \models lin(\phi)$ .  $\square$

## Prompt linear-temporal logic.

We now consider Prompt-LTL in more detail.

*Notation.* For a path  $\pi \in cmp(S)$  and a Prompt-LTL formula  $\psi$ , write  $\pi \models \psi$  instead of  $(S, \pi) \models \psi$ . Also, if  $v \in (2^{AP})^\omega$  then we abuse notation and write  $v \models \psi$ .

Recall from the definitions of syntax that we define Prompt-LTL as the syntactic fragment of Prompt-KATL<sup>\*</sup> in which  $\langle\langle A\rangle\rangle$  and  $[[A]]$  do not occur. Thus, if  $\psi$  is a Prompt-LTL formula and  $S$  is a CGS, then  $(S, s) \models A\psi$  expresses that there is a bound  $k \in \mathbb{N}$  such that for every  $\pi \in cmp(S)$  starting in  $s$  we have that  $(S, \pi) \models^k \psi$ .<sup>6</sup>

<sup>6</sup>Remark: the syntax in (Kupferman, Piterman, and Vardi 2009) is different. There they write, e.g.,  $S \models \psi$  instead of  $S \models A\psi$ .

**Proposition 1.** (Kupferman, Piterman, and Vardi 2009) The following problem is decidable: given Prompt-LTL formula  $\psi$ , a finite CGS  $S$ , and a state  $s \in S$ , decide whether  $(S, s) \models \mathbf{A} \psi$ . Moreover, the complexity is PSPACE in the size of  $\psi$  and NLOGSPACE in the size of  $S$ .

The previous proposition allows us to deal with universal quantifiers. We now deal with the existential quantifier.

**Definition 3.** If  $\psi$  is a Prompt-LTL formula, define  $\text{live}(\psi)$  as the LTL formula that results from  $\psi$  by replacing every  $F_P$  by  $F$ .

The following lemma says that  $(S, s) \models E \psi$  if and only if  $(S, s) \models E \text{live}(\psi)$ . The reason, roughly, is as follows. For the forward direction, note that if  $\psi$  holds promptly on a computation, then in particular, it holds eventually (this is because our formulas are in positive-normal form and so the prompt eventualities can not be negated). For the reverse direction, use the fact that if  $S$  has a computation satisfying  $\text{live}(\psi)$  then it has such a computation that is a lasso, i.e., of the form  $uv^\omega$  (this holds for all LTL formulas, and thus  $\text{live}(\psi)$  in particular, (Vardi and Wolper 1994)). In this case, every subformula of  $\psi$  that holds in a suffix  $\pi_{\geq j}$  of  $\pi$  (with  $j \geq |u| + |v|$ ) also holds in the suffix  $\pi_{\geq j-|v|}$ . Thus all eventualities hold promptly (i.e., within  $|u| + |v|$  steps). The formal proof is in the full version of the paper.

**Lemma 2.** If  $\pi = uv^\omega$  is a lasso, and  $\psi$  is a Prompt-LTL formula, then the following are equivalent:

- a)  $\pi \models \psi$ ,
- b)  $\pi \models \text{live}(\psi)$ ,
- c)  $\pi \models^b \psi$  where  $b = |u| + |v|$ .

*Proof.* Clearly c)  $\rightarrow$  a). For a)  $\rightarrow$  b) an easy induction on  $\psi$  shows that for every computation  $\pi$ , and every  $k \in \mathbb{N}$ , if  $\pi \models^k \psi$  then  $\pi \models \text{live}(\psi)$ . For b)  $\rightarrow$  c) we prove, by induction on  $\psi$ , that for all  $i \in \mathbb{N}$ , and all subformulas  $\psi'$  of  $\psi$ : if  $\pi_{\geq i} \models \text{live}(\psi')$  then  $\pi_{\geq i} \models^b \psi'$  (to complete the proof take  $i = 0$  and  $\psi' = \psi$ ). The only non-trivial case is  $\psi' = F_P \psi_1$ . Thus, suppose  $\pi_{\geq i} \models \text{live}(F_P \psi_1)$ . Then  $\pi_{\geq i} \models F \text{live}(\psi_1)$ , and so there exists  $j \geq i$  such that  $\pi_{\geq j} \models \text{live}(\psi_1)$ . There are two cases.

Suppose  $j < |u|$ . By induction,  $\pi_{\geq j} \models^b \psi_1$ , and so  $\pi_{\geq i} \models^{\max\{j-i, b\}} F_P \psi_1$ . Since  $j - i < |u| - i < b$ , we have that  $\pi_{\geq i} \models^b F_P \psi_1$ , as required.

Suppose  $j \geq |u|$ . Pick  $n_0$  so that  $\max\{i, |u|\} \leq j - |v|n_0 \leq \max\{i, |u|\} + |v|$ . Since  $\pi = uv^\omega$ , we have that  $\pi_{\geq j-|v|n_0}$  is equal to  $\pi_j$ . Thus  $\pi_{\geq j-|v|n_0} \models \text{live}(\psi_1)$ , and so by induction  $\pi_{\geq j-|v|n_0} \models^b \psi_1$ , and so  $\pi_{\geq i} \models^{\max\{j-|v|n_0-i, b\}} F_P \psi_1$ . Since  $j - |v|n_0 - i \leq \max\{i, |u|\} + |v| - i \leq |u| + |v| = b$ , we have  $\pi_{\geq i} \models^b F_P \psi_1$ , as required.  $\square$

We now get:

**Proposition 2.** For every Prompt-LTL formula  $\psi$ , CGS  $S$ , and state  $s \in S$ , we have that  $(S, s) \models E \psi$  if and only if  $(S, s) \models E \text{live}(\psi)$ . Moreover, the cost of checking this fact is PSPACE in the size of  $\psi$  and NLOGSPACE in the size of  $S$ .

*Proof.* Since  $\text{live}(\psi)$  is an LTL formula, we have that  $(S, s) \models E \text{live}(\psi)$  if and only if there is a lasso  $\pi = uv^\omega$  starting in  $s$  such that  $(S, \pi) \models \text{live}(\psi)$  (Vardi and Wolper 1994). By Lemma 2 this is equivalent to  $(S, \pi) \models^b \psi$  where  $b = |u| + |v|$ . By definition of  $\models^b$ , this is equivalent to  $(S, s) \models E \psi$ . The complexity follows from the model-checking complexity of LTL (Sistla and Clarke 1985; Kupferman, Piterman, and Vardi 2009).  $\square$

The final lemma of this section will be used in the proof of Proposition 4 (used as part of Theorem 4 on co-operative strategies). The lemma relies on the alternating-color technique from (Kupferman, Piterman, and Vardi 2009), that we now describe. Given a computation  $\pi$  of an iCGS  $S$ , add a new atomic proposition  $\text{red}$ , and imagine that states in which  $\text{red}$  holds are colored red, and otherwise white. Given a Prompt-LTL formula  $\psi$ , derive from it an LTL formula  $\text{col}(\psi)$  by replacing every subformula of the form  $F_P \phi$  with a subformula that says that  $\phi$  holds before the color changes twice. Now, if we can come up with a coloring of  $\pi$  in which the colors alternate fast enough (say every  $k$  steps or less), such that the colored version of  $\pi$  models  $\text{col}(\psi)$ , then we can deduce that  $\pi$  models  $\psi$  with bound  $2k$ ; conversely, if  $\pi$  models  $\psi$  with bound  $k$ , then by changing colors every  $k$  steps we can ensure that  $\text{col}(\psi)$  is satisfied. This allows us to replace reasoning about Prompt-LTL formulas with reasoning about colorings and LTL formulas. We now formally describe the required elements for applying this reasoning.

For  $w \in (2^{\text{AP} \cup \{\text{red}\}})^\omega$ , a *block* is a maximal subword  $w_i \dots w_j$  of  $w$  such that  $\text{red} \in w_l$  for all  $l \in [i, j]$  or  $\text{red} \notin w_l$  for all  $l \in [i, j]$ . For  $k \in \mathbb{N}$ , say that  $w$  is *k-coloured* if the size of every block of  $w$  has length at most  $k$ . Say that  $w \in (2^{\text{AP} \cup \{\text{red}\}})^\omega$  is a *colouring* of  $v \in (2^{\text{AP}})^\omega$  if and only if for every  $i$ ,  $w_i \cap \text{AP} = v_i$ . Say that  $w$  is a *k-colouring* of  $v$  if  $w$  is a colouring of  $v$  and is *k-coloured*.

For a Prompt-LTL formula  $\psi$ , let  $\psi'$  be the LTL formula obtained by replacing every subformula of  $\psi$ , of the form  $F_P \phi$ , by  $\text{within}(\phi) := (\text{red} U (\neg \text{red} U \phi)) \vee (\neg \text{red} U (\text{red} U \phi))$ , which states that  $\phi$  should hold at some point within this color block or the next. Let  $\text{col}(\psi) := \psi' \wedge \rho$ , where  $\rho := GF(\text{red} \wedge X \neg \text{red})$  states that the colors change infinitely often. It is important to note that while  $\text{col}(\psi)$  is exponentially larger than  $\psi$ , the number of subformulas it has is linear in the number of subformulas of  $\psi$ .

**Lemma 3.** For every Prompt-LTL formula  $\psi$ , word  $v \in (2^{\text{AP}})^\omega$  and  $k \in \mathbb{N}$ : (i) if there is a *k-colouring*  $w$  of  $v$  such that  $w \models \text{col}(\psi)$  then  $v \models^{2k} \psi$ ; (ii) if  $v \models^k \psi$  then there is a *k-colouring*  $w$  of  $v$  such that  $w \models \text{col}(\psi)$ , moreover,  $w$  can be chosen with all blocks of size exactly  $k$ .

*Proof.* Although this lemma is easily extractable from (Kupferman, Piterman, and Vardi 2009), for completeness we provide the proof. For (i), given a *k-colouring*  $w$  of  $v$ , if  $w \models \text{col}(\psi)$  then the result (that  $w \models^{2k} \psi$ ) follows by induction on  $\psi$  and the following observation: for every  $i \in \mathbb{N}$  and every subformula  $F_P \phi$  of  $\psi$ , we have  $w_{\geq i} \models \text{within}(\phi)$  implies that  $v_{\geq i} \models^{2k} F_P \phi$ .

For (ii), if  $v \models^k \psi$  then colour  $v$  by blocks of size exactly  $k$  to get  $w$ . The result (that  $w \models \text{col}(\psi)$ ) follows by induc-

tion on  $\psi$  and the following observation: for every  $i \in \mathbb{N}$  and every subformula  $F_P \phi$  of  $\psi$ , we have  $v_{\geq i} \models^k F_P \phi$  implies that  $w_{\geq i} \models \text{within}(\phi)$ .  $\square$

## Deciding the model-checking problems

The model-checking problem for a logic  $\mathcal{L}$  is the following: given a formula  $\varphi$  from  $\mathcal{L}$  and a finite iCGS  $S$ , decide whether  $S \models \varphi$ .

**Fact 1.** *Model checking ATL\* is undecidable over concurrent game structures with  $|Ag| \geq 3$ , and imperfect information, already for the existential fragment (Pnueli and Rosner 1989; Dima and Tiplea 2011).*

Thus, also model checking Prompt-KATL\* is undecidable for three or more agents with imperfect information. We show that one can regain decidability (and we provide the optimal complexity) in four ways: (i) restricting to iCGS with perfect information, or (ii) restricting to path quantifiers (instead of strategy quantifiers), or (iii) restricting to memoryless strategies, or (iv) restricting to cooperative strategies and the existential fragment.

### Prompt-ATL\* with perfect information

**Proposition 3.** *The following problems are decidable: given a Prompt-LTL formula  $\psi$ , a finite (perfect-information) CGS  $S$ , a set of agents  $A \subseteq Ag$ , and state  $s \in S$ , decide whether  $(S, s) \models \langle\langle A \rangle\rangle \psi$ ; and, similarly, decide whether  $(S, s) \models [[A]]\psi$ . Moreover, the complexity of these problems is 2EXPTIME in the number of subformulas of  $\psi$ , and polynomial in the size of  $S$ .*

*Proof.* We will reduce each question to the problem of deciding if a given player has a winning strategy in Prompt-LTL turn-based games. The latter are solvable in 2EXPTIME (Zimmermann 2013).

We first consider the case of  $\langle\langle A \rangle\rangle \psi$ . In the first step, build a two-player arena  $G$  such that  $(\star)$ :  $(S, s) \models \langle\langle A \rangle\rangle \psi$  if and only if there exists  $k \in \mathbb{N}$  such that Player 0 can enforce (in  $G$ ) the LTL goal  $\psi_k$  from  $s$ , where  $\psi_k$  is formed from  $\psi$  by replacing every subformula of the form  $F_P \phi$  by  $\bigvee_{i \leq k} X^i \phi$ . The idea is that Player 0 corresponds to the coalition  $A$  and Player 1 to the coalition  $Ag \setminus A$ . In more detail:  $S = \langle Ag, AP, Act, S, \lambda, \delta \rangle$ , define  $\tilde{G}$  to be the (perfect information) CGS with two agents, Player 0 and Player 1,  $\langle \{0, 1\}, AP, Act_0 \cup Act_1, S, \lambda, \delta_G \rangle$  as follows:

- action sets  $Act_0 := Act^A$  and  $Act_1 := Act^{Ag \setminus A}$ ,
- state set  $S$ , labeling  $\lambda$ ,
- transition function  $\delta_G : S \times Act_0 \times Act_1 \rightarrow S$  that maps  $(s, (d_1, d_2)) \mapsto \delta(s, d_1 \otimes d_2)$  where  $d_1 \otimes d_2 \in Act_0^{Ag}$  maps  $a$  to  $d_1(a)$  for  $a \in A$  and otherwise (for  $a \in Ag \setminus A$ ) to  $d_2(a)$ .

It is immediate from the definitions (of  $G$  and  $\models$ ) that  $(\star)$  holds. For the next step, recall the following folk fact  $(\dagger)$ : Player 0 has a winning strategy in a concurrent game  $G$  with goal  $\Upsilon$  if and only if Player 0 has a winning strategy in the turn-based game  $G^{tb}$ , which simulates  $G$  as follows: first, Player 0 moves (thus revealing his chosen action) and then

Player 1 chooses his action and the simulated move is completed. To “skip” the intermediate nodes that  $G^{tb}$  introduces, we use the goal  $\Upsilon^{tb}$  which is defined to be the set of all computations such that the subsequence consisting of only the even positioned nodes is in  $\Upsilon$ . The intuitive reason that this lemma is true is that in the game  $G^{tb}$  Player 0 has no new information available to it, and thus it is exactly as hard (or easy) for him to win as in  $G$ .<sup>7</sup>

Formally, we build the turn-based game  $G^{tb}$  of perfect information with goal  $\Upsilon_k^{tb}$  as follows. Let  $G^{tb}$  be the 2-player game, over atoms  $AP$ , and such that:

- the state set  $S^{tb}$  is  $S \cup (S \times Act_0)$ ,
- the labeling function  $\lambda^{tb}$  maps  $s \mapsto \lambda(s)$  and  $(s, d) \mapsto \emptyset$ ,
- the transition function  $\delta^{tb} : S^{tb} \times Act_0 \times Act_1 \rightarrow S^{tb}$  maps state  $s \in S$  and actions  $(d_0, d_1)$  to  $(s, d_0)$ , and maps a state  $(s, d_0) \in S \times Act_0$  and actions  $(d'_0, d_1)$  to  $\delta(s, d_0 \otimes d_1)$  (for all  $s \in S$ ,  $d_0, d'_0 \in Act_0$  and  $d_1 \in Act_1$ ).

Moreover,  $\Upsilon_k^{tb}$  consists of those plays whose subsequence consisting of every other node satisfies  $\psi_k$ .

Thus, we have reduced our problem to deciding if there exists  $k \in \mathbb{N}$  such that Player 0 has a winning strategy in the game  $G^{tb}$  with goal  $\Upsilon_k^{tb}$ . The latter is the problem of solving a two-player turn-based Prompt-LTL game, which, by (Zimmermann 2013), can be done in 2-EXPTIME. Moreover, that procedure is already able to deal with goals which only look at every second node of the play (so called “blinking semantics”). Thus, the procedure can be easily adapted to decide if there exists  $k \in \mathbb{N}$  such that Player 0 has a winning strategy in the game  $G^{tb}$  with goal  $\Upsilon_k^{tb}$ , which completes the  $\langle\langle A \rangle\rangle \psi$  case.

We turn to the  $[[A]]\psi$  case. Dually to the case above, build a two-player game  $H$  by associating the coalition  $A$  with Player 1 and associating the coalition  $Ag \setminus A$  with Player 0. Thus,  $(S, s) \models [[A]]\psi$  if and only if there exists  $k \in \mathbb{N}$  such that Player 1 does not have a winning strategy in the game  $H$  with goal  $\psi_k$ . Dually again, build  $H^{tb}$  in which Player 1 moves first and use the fact  $(\dagger)$  to deduce that, for every  $k \in \mathbb{N}$ , Player 1 does not have a winning strategy in  $H$  with goal  $\psi_k$  if and only if Player 1 does not have a winning strategy in the game  $H^{tb}$  with goal  $\Upsilon_k^{tb}$ . Since turn-based games of perfect information with LTL goals (the “blinking semantics” is easily accommodated) are determined (i.e., one of the players has a winning strategy), then this is equivalent to Player 0 having a winning strategy in the game  $G^{tb}$  with goal  $\Upsilon_k^{tb}$ . But this is the same type of game we had already solved in the case of  $\langle\langle A \rangle\rangle \psi$ . This completes the  $[[A]]\psi$  case.  $\square$

**Theorem 1.** *The model checking problem for Prompt-ATL\* (resp. Prompt-ATL) is 2EXPTIME-complete (resp. PTIME-complete).*

*Proof.* By (Alur, Henzinger, and Kupferman 2002), the lower bound already holds for ATL\* (resp. ATL). For the upper bound, we adapt the marking algorithm for model checking ATL\* (Alur, Henzinger, and Kupferman 2002).

<sup>7</sup>However, Player 1 can win in  $G^{tb}$  from states in which he can only prevent Player 0 from winning in  $G$ , but not ensure he himself wins.

Let  $\varphi$  be a state **Prompt-ATL**<sup>\*</sup> formula, and mark every state  $s \in S$  by the state subformulas  $\varphi'$  of  $\varphi$  that are satisfied at  $s$ . This is done inductively. The case that  $\varphi'$  is an atom, a negation of an atom, a disjunction or a conjunction is immediate (e.g., if  $\varphi' = \varphi_1 \vee \varphi_2$  and  $s$  is marked by  $\varphi_1$  then also mark  $s$  by  $\varphi_1 \vee \varphi_2$ ).

The case that  $\varphi' = \langle\!\langle A \rangle\!\rangle \psi$  is dealt with as follows. Recall that  $lin(\psi)$  is a **Prompt-LTL** formula over atoms  $max(\psi)$ . By Lemma 1, deciding if  $(S, s) \models \langle\!\langle A \rangle\!\rangle \psi$  is equivalent to deciding if  $(S_\psi, s) \models \langle\!\langle A \rangle\!\rangle lin(\psi)$ . By induction, the satisfaction of all state subformulas of  $\psi$  have already been determined. In particular, we have enough information to form  $S_\psi$ . By Proposition 3, deciding whether  $(S_\psi, s) \models \langle\!\langle A \rangle\!\rangle lin(\psi)$  can be done in 2EXPTIME. The case that  $\varphi' = [[A]]\psi$  is similar.

The case of **Prompt-ATL** follows by noting that the formula  $lin(\psi)$  is always of constant size since **Prompt-ATL** does not allow temporal operators to be directly nested.  $\square$

### Prompt-KCTL<sup>\*</sup>

In the case of **Prompt-KCTL**<sup>\*</sup>, the strategy quantifiers are restricted to E, A. This inherent restriction on the strategy quantifiers results in a decidable model checking problem, also in the presence of imperfect information.

**Theorem 2.** *Model checking **Prompt-KCTL**<sup>\*</sup> is PSPACE-complete, and the structure complexity is PTIME-complete.*

*Proof.* The lower-bounds already hold for CTL<sup>\*</sup> (Kupferman, Vardi, and Wolper 2000). For the upper-bounds, let  $\varphi$  be a state **Prompt-KCTL**<sup>\*</sup> formula, and mark every state  $s \in S$  by the state subformulas  $\varphi'$  of  $\varphi$  that are satisfied at  $s$ . This is done inductively as in Theorem 1. The cases we have to consider are the epistemic operators and the complexity of the path quantifiers. The case that  $\varphi'$  is of the form  $\mathbb{K}_a \phi$  is dealt with as follows: mark  $s$  by  $\mathbb{K}_a \phi$  iff every  $s' \sim_a s$  is already marked by  $\phi$ ; similarly, mark  $s$  by  $\mathbb{D}_A \phi$  iff every  $s' \sim_A s$  is already marked by  $\phi$  (the remaining epistemic operators, including the duals, are similar). Each of these steps can be performed in time polynomial in  $S$ .

We now discuss the case that  $\varphi'$  is of the form  $E \psi$  or  $A \psi$ . Recall that  $lin(\psi)$  is a **Prompt-LTL** formula over atoms  $max(\psi)$ , and that  $live(\cdot)$  replaces every  $F_P$  by F in a **Prompt-LTL** formula. Thus,  $live(lin(\psi))$  is an LTL formula over atoms  $max(\psi)$ . By induction, the satisfaction of all state subformulas of  $\psi$  has already been determined. In particular, we have enough information to form the CGS  $S_\psi$  (note that since  $S$  is a CGS, so is  $S_\psi$ ). By Lemma 1 note that ( $\dagger$ ): for  $Q \in \{E, A\}$ ,  $(S, s) \models Q\psi$  if and only if  $(S_\psi, s) \models Qlin(\psi)$ .

For  $\varphi' = A \psi$  mark  $s$  by  $A \psi$  if and only if  $(S_\psi, s) \models A lin(\psi)$ . To see this is correct use ( $\dagger$ ).

For  $\varphi' = E \psi$ , mark  $s$  by  $E \psi$  if and only if  $(S_\psi, s) \models E live(lin(\psi))$ . To see that this is correct use ( $\dagger$ ), Proposition 2, and the fact that  $lin(\psi)$  is a **Prompt-LTL** formula over atoms  $max(\psi)$ .

For the complexity, note that i) there are only a linear number of subformulas  $\varphi'$  of  $\varphi$ , and ii) each case  $\varphi'$  of the algorithm can be done in PSPACE in the size of  $\varphi'$  and NLOGSPACE in the size of  $S$  (for the  $\varphi' = A \psi$  case

use Proposition 1, and for the  $\varphi' = E \psi$  case use the fact that model checking LTL is in PSPACE (Sistla and Clarke 1985)).  $\square$

### Prompt-KATL<sup>\*</sup> with memoryless strategies

In this section we prove that model checking **Prompt-KATL**<sup>\*</sup> with memoryless strategies is PSPACE-complete. We begin with the relevant definitions.

A strategy  $\sigma$  is called *memoryless* if for all histories  $h, h'$ , and every state  $s$  we have  $\sigma(hs) = \sigma(h's)$ . It is thus common to consider memoryless strategies as functions from states (not histories) to actions.

Define  $\models_{mem}^k$  like  $\models^k$  except replace the definition of the semantics of the strategy quantifiers to limit the agents to memoryless (observational) strategies as follows:<sup>8</sup>

- $(S, s) \models_{mem}^k \langle\!\langle A \rangle\!\rangle \psi$  (resp.  $[[A]]\psi$ ) iff there exists a set (resp. for all sets)  $\Sigma_A$  of memoryless observational strategies, one strategy for each agent in  $A$ , such that for all (resp. for at least one) computation  $\pi \in out(s, \Sigma_A)$ , we have  $(S, \pi) \models^k \psi$ .

Define  $(S, s) \models_{mem} \varphi$  iff there exists  $k \in \mathbb{N}$  such that  $(S, s) \models_{mem}^k \varphi$ ,

**Theorem 3.** *The following problem is PSPACE-complete: given a **Prompt-KATL**<sup>\*</sup> formula  $\varphi$  and a finite iCGS  $S$ , decide whether  $S \models_{mem} \varphi$ .*

*Proof.* The lower-bound already holds for CTL<sup>\*</sup> (Kupferman, Vardi, and Wolper 2000). For the upper bound, we use the marking algorithm as we did in Theorem 1. We show how to deal with the existential strategy quantifier (the universal quantifier is symmetric). Apply Lemma 1 and reduce  $(S, s) \models_{mem} \langle\!\langle A \rangle\!\rangle \psi$  to  $(S_\psi, s) \models_{mem} \langle\!\langle A \rangle\!\rangle lin(\psi)$ . To solve the latter, observe the following: i) each agent has finitely many observational memoryless strategies (each of polynomial size) in  $S$ , ii) instantiating a set  $\Sigma_A$  of such strategies, one for each agent in  $A$ , results in a sub-area  $S'$  of  $S_\psi$  in which we have to check whether  $(S', s) \models A lin(\psi)$ . By Proposition 1, each step ii) can be done in PSPACE. Thus one can, in PSPACE, search for a set of observational memoryless strategies  $\Sigma_A$  such that ii) holds.  $\square$

### Existential Fragment of **Prompt-KATL**<sup>\*</sup> with co-operative strategies

In this section we prove that model checking the existential fragment of **Prompt-KATL**<sup>\*</sup> with co-operative strategies is 2EXPTIME-complete. We begin with some definitions.

Define  $\models_{co}^k$  like  $\models^k$  except replace the definition of the semantics of the strategy quantifiers  $\langle\!\langle A \rangle\!\rangle, [[A]]$  as follows:

- $(S, s) \models_{co}^k \langle\!\langle A \rangle\!\rangle \psi$  (resp.  $[[A]]\psi$ ) iff there exists a set (resp. for all sets)  $\Sigma_A$  of strategies, one strategy for each agent in  $A$ , that is co-operatively observational, such that for all computations (resp. for at least one computation)  $\pi \in out(s, \Sigma_A)$  we have  $(S, \pi) \models^k \psi$ .

<sup>8</sup>One can also define a “uniform” semantics in which also the agents not quantified over (i.e. the ones not in  $A$ ) are limited to memoryless strategies. Our techniques can be easily adapted also to this uniform semantics.

Define  $(S, s) \models_{co} \varphi$  iff there exists  $k \in \mathbb{N}$  such that  $(S, s) \models_{co}^k \varphi$ .

**Theorem 4.** *The model-checking problem for the existential fragment of Prompt-KATL\* (resp. Prompt-KATL) with cooperative strategies is 2EXPTIME-complete (resp. in PTIME).*

The lower bound holds already for the existential fragment of ATL\* (see the proof of Theorem 5.6 in (Alur, Henzinger, and Kupferman 2002)). For the upper bound, the proof proceeds as in previous constructions, using the marking algorithm from the proof of Theorem 1. The interesting case is the strategy quantifier  $\langle\langle A \rangle\rangle$ , which is dealt with by the following proposition.

**Proposition 4.** *The following problem is decidable: given a Prompt-LTL formula  $\psi$ , a finite iCGS  $S$ , a set of agents  $A \subseteq Ag$ , and state  $s \in S$ , decide whether  $(S, s) \models_{co} \langle\langle A \rangle\rangle \psi$ . Moreover, this can be done in 2EXPTIME in the number of subformulas of  $\psi$ , and polynomial time in the size of  $S$ .*

*Proof.* The outline of the proof is as follows: we build a two-player arena  $G$  such that  $(\star)$ :  $(S, s) \models_{co} \langle\langle A \rangle\rangle \psi$  if and only if there exists  $k \in \mathbb{N}$  such that Player 0 has a strategy in  $G$  that enforces the LTL goal  $\psi_k$  from  $s$  (as in Proposition 3,  $\psi_k$  is formed from  $\psi$  by replacing every subformula of the form  $F_P \phi$  by  $\bigvee_{i \leq k} X^i \phi$ ). The idea is that Player 0 corresponds to the coalition  $A$  and Player 1 to the coalition  $Ag \setminus A$ . We are justified in treating all players in  $A$  as a single player by our assumption of cooperative strategies for them (the antagonists in  $Ag \setminus A$  are always treated as a single player since all of their strategies have to be defeated).

We then modify  $G$  to obtain a new arena  $G'$ , which allows Player 0 to choose colours (*red* or *not red*) at every second step. We then prove  $(\star\star)$  for every  $k \in \mathbb{N}$ , if Player 0 has an observational strategy in  $G$  that enforces goal  $\psi_k$  from  $s$  then it has an observational strategy in  $G'$  that enforces, from  $s$ , the following goal  $(\dagger)$ :  $col(\psi)$  holds and no colour consecutively repeats more than  $k$  times<sup>9</sup>, and vice versa (but with  $\psi_{2k}$  substituted for  $\psi_k$ , i.e., that if Player 0 has an observational strategy in  $G'$  that enforces  $(\dagger)$  then it has an observational strategy in  $G$  that enforces  $\psi_{2k}$ ). Finally, using the fact that LTL games of imperfect information admit finite-state strategies, we will show how to decide (in 2EXPTIME) whether there exists  $k \in \mathbb{N}$  such that Agent 0 has a strategy in  $G'$  that enforces  $(\dagger)$  from  $s$ . Here are some more details.

If  $S = \langle Ag, AP, Act, S, \lambda, \delta, \{\sim_a : a \in Ag\} \rangle$ , define  $G$  to be the iCGS  $\langle \{0, 1\}, AP, Act_0 \cup Act_1, S, \lambda, \delta_G, \{\sim_0, \sim_1\} \rangle$  as follows:

- action sets  $Act_0 := Act^A$  and  $Act_1 := Act^{Ag \setminus A}$ ,
- state set  $S$ , labeling  $\lambda$ ,
- transition function  $\delta_G : S \times Act_0 \times Act_1 \rightarrow S$  that maps  $(s, (d_1, d_2)) \mapsto \delta(s, d_1 \otimes d_2)$  where  $d_1 \otimes d_2 \in Act^{Ag}$  maps  $a$  to  $d_1(a)$  for  $a \in A$  and otherwise (for  $a \in Ag \setminus A$ ) to  $d_2(a)$ ,
- relation  $\sim_0, \sim_1$  defined as follows:  $s \sim_0 r$  if  $s \sim_A r$ , and  $s \sim_1 r$  if  $s \sim_{Ag \setminus A} r$ .

---

<sup>9</sup>W remind the reader that  $col(\psi)$  is defined before Lemma 3.

It is immediate from the definitions (of  $G$  and  $\models_{co}$ ) that  $(\star)$  holds. Define  $G'$  to be the iCGS obtained from  $G$  by adding a new atom *red* and splitting every transition from  $s$  to  $s'$  using decision  $d$ , into a diamond shape, where the first decision is either to go left from  $s$  (decision  $d$  with *red*) or go right (decision  $d$  with *not red*) to one of two intermediate nodes. The choice of colour is made entirely by Player 0. Then, from each of these intermediate nodes every decision leads to the node  $s'$ . The observation sets are defined such that the intermediate nodes that are successors of indistinguishable nodes are themselves indistinguishable (and thus no new information leaks). Formally,  $G'$  is the iCGS<sup>10</sup>  $\langle \{0, 1\}, AP \cup \{\text{red}\}, Act', S', \lambda', \delta', \{\sim'_0, \sim'_1\} \rangle$  where:

- $Act'_0 := Act_0 \times \{\text{red}, \text{not red}\}$  and  $Act'_1 := Act_1$ ,
- $S' := S \cup (S \times Act^{Ag} \times \{\text{red}, \text{not red}\})$ ,
- the labeling  $\lambda'$  maps  $s \mapsto \lambda(s)$  and  $(s, d, x) \mapsto \{x\}$ ,
- transition function  $\delta' : S' \times Act'_0 \times Act'_1 \rightarrow S'$  that, for actions  $(d_0, x) \in Act'_0$  and  $d_1 \in Act_1$ , maps state  $s \in S$  and actions  $((d_0, x), d_1)$  to  $(s, d_0 \otimes d_1, x)$ ; and that maps state  $(s, d, x)$  and all actions of the players to  $\delta_G(s, d)$ ;
- REMOVED TEXT THAT DOESNT COMPILE
- observation sets  $\sim'_0, \sim'_1$  defined as follows:  $s \sim'_i r$  if  $s \sim_i r$ , and  $(s, d, x) \sim'_i (r, d', x')$  if  $s \sim_i r$ .

We now describe, for every  $k \in \mathbb{N}$ , the natural transformation of strategies for Player 0 between  $G$  and  $G'$ . For  $h' \in \text{hist}(S')$  define  $\text{proj}(h') \in \text{hist}(S)$  to be the string in which every second symbol of  $h'$  is removed. Fix an action  $\alpha \in Act'_0$ . An observational strategy  $\sigma_0 : \text{hist}(S) \rightarrow Act_0$  in  $G$  induces the strategy  $\sigma'_0 : \text{hist}(S') \rightarrow Act'_0$  in  $G'$  defined as follows. If  $h'$  ends in an element of  $S$  then  $\sigma'_0(h') := (\sigma_0(\text{proj}(h')), x)$ , where  $x$  is *red* if the integer part of  $\frac{|\text{proj}(h')|}{k}$  is odd, and otherwise (if it is even),  $x$  is *not red* (in words, use  $\sigma_0$  and swap the colour every  $k$  steps in  $G$ ). If  $h'$  does not end in an element of  $S$  then define  $\sigma'_0(h') := \alpha$  (recall that all transitions from the intermediate nodes go to the same destination). It is easy to see that  $\sigma'_0$  is observational since  $\sigma_0$  is.

Before we prove the converse, we state a useful fact that follows from an induction on the length of histories: if  $\text{proj}(h'_1) \sim_0 \text{proj}(h'_2)$  then  $h'_1 \sim'_0 h'_2$ . Now, an observational strategy  $\sigma'_0 : \text{hist}(S') \rightarrow Act'_0$  in  $G'$  induces the strategy  $\sigma_0 : \text{hist}(S) \rightarrow Act_0$  in  $G$  defined as follows:  $\sigma_0(h) := \sigma'_0(h')$  where  $\text{proj}(h') = h$ . This is well defined because, if  $\text{proj}(h'_1) = \text{proj}(h'_2) = h$  then  $h'_1 \sim'_0 h'_2$  (by the useful fact), and so  $\sigma'_0(h'_1) = \sigma'_0(h'_2)$  (since  $\sigma'_0$  is observational). To see that  $\sigma_0$  is observational let  $h_1 \sim_0 h_2$  be equivalent histories in  $G$ , and suppose  $\text{proj}(h'_i) = h_i$ . Then by the useful fact, also  $h'_1 \sim'_0 h'_2$ , and thus, since  $\sigma'_0$  is observational, we have  $\sigma'_0(h'_1) = \sigma'_0(h'_2)$ .

We now establish  $(\star\star)$ . Fix  $k$ . A strategy  $\sigma_0$  in  $G$  that ensures  $\psi_k$  implies (by Lemma 3) that every play consistent with  $\sigma_0$  can be  $k$ -coloured by blocks of size exactly  $k$  in a way that satisfies  $col(\psi)$ . Thus, the transformed strategy  $\sigma'_0$  of  $G'$  ensures  $(\dagger)$ . Conversely, suppose strategy  $\sigma'_0$  in  $G'$

---

<sup>10</sup>We assume that the two agents use different sets of actions  $Act'_0$  and  $Act'_1$  — see discussion after the definition of iCGS.

ensures ( $\dagger$ ). Then (by Lemma 3) the transformed strategy  $\sigma_0$  of  $G$  ensures  $\psi_{2k}$ .

Finally, we show that Player 0 can enforce ( $\dagger$ ) from  $s$  in  $G'$  if and only if Player 0 can enforce  $\text{col}(\psi)$  from  $s$  in  $G'$ . The “only if” direction is immediate from the definitions. For the “if” direction, it is implicit in (Kupferman and Vardi 1997a; Kupferman and Vardi 1997b) that LTL games of imperfect information admit finite-state strategies. We claim that if a strategy uses memory  $k$  then it  $(|S| \times k)$ -colours the play. Indeed, let  $h$  be a history, and take  $i < j \leq |h|$  such that the infix between positions  $i$  and  $j$  is of length at least  $|S| \times k$ . Hence, there exists  $i \leq i' < j' \leq j$  such that the last states of  $x := h_{\leq i'}$  and  $y := h_{\leq j'}$  are equal, and the strategy is in the same memory-state after processing the histories  $x$  and  $y$ . In particular, the strategy gives the same action on  $x$  as on  $y$ . Thus, the opponent can repeatedly play the infix between  $i'$  and  $j'$ , and thus this infix must contain a colour change since we assumed that the strategy enforces  $\text{col}(\psi)$  (which states, in particular, that the colours alternate infinitely often). This completes the “if” direction.  $\square$

## Conclusion

Reasoning about promptness has recently received attention for linear specifications (Alur et al. 2001; Kupferman, Piterman, and Vardi 2009; Zimmermann 2013). This is due to the fact that questions like “a specific state is eventually reached in a computation” have a clear meaning and application in the theory of formal verification, but are useless in practical scenarios if there is no bound on the time before the specific state is reached.

In this work, we initiated the study of prompt requirements over branching time specifications for multi-agent systems. We introduced the logic **Prompt-KATL\***, an extension of the classic **ATL\***, in which we added the prompt eventuality temporal operator  $F_P$ , and settled the model-checking complexity of many natural fragments, under various restrictions, i.e., perfect information, memoryless strategies, and the existential fragment with co-operative strategies. In particular, we prove that model-checking **Prompt-KCTL\*** is PSPACE-complete without any restrictions. Note that in the case of two players the restriction to co-operative strategies is not a real restriction as these correspond to ordinary strategies. Our results for **Prompt-KATL\*** and **Prompt-KATL** are summarised in the following tables:

Perfect Info. or co-operative $\exists$ Fragment	Memoryless
2EXPTIME-complete	PSPACE-complete

Figure 1: Complexity of model checking **Prompt-KATL\***

Perfect Info. or co-operative $\exists$ Fragment	Memoryless
in PTIME	in PSPACE

Figure 2: Complexity of model checking **Prompt-KATL**

As our results show, the complexity of model checking the considered logics with the prompt operator is the same

as without the prompt operator <sup>11</sup>.

We remark that our upper-bounds for **Prompt-KATL** were obtained essentially as a side-effect of the results for **Prompt-KATL\***, i.e., by analysing the complexity of the algorithms we provided for **Prompt-KATL\*** in the restricted case of **Prompt-KATL**. However, one can directly reason on **Prompt-KATL**. For example, in the perfect-information case, one can show that (like for **Prompt-CTL**)  $S \models \varphi$ , for a **Prompt-ATL** formula  $\varphi$ , iff  $S \models \phi$ , where  $\phi$  is the ATL formula obtained by replacing every prompt-eventuality  $F_P$  in  $\varphi$  by a regular eventuality  $F$ . The underlying reason is that in **Prompt-KATL**  $F_P$  only ranges over state sub-formulas and thus, reasoning about sub-formulas of the form  $\langle\langle A \rangle\rangle F_P \psi$  and  $[[A]] F_P \psi$  reduces to reasoning about two player games with reachability goals. In such games a (memoryless<sup>12</sup>) winning strategy does not admit a lasso in which  $\psi$  does not hold on all its states (since then the opponent can pump the loop of the lasso and win). Thus, if the goal  $\psi$  is achieved, it is achieved within a number of steps that is at most the size of the model, and thus promptly. For the case of imperfect information, more complicated reasoning can be employed.

Our work opens up several avenues of research. First, an intriguing open question is whether model-checking **Prompt-KATL\*** under cooperative strategies is decidable, even for two players (in this work we established the optimal complexity for its existential fragment). Second, the satisfiability problem has yet to be tackled, and we believe that the techniques introduced in this paper would yield useful for this investigation.

**Acknowledgments.** Benjamin Aminof and Florian Zuleger are supported by the Austrian National Research Network S11403-N23 (RiSE) of the Austrian Science Fund (FWF) and by the Vienna Science and Technology Fund (WWTF) through grant ICT12-059. Sasha Rubin is a Marie Curie fellow of the Istituto Nazionale di Alta Matematica. Aniello Murano was supported in part by GNCS 2016 project: Logica, Automi e Giochi per Sistemi Auto-adattivi.

## References

- [Almagor, Hirshfeld, and Kupferman 2010] Almagor, S.; Hirshfeld, Y.; and Kupferman, O. 2010. Promptness in omega-regular automata. In *ATVA 2010*, LNCS 6252, 22–36. Springer.
- [Alur et al. 2001] Alur, R.; Etessami, K.; La Torre, S.; and Peled, D. 2001. Parametric temporal logic for model measuring. *ACM Transactions on Computational Logic* 2(3):388–407.
- [Alur, Henzinger, and Kupferman 2002] Alur, R.; Henzinger, T.; and Kupferman, O. 2002. Alternating-Time Temporal Logic. *Journal of the ACM* 49(5):672–713.
- [Aminof et al. 2013] Aminof, B.; Legay, A.; Murano, A.; Serre, O.; and Vardi, M. Y. 2013. Pushdown module checking with imperfect information. *Inf. Comput.* 223:1–17.

<sup>11</sup>except for the case of **Prompt-KATL** under observational memoryless strategies.

<sup>12</sup>Winning strategies in such games can be taken w.l.o.g. to be memoryless.

- [Aminof, Murano, and Vardi 2007] Aminof, B.; Murano, A.; and Vardi, M. Y. 2007. Pushdown module checking with imperfect information. In *CONCUR 2007*, 460–475.
- [Bulling and Jamroga 2014] Bulling, N., and Jamroga, W. 2014. Comparing variants of strategic ability: how uncertainty and memory influence general properties of games. *Journal of Autonomous Agents and Multi-Agent Systems* 28(3):474–518.
- [Čermák et al. 2014] Čermák, P.; Lomuscio, A.; Mogavero, F.; and Murano, A. 2014. MCMAS-SLK: A Model Checker for the Verification of Strategy Logic Specifications. In *CAV’14*, LNCS 8559, 524–531. Springer.
- [Čermák, Lomuscio, and Murano 2015] Čermák, P.; Lomuscio, A.; and Murano, A. 2015. Verifying and Synthesising Multi-Agent Systems against One-Goal Strategy Logic Specifications. In *AAAI 2015*, 2038–2044. AAAI Press.
- [Chatterjee, Henzinger, and Horn 2009] Chatterjee, K.; Henzinger, T. A.; and Horn, F. 2009. Finitary winning in  $\omega$ -regular games. *ACM Transactions on Computational Logic* 11(1):1.
- [Chatterjee, Henzinger, and Piterman 2010] Chatterjee, K.; Henzinger, T.; and Piterman, N. 2010. Strategy Logic. *Information and Computation* 208(6):677–693.
- [Clarke and Emerson 1981] Clarke, E., and Emerson, E. 1981. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *LP’81*, LNCS 131, 52–71. Springer.
- [Dima and Tiplea 2011] Dima, C., and Tiplea, F. 2011. Model-checking ATL under Imperfect Information and Perfect Recall Semantics is Undecidable. Technical report, arXiv.
- [Emerson and Halpern 1986] Emerson, E., and Halpern, J. 1986. “Sometimes” and “Not Never” Revisited: On Branching Versus Linear Time. *Journal of the ACM* 33(1):151–178.
- [Fijalkow and Zimmermann 2012] Fijalkow, N., and Zimmermann, M. 2012. Cost-parity and cost-streett games. In *FSTTCS*, volume LIPIcs 18, 124–135.
- [Huang and van der Meyden 2014] Huang, X., and van der Meyden, R. 2014. An epistemic strategy logic. *arXiv preprint arXiv:1409.2193*.
- [Huang, Luo, and Van Der Meyden 2011] Huang, X.; Luo, C.; and Van Der Meyden, R. 2011. Improved bounded model checking for a fair branching-time temporal epistemic logic. In *MoChArt*. Springer. 95–111.
- [Jamroga and Ågotnes 2007] Jamroga, W., and Ågotnes, T. 2007. Constructive knowledge: what agents can achieve under imperfect information. *J. Applied Non-Classical Logics* 17(4):423–475.
- [Jamroga and Murano 2015] Jamroga, W., and Murano, A. 2015. Module checking of strategic ability. In *AAMAS 2015*, 227–235.
- [Kupferman and Vardi 1997a] Kupferman, O., and Vardi, M. Y. 1997a. Module checking revisited. In *CAV’97*, 36–47. Springer.
- [Kupferman and Vardi 1997b] Kupferman, O., and Vardi, M. 1997b. Synthesis with incomplete information. In *2nd International Conference on Temporal Logic*, 91–106.
- [Kupferman, Piterman, and Vardi 2009] Kupferman, O.; Piterman, N.; and Vardi, M. Y. 2009. From liveness to promptness. *Formal Methods in System Design* 34(2):83–103.
- [Kupferman, Vardi, and Wolper 2000] Kupferman, O.; Vardi, M.; and Wolper, P. 2000. An Automata Theoretic Approach to Branching-Time Model Checking. *Journal of ACM* 47(2):312–360.
- [Lomuscio, Qu, and Raimondi 2009] Lomuscio, A.; Qu, H.; and Raimondi, F. 2009. MCMAS: A Model Checker for the Verification of Multi-Agent Systems. In *CAV’09*, LNCS 5643, 682–688. Springer.
- [Mogavero et al. 2014] Mogavero, F.; Murano, A.; Perelli, G.; and Vardi, M. 2014. Reasoning About Strategies: On the Model-Checking Problem. *ACM Transactions on Computational Logic* 15(4):34:1–42.
- [Mogavero, Murano, and Sorrentino 2013] Mogavero, F.; Murano, A.; and Sorrentino, L. 2013. On Promptness in Parity Games. In *LPAR’13*, 601–618. Springer.
- [Pnueli and Rosner 1989] Pnueli, A., and Rosner, R. 1989. On the Synthesis of a Reactive Module. In *POPL’89*, 179–190. Association for Computing Machinery.
- [Queille and Sifakis 1981] Queille, J., and Sifakis, J. 1981. Specification and Verification of Concurrent Programs in Cesar. In *International Symposium on Programming’81*, LNCS 137, 337–351. Springer.
- [Reif 1984] Reif, J. H. 1984. The complexity of two-player games of incomplete information. *J. Comput. Syst. Sci.* 29(2):274–301.
- [Sistla and Clarke 1985] Sistla, A. P., and Clarke, E. M. 1985. The complexity of propositional linear temporal logics. *J. ACM* 32(3):733–749.
- [van der Hoek and Wooldridge 2002] van der Hoek, W., and Wooldridge, M. 2002. Tractable Multiagent Planning for Epistemic Goals. In *Autonomous Agents and Multiagent Systems’02*, 1167–1174.
- [Vardi and Wolper 1994] Vardi, M. Y., and Wolper, P. 1994. Reasoning about infinite computations. *Information and Computation* 115(1):1–37.
- [Vester 2013] Vester, S. 2013. Alternating-time temporal logic with finite-memory strategies. In *GandALF 2013*, 194–207.
- [Zimmermann 2013] Zimmermann, M. 2013. Optimal bounds in parametric LTL games. *Theor. Comput. Sci.* 493:30–45.

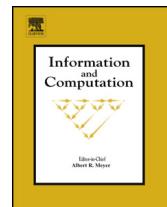
## **4.9 Publication**

The rest of this page is intentionally left blank



Contents lists available at ScienceDirect

## Information and Computation

[www.elsevier.com/locate/yinco](http://www.elsevier.com/locate/yinco)First-cycle games <sup>☆</sup>Benjamin Aminof<sup>a</sup>, Sasha Rubin<sup>b,\*</sup><sup>1</sup><sup>a</sup> Technische Universität Wien, Austria<sup>b</sup> Università degli Studi di Napoli "Federico II", Italy

## ARTICLE INFO

## Article history:

Received 21 November 2014

Available online xxxx

## Keywords:

Graph games  
 Cycle games  
 Memoryless determinacy  
 Parity games  
 Mean-payoff games  
 Energy games

## ABSTRACT

First-cycle games (FCG) are played on a finite graph by two players who push a token along the edges until a vertex is repeated, and a simple cycle is formed. The winner is determined by some fixed property  $Y$  of the sequence of labels of the edges (or nodes) forming this cycle. These games are intimately connected with classic infinite-duration games such as parity and mean-payoff games. We initiate the study of FCGs in their own right, as well as formalise and investigate the connection between FCGs and certain infinite-duration games.

We establish that (for efficiently computable  $Y$ ) the problem of solving FCGs is PSPACE-complete; we show that the memory required to win FCGs is, in general,  $\Theta(n)!$  (where  $n$  is the number of nodes in the graph); and we give a full characterisation of those properties  $Y$  for which all FCGs are memoryless determined.

We formalise the connection between FCGs and certain infinite-duration games and prove that strategies transfer between them. Using the machinery of FCGs, we provide a recipe that can be used to very easily deduce that many infinite-duration games, e.g., mean-payoff, parity, and energy games, are memoryless determined.

© 2016 Elsevier Inc. All rights reserved.

## 1. Introduction

Infinite-duration games are studied in computer science in the context of decidability of logical theories as well as verification and synthesis in formal methods. In particular, reactive systems, which consist of an ongoing interaction between a program and its environment, may be formalised as a game played on a graph, by two players, who push a token along the edges of the graph, resulting in an infinite path, called a play. The winner of the play is determined by some winning condition. For example, in (one version of) mean-payoff games each edge in the graph carries a real number, called a weight, and Player 0 wins the play if and only if the limit-supremum of the running averages of the weights is positive; and otherwise Player 1 wins. Other types of games from the formal verification literature are reachability games, Büchi games, parity games, Muller games, energy games, etc.

Intuitively, certain games on finite graphs can be won using greedy reasoning. For instance, to win a mean-payoff game, it is sufficient for Player 0 to ensure that the average weight of each cycle that is formed is positive. This, in turn, can be ensured by enforcing the average weight of the *first cycle* formed to be positive (because Player 0 could then "forget" that the cycle was formed, and play another cycle with positive average weight, and so on). Thus, in some cases, one can

<sup>☆</sup> Some of the results in this paper were reported in the Proceedings of the Second International Workshop on Strategic Reasoning (SR), April 2014.<sup>\*</sup> Corresponding author.E-mail addresses: [benj@forsyte.at](mailto:benj@forsyte.at) (B. Aminof), [rubin@unina.it](mailto:rubin@unina.it) (S. Rubin).<sup>1</sup> Marie Curie fellow of the Istituto Nazionale di Alta Matematica.

reduce reasoning about infinite-duration games to *first-cycle games*, i.e., games in which the winner is determined by the first cycle on the play. Such reasoning appears in [6], where first-cycle mean-payoff games were defined and used to prove that mean-payoff games can be won using memoryless strategies (i.e., the next move of a player does not depend on the full history up till now, but only on the current node of the play). Memoryless strategies are extremely useful, e.g., they are used to prove deep results in the theory of automata (e.g., Rabin's theorem that automata on infinite trees can be complemented), and to prove upper bounds on the complexity of solving certain classes of games (e.g., that solving parity games is in  $\text{NP} \cap \text{co-NP}$ ).

In this paper we study first-cycle games in their own right and formalise the greedy reasoning above which connects first-cycle games with certain infinite-duration games. We now discuss our main contributions.

**First-cycle games.** We define first-cycle games (FCGs). These games are played on a finite graph (called an arena) by two players who push a token along the edges of the graph until a cycle is formed. Player 0 wins the play if the sequence of labels of the edges (or nodes) of the cycle is in some fixed set  $Y$ , and otherwise Player 1 wins. The set  $Y$  is called a cycle property, and we say that a cycle satisfies  $Y$  if its sequence of labels is in  $Y$ . For example, if every vertex is labelled by an integer priority, and  $Y = \text{cyc-Parity}$  comprises sequences whose largest priority is even, then a cycle satisfies cyc-Parity if the largest priority occurring on the cycle is even. For a fixed cycle property  $Y$ , we write  $\text{FCG}(Y)$  for the family of games over all possible arenas with this winning condition.

**Complexity and memory requirements.** We give a simple example showing that first cycle games (FCGs) are not necessarily memoryless determined. We then show that, for a graph with  $n$  nodes, whereas no winning strategy needs more than  $n!$  memory (since this is enough to remember the whole history of the game), some winning strategies require at least  $(\frac{n-1}{3})!$  memory (Proposition 1, Page 7). We analyse the complexity of solving FCGs and show that it is PSPACE-complete. More specifically, we show that if one can decide in PSPACE whether a given cycle satisfies the property  $Y$ , then solving the games in  $\text{FCG}(Y)$  is in PSPACE; and that there is a trivially computable cycle property  $Y$  for which solving the games in  $\text{FCG}(Y)$  is PSPACE-hard (Theorem 1, Page 8).

**First-cycle games and infinite-duration games (Section 5).** The central object that connects cycle games with infinite-duration games is the *cycles-decomposition* of a path (used for example by [12] to derive the value of a mean-payoff game). Informally, a path is decomposed by pushing the edges of the path onto a stack and, as soon as a cycle is detected in the stack, the cycle is output, popped, and the algorithm continues. This decomposes all but finitely many edges of the path into a sequence of simple cycles.

In order to connect FCGs with infinite-duration games we define the following notion: a winning condition  $W$  (such as the parity winning condition) is  $Y$ -greedy on arena  $\mathcal{A}$  if, in the game on arena  $\mathcal{A}$  with winning condition  $W$ , Player 0 is guaranteed to win by ensuring that every cycle in the cycles-decomposition of the play satisfies  $Y$ , and Player 1 is guaranteed to win by ensuring that every cycle in the cycles-decomposition does not satisfy  $Y$  (Definition 5, Page 14). We prove a *Strategy Transfer Theorem*: if  $W$  is  $Y$ -greedy on  $\mathcal{A}$  then the winning regions in the following two games on arena  $\mathcal{A}$  coincide, and memoryless winning strategies transfer between them: the infinite-duration game with winning condition  $W$ , and the FCG with cycle property  $Y$  (Theorem 7, Page 15).

To illustrate the usefulness of being  $Y$ -greedy, we instantiate the definition to well-studied infinite-duration games such as parity, mean-payoff, and energy games.

**Memoryless determinacy of first-cycle games.** We next address the fundamental question: for which cycle properties  $Y$  is every game in  $\text{FCG}(Y)$  memoryless determined (i.e., no matter the arena)? We provide sufficient and necessary conditions for all games in  $\text{FCG}(Y)$  to be memoryless determined (Theorem 6, Page 14). Although applying this characterisation may not be hard, it involves reasoning about arenas, which is sometimes inconvenient. Therefore, we also provide the following easy-to-check conditions on  $Y$  that ensure memoryless determinacy for all games in  $\text{FCG}(Y)$  (Theorem 9, Page 16):  $Y$  is closed under cyclic permutations (i.e., if  $ab \in Y$  then  $ba \in Y$ ), and both  $Y$  and its complement are closed under concatenation (a set of strings  $X$  is closed under concatenation if  $a, b \in X$  implies  $ab \in X$ ). We demonstrate the usefulness of these conditions by observing that natural cycle properties are easily seen to satisfy them, e.g., cyc-Parity, cyc-MeanPayoff<sub>v</sub> (which states that the limsup average of the weights is at most  $v$ ), and cyc-Energy (which states that the sum of the weights is positive).

**Easy-to-follow recipe.** We integrate the previous results by providing an easy-to-follow recipe that allows one to deduce memoryless determinacy of all classic games that are memoryless determined and many others besides (Section 8). The recipe states that, given a winning condition  $W$ , first “finitise”  $W$  to get a cycle property  $Y$  that is closed under cyclic permutations, then prove that  $W$  is  $Y$ -greedy on the arenas of interest (usually all arenas), and finally, either show that both  $Y$  and its complement are closed under concatenation, or show that  $W$  is prefix-independent. For example, if  $W$  is the parity winning condition (i.e., the largest priority occurring infinitely often is even), the natural “finitisation” of  $W$  is the cycle property  $Y = \text{cyc-Parity}$  (i.e., sequences of priorities whose largest priority is even), and it is almost completely trivial to apply the recipe in this case.

**Related work.** The relationship of our work with that in [6] is as follows. First, our paper deals with qualitative games (i.e., a play is either won or lost) whereas [6] consider quantitative (i.e., a play is assigned a real number, which Player 0 wants to minimize and Player 1 wants to maximize) mean-payoff games. Their main result states that mean-payoff games

are memoryless determined. However, they simultaneously prove that first-cycle games with  $Y = \text{cyc-MeanPayoff}_v$  are memoryless determined. We generalise (in the qualitative setting) both of these facts and their proofs.

Referring to their proofs, [6, Page 111] state: “Our proofs are roundabout, we use the infinite game  $F$  to establish facts about the finite game  $G$  and vice versa. Perhaps more direct proofs would be desirable.” In contrast, the proof of the characterisation of memoryless determined FCGs (Theorems 4, 5 and 6) is direct, and does not go through infinite-duration games. Nonetheless, we use their idea of inducting on the choice nodes of the players, and of employing “reset” nodes in the arena.

We briefly discuss other related work. We point out (in Theorem 3, Page 10) that first-cycle games are not necessarily memoryless determined, even if the cycle property  $Y$  is closed under cyclic permutations contrary to the claim in [4, Page 370]. Our work thus also supplies a correct proof Lemma 4 in [5] which relied on this incorrect claim (see Remark 2, Page 16). Conditions that ensure (or characterise) which winning conditions always admit memoryless strategies appear in [3,8,10]. However, none of these exploit the connection to first-cycle games. For example, [8] give a full characterization of winning conditions for which all infinite-duration games are memoryless determined. Unlike our framework, the main objects of study in their work are winning conditions (i.e., languages of infinite sequences of labels) and one cannot use their results to reason about cycles in an arena since every cycle in an arena induces a cycle in the sequences of labels, but not vice versa. Their characterisation of those winning conditions which admit memoryless determined games is very “infinite” in nature, as it involves reasoning about preference relations among infinite words, and their properties concerning all infinite subsets of words recognizable by nondeterministic automata. On the other hand, their result, that if all solitaire games with a given winning condition are memoryless determined then so are all the two player games, provides a much more useful tool.

The present paper differs from the preliminary version of this work [1] mainly in the following respects: we have re-organised some of the definitions, supplied all proofs, established a full characterisation of those cycle properties  $Y$  such that every FCG over  $Y$  and  $Y$ -greedy games are memoryless determined, and extended the easy-to-follow recipe to include the case that the winning condition is prefix-independent.

## 2. Games

In this paper all games are two-player turn-based games of perfect information played on finite graphs. The players are called Player 0 and Player 1.

**Arenas.** An *arena* is a labelled directed graph  $\mathcal{A} = (V_0, V_1, E, \mathbb{U}, \lambda)$  where

1.  $V_0$  and  $V_1$  are disjoint sets of *vertices* (alternatively, *nodes*) of Player 0 and Player 1, respectively; the set of vertices of the arena is  $V := V_0 \cup V_1$ , and is assumed to be finite and non-empty.
2.  $E \subseteq V \times V$  is a set of *edges* with no dead-ends (i.e., for every  $v \in V$  there is some edge  $(v, w) \in E$ ).
3.  $\mathbb{U}$  is a set of possible *labels* (typical choices for  $\mathbb{U}$  are  $\mathbb{Q}$  and  $\mathbb{N}$ ).
4.  $\lambda : E \rightarrow \mathbb{U}$  is a *labelling function* (which will be used by the winning condition).

A *sub-arena* of  $\mathcal{A}$  is an arena of the form  $(V_0, V_1, E', \mathbb{U}, \lambda')$  where  $E' \subseteq E$  and  $\lambda'(e) = \lambda(e)$  for  $e \in E'$ , i.e., it may have fewer edges. If  $V_0 = \emptyset$  or  $V_1 = \emptyset$  then  $\mathcal{A}$  is called a *solitaire arena*.

**Remark 1.** Note that all the results in this paper also hold for vertex labelled arenas, by labelling every edge  $(v, w)$  by the label of its source  $v$ . In particular, we use vertex labelling in some of our examples, with this transformation in mind. Also note that transforming a vertex labelled arena to an edge-labelled one can be done by inserting an intermediate node in the middle of every edge, and giving it the edge label. However, since this transformation changes the structure of the arena, some properties that hold for vertex labelled arenas do not always transfer to edge-labelled ones.

**Sequences.** Let  $\mathbb{N}$  denote the positive integers. The  $i$ th element (for  $i \in \mathbb{N}$ ) in a sequence  $u$  is denoted  $u_i$ . The *length* of  $u$ , denoted  $|u|$ , is the cardinality of the sequence  $u$ . For  $1 \leq i \leq j \leq |u|$ , write  $u[i, j]$  for the sub-sequence  $u_i u_{i+1} \cdots u_j$ . The empty sequence is denoted  $\epsilon$ . By convention,  $u[i, j] = \epsilon$  if  $j < i$ . The set of infinite sequences over alphabet  $X$  is written  $X^\omega$ , the set of finite sequences is written  $X^*$ . We write concatenation as  $u \cdot v$  or simply as  $uv$ .

**Paths, simple paths, and node-paths.** For an edge  $e = (v, w)$  we call  $v$  the *start* and  $w$  the *end* of  $e$ , and write  $\text{start}(e) := v$  and  $\text{end}(e) := w$ ; we say that  $v$  and  $w$  *appear on the edge*  $e$ . A *path*  $\pi$  in  $\mathcal{A}$  is a finite or infinite sequence of edges  $\pi_1 \pi_2 \cdots \in E^* \cup E^\omega$  such that  $\text{end}(\pi_i) = \text{start}(\pi_{i+1})$  for  $1 \leq i < |\pi|$ . The set of paths in  $\mathcal{A}$  is denoted  $\text{paths}(\mathcal{A})$ . We extend *start* and *end* to paths in the natural way:  $\text{start}(\pi) := \text{start}(\pi_1)$ , and if  $\pi$  is of length  $\ell \in \mathbb{N}$  then  $\text{end}(\pi) := \text{end}(\pi_\ell)$ . We say that vertex  $v$  *appears on the path*  $\pi$  if  $v$  appears on some edge  $\pi_i$  of  $\pi$ . We extend  $\lambda$  from edges to paths point-wise:  $\lambda(\pi)$  is defined to be the string of labels  $\lambda(\pi_1)\lambda(\pi_2)\cdots$ . A path  $\pi$  is called *simple* if  $\text{start}(\pi_i) = \text{end}(\pi_j)$  implies  $i = j + 1$ .

A sequence of nodes  $v \in V^* \cup V^\omega$  is called a *node-path* if  $(v_i, v_{i+1}) \in E$  for all  $i$ . If  $\pi \in E^* \cup E^\omega$  is a path then  $\text{nodes}(\pi)$  is a node-path, where  $\text{nodes}(\pi)$  is defined to be the sequence  $\text{start}(\pi_1)\text{start}(\pi_2)\cdots$  if  $\pi$  is infinite, and  $\text{start}(\pi_1)\cdots\text{start}(\pi_{|\pi|})\text{end}(\pi_{|\pi|})$  if it is finite. Every node-path is either a singleton sequence  $v$  or of the form  $\text{nodes}(\pi)$

for some path  $\pi$ . For a finite non-empty sequence  $u$  of vertices (such as a finite node-path), we overload notation and write  $\text{end}(u)$  for the last node in  $u$ .

**Histories and strategies.** A strategy for a player is a function that tells the player what node to move to given the history, i.e., the sequence of nodes visited so far. Define the set  $H_\sigma(\mathcal{A})$  of *histories* for Player  $\sigma \in \{0, 1\}$  by  $H_\sigma(\mathcal{A}) := \{u \in V^* V_\sigma : (u_n, u_{n+1}) \in E, 1 \leq n < |u|\}$ . In other words,  $H_\sigma(\mathcal{A})$  is the set of node-paths ending in  $V_\sigma$ . A *strategy for Player  $\sigma$*  is a function  $S : H_\sigma(\mathcal{A}) \rightarrow V$  such that  $(\text{end}(u), S(u)) \in E$  for all  $u \in H_\sigma(\mathcal{A})$ . A strategy  $S$  for Player  $\sigma$  is *memoryless* if  $S(u) = S(u')$  for all  $u, u' \in H_\sigma(\mathcal{A})$  with  $\text{end}(u) = \text{end}(u')$ . Hence, we usually consider a memoryless strategy as a function  $S : V_\sigma \rightarrow V$ . A node-path  $u \in V^* \cup V^\omega$  is *consistent* with a strategy  $S$  for Player  $\sigma$  if whenever  $u_n \in V_\sigma$  (for  $n < |u|$ ) then  $u_{n+1} = S(u[1, n])$ . A path  $\pi$  is *consistent* with a strategy  $S$  if  $\text{nodes}(\pi)$  is consistent with  $S$ .

A strategy  $S$  for Player  $\sigma$  is *generated by a Mealy machine*  $(M, m_I, \delta, \mu)$  if there exists a finite set  $M$  of *memory states*, an *initial state*  $m_I \in M$ , a *memory update function*  $\delta : V \times M \rightarrow M$ , and a *next-move function*  $\mu : V_\sigma \times M \rightarrow V$ , such that for a history  $u = u_1 u_2 \dots u_l \in H_\sigma(\mathcal{A})$  we have  $S(u) = \mu(u_l, m_l)$  where  $m_l$  is defined inductively by  $m_1 = m_I$  and  $m_{l+1} = \delta(u_l, m_l)$ . A strategy  $S$  is *finite-memory* if it is generated by some Mealy machine. A strategy  $S$  uses memory at most  $k$  if it is generated by some Mealy machine with  $|M| \leq k$ , and it uses memory at least  $k$  if every Mealy machine generating  $S$  has  $|M| \geq k$ .

**Notational abuse.** We sometimes, for a path  $\pi \in E^*$ , write  $S(\pi)$  instead of  $S(\text{nodes}(\pi))$ .

**Plays.** An infinite path in  $\mathcal{A}$  is called a *play*.<sup>2</sup> We denote the set of all plays of  $\mathcal{A}$  by  $\text{plays}(\mathcal{A})$ . For a node  $v \in V$ , and strategy  $S$ , we write  $\text{plays}_\mathcal{A}(S, v)$  for the set of plays  $\pi$  that start with  $v$  and are consistent with  $S$ ; we write  $\text{reach}_\mathcal{A}(S, v)$  for the set of vertices in  $V$  that appear on the plays in  $\text{plays}_\mathcal{A}(S, v)$ . We may drop the subscript  $\mathcal{A}$  when the arena is clear from the context.

**Games and winning.** A *game* is a pair  $(\mathcal{A}, O)$  consisting of an arena  $\mathcal{A} = (V_0, V_1, E, \mathbb{U}, \lambda)$  and an *objective*  $O \subseteq \text{plays}(\mathcal{A})$ . Objectives are usually determined by winning conditions or cycle properties (defined later). A play  $\pi$  in a game  $(\mathcal{A}, O)$  is *won by Player 0* if  $\pi \in O$ , and otherwise it is *won by Player 1*. A strategy  $S$  for Player  $\sigma$  is *winning from* a node  $v \in V$  (in the game  $(\mathcal{A}, O)$ ) if every play in  $\text{plays}_\mathcal{A}(S, v)$  is won by Player  $\sigma$ . If Player  $\sigma$  has a winning strategy from  $v$  we say that Player  $\sigma$  *wins from*  $v$ . A game  $(\mathcal{A}, O)$  is said to be *determined* if, for every  $v \in V$ , one of the players wins from  $v$ .

The *winning region* (resp. *memoryless winning region*) of Player  $\sigma$  is the set of vertices  $v \in V$  such that Player  $\sigma$  has a winning strategy (resp. memoryless winning strategy) from  $v$ . We denote the winning region in the game  $(\mathcal{A}, O)$  of Player  $\sigma$  by  $\text{WR}^\sigma(\mathcal{A}, O)$ .

**Solitaire games and memoryless strategies.** If either  $V_0$  or  $V_1$  is empty, then the game  $(\mathcal{A}, O)$  is called a *solitaire game*.

A simple property of memoryless strategies that is often used is the following. In a game over an arena  $\mathcal{A} = (V_0, V_1, E, \mathbb{U}, \lambda)$ , every memoryless strategy  $S$  for Player  $\sigma$  induces a sub-arena  $\mathcal{A}^{|S|} = (V_0, V_1, E^{|S|}, \mathbb{U}, \lambda^{|S|})$ , obtained by deleting all edges  $(v, w) \in E$  where  $v \in V_\sigma$  and  $w \neq S(v)$ . Observe that in  $\mathcal{A}^{|S|}$  all Player  $\sigma$  nodes have exactly one outgoing edge (namely the edge specified by  $S$ ), and thus Player  $\sigma$  has no choices to make in this arena. For this reason, many times one considers the solitaire arena  $\mathcal{A}^{\parallel S}$  in which all the nodes are assigned to Player  $1 - \sigma$ , i.e.,  $\mathcal{A}^{\parallel S} = (V_0^{\parallel S}, V_1^{\parallel S}, E^{\parallel S}, \mathbb{U}, \lambda^{\parallel S})$ , where  $V_\sigma^{\parallel S} = \emptyset$ , and  $V_{1-\sigma}^{\parallel S} = V$ . The main connection between  $\mathcal{A}$ ,  $\mathcal{A}^{|S|}$  and  $\mathcal{A}^{\parallel S}$  is that every path in  $\mathcal{A}^{|S|}$  ( $\mathcal{A}^{\parallel S}$ ) is a path in  $\mathcal{A}$  that is also consistent with  $S$ , and vice versa. We can thus reason about paths in  $\mathcal{A}^{|S|}$  (or  $\mathcal{A}^{\parallel S}$ ) instead of paths in  $\mathcal{A}$  that are consistent with  $S$ .

**Memoryless determinacy.** A game is *memoryless from*  $v$  if the player that wins from  $v$  has a memoryless strategy that is winning from  $v$ .

A game is *point-wise memoryless for Player  $\sigma$*  if the memoryless winning region for Player  $\sigma$  coincides with the winning region for Player  $\sigma$ . A game is *uniform memoryless for Player  $\sigma$*  if there is a memoryless strategy for Player  $\sigma$  that is winning from every vertex in that player's winning region.

A game is *point-wise memoryless determined* if it is determined and it is point-wise memoryless for both players. A game is *uniform memoryless determined* if it is determined and uniform memoryless for both players.

**Winning conditions.** A *winning condition* is a set  $W \subseteq \mathbb{U}^\omega$ . If  $W$  is a winning condition and  $\mathcal{A}$  is an arena, the objective  $O^\mathcal{A}(W)$  induced by  $W$  is defined as follows:  $O^\mathcal{A}(W) = \{\pi \in \text{plays}(\mathcal{A}) : \lambda(\pi) \in W\}$ . For ease of readability we may drop the superscript  $\mathcal{A}$  and write  $O(W)$  instead of  $O^\mathcal{A}(W)$ .

Here are some standard winning conditions:

- The *parity condition* consists of those infinite sequences  $c_1 c_2 \dots \in \mathbb{N}^\omega$  such that the largest label occurring infinitely often is even.
- For  $v \in \mathbb{R}$ , the  *$v$ -mean-payoff condition* consists of those infinite sequences  $c_1 c_2 \dots \in \mathbb{R}$  such that  $\limsup_{k \rightarrow \infty} \frac{1}{k} \sum_{i=1}^k c_i$  is at most  $v$ .

<sup>2</sup> For ease of presentation, we consider plays of both finite- and infinite-duration games to be infinite. Obviously, in games finite-duration, games such as FCGs, the winner is determined by a finite prefix of the play, and the moves after this prefix are immaterial.

- The *energy condition*, for a given *initial credit*  $r \in \mathbb{N}$ , consists of those infinite sequences  $c_1c_2\cdots \in \mathbb{Z}^\omega$  such that  $r + \sum_{i=1}^k c_i \geq 0$  for all  $k \geq 1$ .
- The *energy-parity condition* (for a given initial credit  $r$ ) is defined as consisting of  $(c_1, d_1)(c_2, d_2)\cdots \in \mathbb{N} \times \mathbb{Z}$  such that  $c_1c_2\cdots$  satisfies the parity condition and  $d_1d_2\cdots$  satisfies the energy condition with initial credit  $r$ .

### 3. Cycles and games

In this section we define the cycles-decomposition of a path, as well as the first-cycle game, and the infinite-duration all-cycles game.

**Cycles-decomposition.** A *cycle* in an arena  $\mathcal{A}$  is a finite path  $\pi$  such that  $\text{start}(\pi) = \text{end}(\pi)$ . Note that, like paths, cycles are ordered. Hence, the cycles  $(v_1, v_2)(v_2, v_1)$  and  $(v_2, v_1)(v_1, v_2)$  are not identical.

---

#### Algorithm 1 CycDEC( $s, \pi$ ).

```

Require:  $s$  is a finite (possibly empty) simple path                                ▷ initial stack content
Require:  $\pi$  is a finite or infinite path  $\pi_1\pi_2\cdots$                                 ▷ the path to decompose
Require: If  $s$  is non-empty then  $\text{end}(s) = \text{start}(\pi)$                                 ▷  $s\pi$  must form a path

  step = 1
  while  $\text{step} \leq |\pi|$  do
    Append  $\pi_{\text{step}}$  to  $s$                                               ▷ Start a step
    Say  $s = e_1e_2\cdots e_m$                                               ▷ Push current edge into stack
    if  $\exists i : e_i e_{i+1} \cdots e_m$  is a cycle then                                ▷ If stack has a cycle
      Output  $e_i e_{i+1} \cdots e_m$                                          ▷ output the cycle
       $s := e_1 \cdots e_{i-1}$                                               ▷ Pop the cycle from the stack
    end if
     $\text{step} := \text{step} + 1$                                               ▷ advance to next input edge
  end while

```

---

The code appearing in [Algorithm 1](#) defines an algorithm CycDEC that takes as input a (usually empty) simple path  $s$ , which is treated as the initial contents of a stack, and a path  $\pi$  (finite or infinite). At step  $j \geq 1$ , the edge  $\pi_j$  is pushed onto the stack and if, for some  $k$ , the top  $k$  edges on the stack form a cycle, this cycle is output, then popped, and the procedure continues to step  $j + 1$ .

Note that CycDEC may take a finite path  $\pi$  and non-empty stack  $s$  as input. Moreover, it halts if and only if  $\pi$  is a finite path.

The sequence of cycles output by this procedure when input the empty stack  $s = \epsilon$  and path  $\pi$ , denoted  $\text{cycles}(\pi)$ , is called the *cycles-decomposition* of  $\pi$ . The *first cycle* of  $\pi$ , written  $\text{first}(\pi)$ , is the first cycle in  $\text{cycles}(\pi)$ . For example, if

$$\pi = (v, w)(w, x)(x, w)(w, v)(v, s)(s, x)[(x, y)(y, z)(z, x)]^\omega,$$

then

$$\text{cycles}(\pi) = (w, x)(x, w), (v, w)(w, v), (x, y)(y, z)(z, x), (x, y)(y, z)(z, x), \dots,$$

and the first cycle of  $\pi$  is  $(w, x)(x, w)$ .

It is easy to see that, if the algorithm CycDEC is run on a path  $\pi$  in an arena with  $n$  nodes, then at most  $n - 1$  edges of  $\pi$  are pushed but never popped (like the edges  $(v, s)$  and  $(s, x)$  in the example above).<sup>3</sup> So we have:

**Lemma 1.** For every path  $\pi$  in arena  $\mathcal{A}$  with vertex set  $V$ , there are at most  $|V| - 1$  indices  $i$  such that  $\pi_i$  does not appear in any of the cycles in  $\text{cycles}(\pi)$ .

Given a play  $\pi$  in  $\mathcal{A}$ , an initial stack content  $s$ , and  $i \geq 0$ , we write  $\text{stack}^i(s, \pi)$  for the contents of the stack of algorithm CycDEC( $s, \pi$ ) at the beginning of step  $i + 1$  (i.e., just before the edge  $\pi_{i+1}$  is pushed). Note that if  $\text{stack}^i(s, \pi)$  is not empty then it ends with the vertex  $\text{start}(\pi_{i+1})$ , whether or not a cycle was popped during step  $i$ . For  $i \geq 1$ , we define  $\text{cycle}^i(s, \pi)$  to be the cycle output by the algorithm CycDEC( $s, \pi$ ) during step  $i$ , or  $\epsilon$  if no cycle was output at step  $i$ . We may drop  $s$  when  $s = \epsilon$ , and we may drop  $i$  when  $i = |\pi|$ . For instance,  $\text{stack}(\pi)$  is the stack content at the end of the algorithm CycDEC( $\epsilon, \pi$ ) for a finite  $\pi$ ; and  $\text{cycle}^i(\pi)$  is the cycle output during step  $i$  of the algorithm CycDEC( $\epsilon, \pi$ ).

Given that, for every  $i \geq 1$ , the behaviour of the algorithm CycDEC( $s, \pi$ ) from the end of step  $i$  onwards is completely determined by  $\text{stack}^i(s, \pi)$ , and the suffix  $\pi_{i+1}\pi_{i+2}\dots$  of  $\pi$ , the following lemma is immediate:

**Lemma 2.** Let  $\pi$  be a play in  $\mathcal{A}$ , let  $i \geq 0$ , let  $w = \text{stack}^i(\pi)$ , and let  $\pi' = \pi_{i+1}\pi_{i+2}\dots$  and  $\pi'' = w \cdot \pi'$ . Then for every  $j \geq 0$  we have that

<sup>3</sup> As we show in Section 8, this allows one to reason, for instance, about the initial credit problem for energy games.

1.  $\text{stack}^{i+j}(\pi) = \text{stack}^j(w, \pi') = \text{stack}^{|w|+j}(\pi'')$ , and
2.  $\text{cycle}^{i+j}(\pi) = \text{cycle}^j(w, \pi') = \text{cycle}^{|w|+j}(\pi'')$ .

Furthermore, for every  $0 \leq l \leq |w|$ , we have that  $\text{stack}^l(\pi'') = w_1 \dots w_l$ , and  $\text{cycle}^l(\pi'') = \epsilon$ .

**Cycle properties.** For a given  $\mathbb{U}$ , a *cycle property* is a set  $Y \subseteq \mathbb{U}^*$ , used later on to define winning conditions for games.<sup>4</sup> Here are some cycle properties that we refer to in the rest of the article:

1. Let cyc-EvenLen be those sequences  $c_1 c_2 \dots c_k \in \mathbb{U}^*$  such that  $k$  is even.
2. Let cyc-Parity be those sequences  $c_1 \dots c_k \in \mathbb{N}^*$  such that  $\max_{1 \leq i \leq k} c_i$  is even.
3. Let cyc-Energy be those sequences  $c_1 \dots c_k \in \mathbb{Z}^*$  such that  $\sum_{i=1}^k c_i \geq 0$ .
4. Let cyc-GoodForEnergy be those sequences  $(c_1, d_1) \dots (c_k, d_k) \in (\mathbb{N} \times \mathbb{Z})^*$  such that  $\sum_{i=1}^k d_i > 0$ , or both  $\sum_{i=1}^k d_i = 0$  and  $c_1 \dots c_k \in \text{cyc-Parity}$ .
5. Let cyc-MeanPayoff $_\nu$  (for  $\nu \in \mathbb{R}$ ) be those sequences  $c_1 \dots c_k \in \mathbb{R}^*$  such that  $\frac{1}{k} \sum_{i=1}^k c_i \leq \nu$ .
6. Let cyc-MaxFirst be those sequences  $c_1 \dots c_k \in \mathbb{N}^*$  such that  $c_1 \geq c_i$  for all  $i$  with  $1 \leq i \leq k$ .
7. Let cyc-EndsZero be those sequences  $c_1 \dots c_k \in \mathbb{N}_0^*$  such that  $c_k = 0$ .

If  $Y \subseteq \mathbb{U}^*$  is a cycle property, write  $\neg Y$  for the cycle property  $\mathbb{U}^* \setminus Y$ . We will usually use  $Y$  to define goals for Player 0 (hence Player 1's goals would be naturally associated with  $\neg Y$ ). It is convenient to define  $Y^0 = Y$ , and  $Y^1 = \neg Y$ . This allows us to refer to the goals of Player  $\sigma$  in terms of  $Y^\sigma$ .

We isolate two important classes of cycle properties (the first is inspired by [4]):

1. Say that  $Y$  is *closed under cyclic permutations* if  $ab \in Y$  implies  $ba \in Y$ , for all  $a \in \mathbb{U}, b \in \mathbb{U}^*$ .
2. Say that  $Y$  is *closed under concatenation* if  $a \in Y$  and  $b \in Y$  imply that  $ab \in Y$ , for all  $a, b \in \mathbb{U}^*$ .

Note, for example, that the cycle properties 1–5 above are closed under cyclic permutations and concatenation; and that  $\neg \text{cyc-EvenLen}$  is closed under cyclic permutations but not under concatenation.

If  $\mathcal{A}$  is an arena with labelling  $\lambda$ , and  $C$  is a cycle in  $\mathcal{A}$ , we say that a *cycle  $C$  satisfies  $Y$*  if  $\lambda(C) \in Y$ .

### First-cycle games and all-cycles games.

For arena  $\mathcal{A} = (V_0, V_1, E, \mathbb{U}, \lambda)$  and cycle property  $Y \subseteq \mathbb{U}^*$ , we define two objectives (subsets of  $\text{plays}(\mathcal{A})$ ):

1.  $\pi \in O_{\text{first}}^{\mathcal{A}}(Y)$  if  $\text{first}(\pi)$  satisfies  $Y$ .
2.  $\pi \in O_{\text{all}}^{\mathcal{A}}(Y)$  if every cycle in  $\text{cycles}(\pi)$  satisfies  $Y$ .

To ease readability, we drop the superscript  $\mathcal{A}$  when the arena is clear from the context and write, for example,  $O_{\text{all}}(Y)$  instead of  $O_{\text{all}}^{\mathcal{A}}(Y)$ .

The game  $(\mathcal{A}, O_{\text{first}}(Y))$  is called a *first-cycle game (over  $Y$ )*, and the family of all first-cycle games over  $Y$  (i.e., taking all possible arenas) is denoted  $\text{FCG}(Y)$ . Similarly, we write  $\text{ACG}(Y)$  for the family of *all-cycles games* over  $Y$ . For instance,  $\text{FCG}(\text{cyc-Parity})$  consists of those games such that Player 0 wins iff the largest label occurring on the first cycle is even.

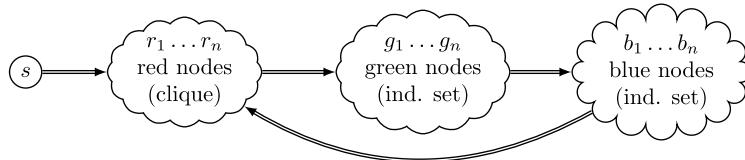
A game  $(\mathcal{A}, O)$  is *finitary* if every play is already won after a finite number of steps, i.e., for every  $\pi \in \text{plays}(\mathcal{A})$  there exists  $K_\pi \in \mathbb{N}$  such that, if Player  $\sigma$  wins the play  $\pi$ , then Player  $\sigma$  also wins every play of  $\mathcal{A}$  starting with the prefix  $\pi[1, K_\pi]$ . Clearly FCGs are finitary since the winner is already determined by the first-cycle of the play (recall that arenas are finite). A basic result states that finitary games (of perfect information) are determined.<sup>5</sup> We recap the proof because we use it in [Theorem 1](#) to determine the computational complexity of deciding which player has a winning strategy in a given FCG.

**Lemma 3.** Every finitary game  $(\mathcal{A}, O)$  is determined.

**Proof.** Suppose  $(\mathcal{A}, O)$  is finitary. Fix a starting node  $v$  and note that the set of finite node-paths starting in  $v$  form a tree, which we call the *game-tree*. Plays in  $\mathcal{A}$  correspond to (infinite) branches in the game-tree. Prune every branch  $\pi$  of the game-tree at its  $K_\pi$ th node to get the pruned game-tree  $T$ . By König's Tree Lemma, the pruned game-tree  $T$  is finite since it is finitely branching and contains no infinite paths. Turn  $T$  into an And-Or tree: label an internal node  $\rho$  by  $\vee$  in case  $\text{end}(\rho) \in V_0$ , and by  $\wedge$  in case  $\text{end}(\rho) \in V_1$ ; label a leaf  $\rho$  by *true* if every play extending  $\rho$  is won by Player 0, and by *false* if every play extending  $\rho$  is won by Player 1. Note that every leaf gets a label. It is easy to see that Player 0 has a winning strategy from  $v$  in the game  $(\mathcal{A}, O)$  if and only if the And-Or tree evaluates to *true*.  $\square$

<sup>4</sup> When  $\mathbb{U}$  is clear from the context, we will not mention it.

<sup>5</sup> This is usually attributed to Zermelo, but also follows from the fact that open games are determined, e.g., [7].



**Fig. 1.** double arrows represent one edge from every node to every node.

#### 4. First-cycle games

##### 4.1. Memory bounds and complexity

In this section we discuss the amount of memory required to win FCGs and the computational complexity of deciding which player has a winning strategy.

We begin by showing that while strategies with memory  $2^{O(|V| \log |V|)}$  always suffice, there are FCGs that require  $2^{\Omega(|V| \log |V|)}$  memory.

##### Proposition 1.

1. For a FCG on an arena with  $n$  vertices, if Player  $\sigma$  wins from  $v$ , then every winning strategy for Player  $\sigma$  starting from  $v$  uses memory at most  $n!$ .
2. For every  $n \in \mathbb{N}$  there exists a FCG on an arena with  $3n + 1$  vertices, and a vertex  $v$ , such that every winning strategy for Player 0 from  $v$  uses memory at least  $n!$ .

**Proof.** For the upper bound, note that it is enough to remember the whole history of the play until a cycle is formed, i.e., all histories of length at most  $n - 1$ . To store all histories of length  $k$  requires  $\sum_{i \leq k} i! < (k + 1)!$  many states. Thus,  $n!$  memory suffices.

We now turn to the lower bound.

**Sketch.** We define a game in which Player 1 can select any possible permutation of  $\{1, \dots, n\}$  and, in order to win, Player 0 must remember this permutation. Consider the arena in Fig. 1. Intuitively, the red nodes are used by Player 1 to select a permutation of  $\{1, \dots, n\}$ , and the green and blue nodes are used by her to query Player 0 as to the relative order of any pair of indices  $i, j$  in this permutation. The outgoing edges from the blue nodes are used by Player 0 to respond to the query by going back to either  $r_i$  or  $r_j$ . Player 0 wins the game if the cycle contains both  $r_i$  and  $r_j$ , hence, he must remember the order of the pair  $i, j$  in the permutation. Since this is true for every such pair of indices, he must use at least  $n!$  memory to store the permutation. The winning condition captures the above intuition, and also ensures that Player 1 loses if she deviates from the above protocol.

**Details.** The arena  $\mathcal{A} = (V_0, V_1, E, \mathbb{U}, \lambda)$  has  $n$  red nodes  $r_1, \dots, r_n$ ,  $n$  green nodes  $g_1, \dots, g_n$ ,  $n$  blue nodes  $b_1, \dots, b_n$ , and an initial node  $s$ . The red nodes form a clique (i.e.,  $(r_i, r_j) \in E$  for every  $1 \leq i \neq j \leq n$ ), the green nodes form an independent set, and so do the blue nodes (i.e.,  $(g_i, g_j) \notin E$  and  $(b_i, b_j) \notin E$  for every  $1 \leq i, j \leq n$ ). In addition, for every  $1 \leq i, j \leq n$  we have the edges  $(s, r_j), (r_i, g_j), (g_i, b_j), (b_i, r_j)$ . Player 0 owns the blue nodes ( $V_0 = \{b_1, \dots, b_n\}$ ), and the rest belong to Player 1. In order to correctly describe the winning condition, we label the nodes as follows: for every  $1 \leq i \leq n$ , the nodes  $r_i, g_i, b_i$  are labelled  $(Red, i), (Green, i), (Blue, i)$ , respectively; and the node  $s$  is labelled by 0. In other words,  $\mathbb{U} = \{0\} \cup \{(Red, Green, Blue) \times \{1, \dots, n\}\}$ , and the induced edge labelling is: for every  $(v, w) \in E$ , we have that  $\lambda(v, w) = (Red, i)$  if  $v = r_i$ ,  $\lambda(v, w) = (Green, i)$  if  $v = g_i$ ,  $\lambda(v, w) = (Blue, i)$  if  $v = b_i$ , and otherwise  $\lambda(v, w) = 0$ .

We now define the winning condition. Player 0 wins a play iff the first cycle  $(v_1, v_2) \dots (v_k, v_{k+1})$  on the play satisfies one of the following three conditions<sup>6</sup>:

1. the node just before the end is not blue (i.e.,  $v_k \notin \{b_1, \dots, b_n\}$ ); or
2. it does not have exactly 1 green node and 1 blue node; or
3. exactly 1 green node  $g_j$ , and 1 blue node  $b_l$  appear on it, and  $b_l = v_k$ , and
  - (a) it has red nodes labelled with the same numbers as  $g_j$  and  $b_l$  (i.e.,  $r_j, r_l$  are on the cycle), and
  - (b) it starts with a red node labelled with a number equal to that of the green or the blue node (i.e.,  $v_1 \in \{r_j, r_l\}$ ).

Informally, the first two conditions above ensure that Player 0 can win if Player 1 does not select a permutation followed by a query, whereas the third condition ensures that if Player 1 does, then Player 0 can win but only if he remembers the whole permutation.

<sup>6</sup> For simplicity, we state the winning condition in terms of the nodes on the cycle and not in terms of the labels of the edges. However, since we essentially label an edge by the name of its source node, the latter can be easily done.

We first prove that Player 0 has a winning strategy.<sup>7</sup> We distinguish between two types of plays:

- (i) Player 1 selects a permutation through the red nodes, and then queries Player 0 by going to some green node  $g_j$ , followed by a blue node  $b_l$ ; i.e.,  $u = (s, r_{i_1})(r_{i_1}, r_{i_2}) \dots (r_{i_{n-1}}, r_{i_n})(r_{i_n}, g_j)(g_j, b_l)$  are the first  $n+2$  edges of the play, and for every  $1 \leq x \leq n$  we have that  $x = i_h$  for some  $1 \leq h \leq n$ . In this case, let  $1 \leq \tilde{j}, \tilde{l} \leq n$  be such that  $i_{\tilde{j}} = j$  and  $i_{\tilde{l}} = l$ , and observe that Player 0 can win by taking the edge  $(b_l, r_{i_h})$  where  $h = \min(\tilde{j}, \tilde{l})$  (i.e., by taking the edge back to the node among  $r_i, r_j$  that appears sooner on  $u$ ). Indeed, the first cycle of the play will then be  $(r_{i_h}, r_{i_{h+1}}) \dots (r_{i_{n-1}}, r_{i_n})(r_{i_n}, g_j)(g_j, b_l)$ , and it will satisfy the third option in the winning condition (in particular, by our choice of  $h$ , both  $r_i$  and  $r_j$  are on the cycle, and  $i_h \in \{j, l\}$ ).
- (ii) The play never reaches a blue node, or when the play reaches a blue node for the first time it does not conform to the pattern given in case (i) above. If the play never reaches a blue node then Player 0 wins since the first cycle formed satisfies the first option in the winning condition. This is also true if a cycle was formed before the play reaches a blue node for the first time. Hence, we are left with the option that the prefix of the play is of the form  $u = (s, r_{i_1})(r_{i_1}, r_{i_2}) \dots (r_{i_{t-1}}, r_{i_t})(r_{i_t}, g_j)(g_j, b_l)$ , where  $t < n$ , and for  $x \neq y$  we have  $r_{i_x} \neq r_{i_y}$ . In this case, Player 0 first takes the edge  $(b_l, r_m)$ , where  $1 \leq m \leq n$  is the index of a red node that did not yet appear on the play (i.e.,  $i_x \neq m$  for all  $1 \leq x \leq t$ ). Note that this does not yet form a cycle and the play continues. Now, the game can proceed in three ways, all losing for Player 1: the first is by keeping the play forever away from blue nodes, thus closing a cycle satisfying option 1 in the winning condition; the second is by closing the first cycle by going back to  $b_l$ , again losing by option 1 (since then  $v_k$  is green); the third is by reaching some blue node  $b_h$  different from  $b_l$  without yet forming a cycle. In this last case, Player 0 wins (using the second option in the winning condition) by taking the edge  $(b_h, r_{i_t})$  and thus forming a cycle that contains two different blue nodes:  $b_l, b_h$ .

We now show that every strategy for Player 0 that uses less than  $n!$  memory is not winning. Let  $\xi$  be a Player 0 strategy that is implemented using a Mealy machine  $\mathcal{M}$  with less than  $n!$  states. Hence, there are two different permutations  $p = i_1, \dots, i_n$  and  $p' = i'_1, \dots, i'_n$  of  $\{1, \dots, n\}$ , such that  $\mathcal{M}$  is in the same state  $m$  after reading the history  $s \cdot r_{i_1} \dots r_{i_n}$ , as after reading the history  $s \cdot r_{i'_1} \dots r_{i'_n}$ . Let  $h$  be the smallest index such that  $i_h \neq i'_h$ , and let  $j := i_h$  and  $l := i'_h$ , and let  $1 \leq \tilde{j}, \tilde{l} \leq n$  be such that  $i'_{\tilde{j}} = j$  and  $i'_{\tilde{l}} = l$ . Simply put,  $j$  appears in position  $h$  in  $p$  and position  $\tilde{j}$  in  $p'$ , whereas  $l$  appears in position  $h$  in  $p'$  and position  $\tilde{l}$  in  $p$ . The minimality of  $h$  implies that  $\tilde{j}, \tilde{l} > h$ , and thus  $l$  appears after  $j$  in  $p$ , but before  $j$  in  $p'$ .

Observe that the two histories  $\rho = s \cdot r_{i_1} \dots r_{i_n} \cdot g_j \cdot b_l$ , and  $\rho' = s \cdot r_{i'_1} \dots r_{i'_n} \cdot g_j \cdot b_l$  also put  $\mathcal{M}$  in the same state and thus,  $\xi$  responds to both histories with the same move, say by taking the edge  $(b_l, r_x)$ . Observe that, since  $p, p'$  are permutations, every red node already appears on  $\rho, \rho'$ . Hence, in both cases, every possible move of Player 0 from  $b_l$  closes a cycle that has exactly one green and one blue node, and the blue node appears just before the end. It follows that, in both cases, in order to win Player 0 must satisfy items 3.a and 3.b of the winning condition. In order to satisfy 3.a he must choose  $r_x = r_l$  or  $r_x = r_j$ . However, that prevents him from satisfying item 3.b in both cases since  $r_j$  appears on  $\rho$  before  $r_l$ , but after it on  $\rho'$ . More formally, consider first the option  $r_x = r_l$  and the history  $\rho$ . Recall that  $i_{\tilde{l}} = l$ , that  $j = i_h$ , and that  $\tilde{l} > h$ . Hence, the first cycle formed is  $(r_{i_{\tilde{l}}}, r_{i_{\tilde{l}+1}}) \dots (r_{i_{n-1}}, r_{i_n})(r_{i_n}, g_j)(g_j, b_l)(b_l, r_{i_{\tilde{l}}})$ , and it does not contain the required  $r_j$ . Similarly, if  $r_x = r_j$  consider the history  $\rho'$ , and note that the first cycle formed is  $(r_{i'_j}, r_{i'_{j+1}}) \dots (r_{i'_{n-1}}, r_{i'_n})(r_{i'_n}, g_j)(g_j, b_l)(b_l, r_{i'_j})$ , and it does not contain the required  $r_l$ . It follows that  $\xi$  is not a winning strategy.  $\square$

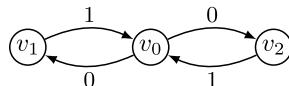
We now address the complexity of solving FCGs with efficiently computable cycle properties. Given a game, and a starting node  $v$ , solving the game is the problem of deciding whether Player 0 or Player 1 wins from  $v$ . We show that this problem is in general PSPACE-complete.

### Theorem 1.

1. If  $Y$  is a cycle property for which solving membership is in PSPACE then the problem of solving games in  $\text{FCG}(Y)$  is in PSPACE.
2. There exist a cycle property  $Y$ , that is computable in linear-time, such that the problem of solving games in  $\text{FCG}(Y)$  is PSPACE-hard.

**Proof.** Let  $\mathcal{A}$  be an arena with  $n$  nodes. Following the proof of Lemma 3, for the special case of FCGs, prune a branch of the game-tree once the first cycle is formed, and label the game-tree to get an And-Or tree. Every node in the And-Or tree is of depth at most  $n$ , and has at most  $n$  children. Hence, evaluating the tree can be done, using a depth-first algorithm, in space polynomial in  $n$  plus the maximal space required to compute the labels of the leaves  $\rho$  in the tree. Note that the label of a leaf  $\rho$  is determined by whether  $\lambda(c) \in Y$  or not (where  $c$  is the first-cycle on  $\rho$ ). By our assumption on  $Y$  this can be done in polynomial space, hence the upper bound follows.

<sup>7</sup> We define the strategy only for histories that are consistent with it. Obviously, it can be arbitrarily defined on any other history.



**Fig. 2.** Sample arena  $\mathcal{A}$ . Circles are Player 0 nodes, solid lines are edges.

The lower bound follows immediately by reducing the PSPACE-hard problem QBF to solving  $\text{FCG}(Y)$ , where  $Y$  is the set of all sequences whose last two elements are identical. The proof uses the same arena (which is vertex-labelled – see Remark 1) as in the reduction of QBF to Generalised Geography (e.g., [11, Theorem 8.11]). To quickly recap, the basic idea there is to view QBF as a game in which Player 0 (resp. Player 1) assigns the values of an existentially (resp. universally) quantified variable  $x_i$  by picking a node labelled by the literal  $x_i$  or the literal  $\neg x_i$ ; after all variables are thus assigned, Player 1 picks a clause (challenging Player 0 to show that it is true), and finally Player 0 responds by picking a literal of this clause (that he claims is true). The structure of the arena is such that the only outgoing edge from the node picked by Player 0 in this last step is the node labelled by the same literal that was available during the value-assigning phase. Thus, if indeed that literal was picked in the assignment phase, then the next move closes a cycle that satisfies  $Y$  and otherwise, taking this edge does not close a cycle, forcing Player 0 to close a losing cycle in the next step. Overall, the QBF formula is true iff Player 0 wins this FCG.  $\square$

#### 4.2. The relation between point-wise and uniform memoryless determinacy

In this section we consider the difference between point-wise memoryless determinacy and uniform memoryless determinacy in FCGs. We begin by considering the simple case of solitaire games.

**Theorem 2.** All solitaire FCGs are point-wise memoryless determined. However, some solitaire FCG's are not uniform memoryless determined.

**Proof.** The fact that a solitaire FCG is point-wise memoryless determined is simply because once a node repeats the game is effectively over, and thus the player makes at most one meaningful move from any node. In other words, Player  $\sigma$  can win from a node  $v$  iff there is a play  $\pi$  starting in  $v$  such that the first cycle  $\pi_i \dots \pi_j$  on  $\pi$  satisfies  $Y^\sigma$ . Traversing the prefix  $\pi_1 \dots \pi_j$  requires no memory since each vertex on it is the source of exactly one edge. For the second item, consider the arena in Fig. 2. Observe that, for the cycle property  $Y = \text{cyc-EndsZero}$ , no memoryless strategy is winning from both  $v_1$  and  $v_2$ .  $\square$

We now consider two-player games. In contrast with solitaire games, some two-player FCGs are not point-wise memoryless determined. Indeed, any solitaire game (in which all nodes belong to Player 0) that is not uniform memoryless determined, can be turned into a two player game in which Player 0 wins from some node  $w$ , but requires memory to do so: simply add a single Player 1 node  $w$  that has outgoing edges to all nodes in the original game.

As we later show (Corollary 1, Page 11), if a cycle property  $Y$  is closed under cyclic permutations then all solitaire games in  $\text{FCG}(Y)$  are uniform memoryless determined. Unfortunately, for two player games, this is not enough even for point-wise memoryless determinacy. Indeed, Theorem 3 shows that closure of  $Y$  under cyclic permutations is a necessary condition for having all games in  $\text{FCG}(Y)$  be point-wise memoryless determined, but it is not a sufficient one.<sup>8</sup>

We also show that, for cycle properties  $Y$  that are closed under cyclic permutations, if a game in  $\text{FCG}(Y)$  is point-wise memoryless for a player then it is also uniform memoryless for this player (Theorem 3 Item 3). The proof of this fact uses the following Lemma.

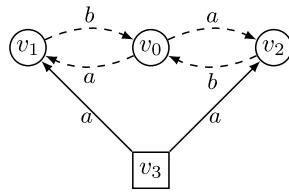
**Lemma 4.** Suppose  $Y$  is closed under cyclic permutations, and let  $S$  be a strategy for Player  $\sigma$  in arena  $\mathcal{A}$ . If  $S$  is winning from  $v$ , then  $S$  is winning from every node  $w \in \text{reach}_{\mathcal{A}}(S, v)$ .

**Proof.** We begin with the intuition.

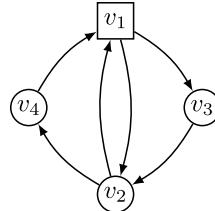
**Sketch.** If  $c$  is the first cycle of some play  $\pi \in \text{plays}(S, w)$ , then we can construct a path  $\pi'$  starting in  $v$  and consistent with  $S$  whose first cycle is some cyclic permutation  $c'$  of  $c$ , as follows: we proceed along some simple path from  $v$  to  $w$  until we hit a node on  $\pi$ ; if this node is not on  $c$ , we continue along  $\pi$  until we reach a node on  $c$ ; and finally, we cycle along the nodes of  $c$  until we return to where we started, forming a cycle  $c'$ . Note that  $c'$  is not necessarily identical to  $c$  since in the first stage we may have hit  $\pi$  somewhere in the middle of  $c$ . Since  $c' = \text{first}(\pi') \in Y^\sigma$ , and  $Y^\sigma$  is closed under cyclic permutations (note that  $Y$  is closed under cyclic permutations if and only if  $\neg Y$  is closed under cyclic permutations), we get that  $\pi$  is won by Player  $\sigma$ .

**Details.** It is enough to show that in the arena  $\mathcal{A}^{|S|}$ , for every play  $\pi$  starting in  $w$ , there is a path  $\pi'$  starting in  $v$ , such that the first cycle of  $\pi'$  is a cyclic permutation of the first cycle  $\pi_m \dots \pi_n$  of  $\pi$ . We construct  $\pi'$  as follows. Let  $\rho$  be a path

<sup>8</sup> This result corrects a mistake in [4] that claims that this is a sufficient condition (note that they do not address the question of whether it is necessary).



**Fig. 3.** Sample arena  $B$ . Circles are Player 0 nodes, squares are Player 1 nodes, solid lines are edges, dashed lines represent (sequences of edges that are) simple paths.



**Fig. 4.** Sample arena  $C$ . Circles are Player 0 nodes, squares are Player 1 nodes, solid lines are edges.

in  $\mathcal{A}^{|S|}$  from  $v$  to  $w$  (such a path exists since  $w \in \text{reach}(S, v)$ ). Let  $\ell$  be the smallest index such that  $\rho_\ell$  intersects  $\pi[1, n]$ , i.e., such that  $\text{end}(\rho_\ell) = \text{start}(\pi_j)$  for some  $j \leq n$ . Note that  $\ell$  is well defined since  $\text{end}(\rho) = w = \text{start}(\pi_1)$ . We consider two cases (depending on whether or not  $\rho$  first intersects  $\pi$  before  $\pi$  starts traversing  $\text{first}(\pi)$ ).

Case  $j \leq m$ . Let  $\pi' = \rho_1 \dots \rho_\ell \pi_j \dots \pi_n$ , and note that  $\text{first}(\pi') = \text{first}(\pi)$ .

Case  $m < j \leq n$ . Let  $\pi' = \rho_1 \dots \rho_\ell \pi_j \dots \pi_n \pi_m \dots \pi_{j-1}$ , and note that  $\text{first}(\pi') = \pi_j \dots \pi_n \pi_m \dots \pi_{j-1}$  which is a cyclic permutation of  $\text{first}(\pi)$ .  $\square$

### Theorem 3.

1. If  $Y$  is not closed under cyclic permutations then there is a game in  $\text{FCG}(Y)$  that is not point-wise memoryless determined.
2. There exists a cycle property  $Y$ , closed under cyclic permutations, and a game in  $\text{FCG}(Y)$  that is not point-wise memoryless determined.
3. If a cycle property  $Y$  is closed under cyclic permutations, then every game in  $\text{FCG}(Y)$  that is point-wise memoryless for Player  $\sigma$  is also uniform memoryless for Player  $\sigma$ .

**Proof.** For the first item, assume that  $Y$  is not closed under cyclic permutations, and let  $a, b \in \mathbb{U}^*$  be such that  $ab \in Y$  but  $ba \notin Y$ . Consider the arena in Fig. 3. Observe that Player 0 wins from  $v_3$ , but in order to do so he needs to remember if the play arrived to  $v_0$  from  $v_1$  or from  $v_2$ .

For the second item, consider the arena in Fig. 4, and the cycle property  $Y = \text{cyc-EvenLen}$ . Obviously,  $Y$  is closed under cyclic permutations. However, starting at  $v_1$ , Player 0 has a winning strategy, but no memoryless winning strategy – when choosing the outgoing edge from  $v_2$  the player needs to remember if the previous node was  $v_1$  (in which case he should return to  $v_1$ ), or  $v_3$  (in which case he should go to  $v_4$ ).

For the third item, write  $W_\sigma := \text{WR}^\sigma(\mathcal{A}, O)$ , and assume that the game is point-wise memoryless for Player  $\sigma$ , and for every  $v \in W_\sigma$  let  $S_v$  be a memoryless winning strategy for Player  $\sigma$  from  $v$ .

**Sketch.** The idea is to define a memoryless strategy  $S$  for Player  $\sigma$  that is winning from every node  $v \in W_\sigma$  by considering the nodes of  $W_\sigma$  (in some arbitrary order), and if  $v$  is the next node in  $W_\sigma$  for which  $S(v)$  is not yet defined, then define  $S(w) := S_v(w)$  for all  $w \in \text{Reach}(S_v, v)$  that have not yet been defined. Thus,  $S$  is memoryless by construction. The main work is to show that it is winning (this is where Lemma 4 is used).

**Details.** Fix some arbitrary linear ordering  $v_1 \prec v_2 \prec \dots \prec v_n$  of the nodes in  $W_\sigma$ . We iteratively build a memoryless strategy  $S$  for Player  $\sigma$  that is winning from every node  $v \in W_\sigma$ . At each round  $j \geq 0$  of this construction, we write  $V^j \subseteq V$  for the set of nodes considered by the end of that round, and have that  $S$  is defined for every node in  $V^j \cap W_\sigma$ . At round 0, we begin with  $V^0 = \emptyset$ , and with  $S(v)$  undefined for all  $v \in W_\sigma$ . At round  $j > 0$ , if  $V^{j-1} = V$  then we are done, and otherwise we proceed as follows:

- (1) If  $W_\sigma \not\subseteq V^{j-1}$ , let  $v$  be the smallest vertex (by the ordering  $\prec$ ) such that  $v \in W_\sigma \setminus V^{j-1}$ . Set  $V^j := V^{j-1} \cup \text{reach}(S_v, v)$ , and for every  $w \in (V_\sigma \cap \text{reach}(S_v, v)) \setminus V^{j-1}$  define  $S(w) := S_v(w)$ .
- (2) If  $W_\sigma \subseteq V^{j-1}$ , then set  $V^j = V$ , and for every node  $v \in V_\sigma \setminus V^{j-1}$  define  $S(v)$  arbitrarily. Intuitively, moves from these nodes are unimportant since they are not reachable from any node in  $W_\sigma$  on any play consistent with  $S$ .

It is easy to see that  $S$  is well defined and memoryless.

It remains to show that  $S$  is winning from every  $w \in W_\sigma$ . The proof is by induction on the round number  $j$ . The induction hypothesis is that (i) if  $w \in V^j$  then  $\text{reach}(S, w) \subseteq V^j$  and; (ii) if  $W_\sigma \not\subseteq V^{j-1}$  then  $S$  is winning from every  $w \in V^j$ .

Observe that, by taking the maximal  $j$  for which  $W_\sigma \not\subseteq V^{j-1}$ , item (ii) in the induction hypothesis implies that  $S$  is winning from every  $w \in W_\sigma$ .

For  $j = 0$ , the hypothesis is trivially true. Assume that the hypothesis holds for all  $0 \leq l < j$ , and consider round  $j$ . If  $W_\sigma \subseteq V^{j-1}$ , then (i) is true since the construction uses case (2) and sets  $V^j = V$ , and (ii) is trivially true. If, on the other hand,  $W_\sigma \not\subseteq V^{j-1}$  (i.e., the construction uses cases (1)), then let  $w \in \text{reach}(S_v, v)$  be some node added at round  $j$ . Note that to prove that (i) holds, it is enough to show that every node  $w'$ , that is reachable in  $\mathcal{A}^{|S|}$  from  $w$ , was added before or at round  $j$ . In other words, we have to show that  $\text{reach}(S, w) \subseteq (V^{j-1} \cup \text{reach}(S_v, v))$ . This can be easily done by inducting on the length of the node-path  $\rho = v_1 \dots v_k \in V^*$  from  $w$  to  $w'$  in  $\mathcal{A}^{|S|}$ . For  $k = 0$  this is trivially true. Assume it is true for  $\rho$  of length  $k$ , and consider  $\rho$  of length  $k + 1$ . Now, if  $v_k$  was added at a previous round, then so did  $v_{k+1}$  by (i) in the induction hypothesis applied to  $v_k$ . Otherwise,  $v_k$  was added at this round (and thus  $v_k \in \text{reach}(S_v, v)$ ), in which case if  $v_k$  belongs to the opponent (i.e.,  $v_k \in V_{1-\sigma}$ ) then all its successors, and in particular  $v_{k+1}$ , are in  $\text{reach}(S_v, v)$ ; and if  $v_k \in V_\sigma$  then  $v_{k+1} = S(v_k) = S_v(v_k) \in \text{reach}(S_v, v)$ . To complete the proof, we have to show that (ii) holds, i.e., that  $S$  is winning from  $w$ .

To see that  $S$  is winning from  $w$ , take any play  $\pi \in \text{plays}(S, w)$ , and let  $\pi_m \dots \pi_n$  be the first cycle of  $\pi$ . There are two options: either the prefix  $\pi_1 \dots \pi_n$  is consistent with  $S_v$ , or it isn't. If it is, since  $S_v$  is winning for Player  $\sigma$  from every node in  $\text{reach}(S_v, v)$  (Lemma 4), and thus in particular from  $w$ , we have that  $\pi$  is won by Player  $\sigma$ . Assume then that  $\pi_1 \dots \pi_n$  is not consistent with  $S_v$ . Note that by our choice of  $\pi$  it is consistent with  $S$  and thus, for some  $1 \leq h < n$ , we have that  $S_v(\text{start}(\pi_h)) \neq \text{end}(\pi_h) = S(\text{start}(\pi_h))$ . Let  $k \leq n$  be the smallest index such that  $\text{start}(\pi_k) \in V^{j-1}$ . Such a  $k$  exists by the above inequality and the fact that (by construction)  $S$  agrees with  $S_v$  on all nodes added at round  $j$ . Observe that if  $k > m$  then all the nodes appearing on  $\pi_m \dots \pi_n$  are in  $\text{reach}(S, \pi_k)$ , simply by following the cycle  $\pi_k \dots \pi_n \pi_m \dots \pi_{k-1}$ . Hence, since by item (i) in the induction hypothesis  $\text{reach}(S, \text{start}(\pi_k)) \subseteq V^{j-1}$ , the minimality of  $k$  implies that  $k \leq m$ . We conclude that the suffix  $\pi' = \pi_k \pi_{k+1} \dots$  of  $\pi$ , which is a play consistent with  $S$  starting from  $\text{start}(\pi_k) \in V^{j-1}$ , satisfies  $\text{first}(\pi') = \text{first}(\pi)$ . By item (ii) in the induction hypothesis,  $\pi'$  is won by Player  $\sigma$ , hence so is  $\pi$ , which completes the proof.  $\square$

Theorems 2 and 3 give us the following corollary:

**Corollary 1.** *If a cycle property  $Y$  is closed under cyclic permutations, then every solitaire game in  $\text{FCG}(Y)$  is uniform memoryless determined.*

## 5. Memoryless determinacy of first-cycle games

In this section we give a necessary and sufficient condition for a FCG to be memoryless determined. In Section 7 we give an easy-to check condition that is sufficient, but not necessary.

### 5.1. A full characterisation of memoryless determinacy of all games in $\text{FCG}(Y)$

We begin by introducing some useful shorthand notation. Given an arena  $\mathcal{A} = (V_0, V_1, E, \mathbb{U}, \lambda)$ , and a node  $z \in V$ , for a path  $\pi \in E^* \cup E^\omega$ , define  $N_z(\pi) \in \mathbb{N} \cup \{\infty\}$  to be the index of the first edge that starts with  $z$ , if one exists. Formally,  $N_z(\pi) := \infty$  if  $z$  does not occur on  $\pi$ , and otherwise  $N_z(\pi) := \min\{j : \text{start}(\pi_j) = z\}$ . Also, define  $\text{head}_z(\pi) := \pi[1, N_z(\pi) - 1]$  to be the prefix of  $\pi$  before  $N_z(\pi)$ , and  $\text{tail}_z(\pi) := \pi[N_z(\pi), |\pi|]$  to be the suffix of  $\pi$  starting at  $N_z(\pi)$ . By convention, if  $N_z(\pi) = \infty$  then  $\text{head}_z(\pi) = \pi$  and  $\text{tail}_z(\pi) = \epsilon$ .

We now define a game, that is very similar to the first-cycle game, except that one of the nodes of the arena is designated as a “reset” node:

**Definition 1.** Fix an arena  $\mathcal{A}$ , a vertex  $z \in V$ , and a cycle property  $Y$ . Define the objective  $O_{z\text{-first}}^{\mathcal{A}}(Y) \subseteq \text{plays}(\mathcal{A})$  to consist of all plays  $\pi$  satisfying the following property: if  $\text{head}_z(\pi)$  is not a simple path then  $\text{first}(\pi) \in Y$ , and otherwise  $\text{first}(\text{tail}_z(\pi)) \in Y$ .

Playing the game with objective  $O_{z\text{-first}}(Y)$  is like playing the first-cycle game over  $Y$ , however, if no cycle is formed before reaching  $z$  for the first time, the prefix of the play up to that point is ignored. Thus, in a sense, the game is reset. Also note that if play starts from  $z$ , then the game reduces to a first-cycle game. It turns out that we may assume that a strategy of  $(\mathcal{A}, O_{z\text{-first}}(Y))$  makes the same move every time it reaches  $z$ :

**Definition 2 (Forgetful at  $z$  from  $v$ ).** For an arena  $\mathcal{A}$ , a vertex  $v \in V$ , a Player  $\sigma \in \{0, 1\}$ , and a vertex  $z \in V_\sigma$  belonging to Player  $\sigma$ , we call a strategy  $T$  for Player  $\sigma$  *forgetful at  $z$  from  $v$*  if there exists  $z' \in V$  such that  $(z, z') \in E$  and for all  $\pi \in \text{plays}(T, v)$ , and all  $n \in \mathbb{N}$ , if  $\text{start}(\pi_n) = z$  then  $\text{end}(\pi_n) = z'$ .

**Lemma 5** (*Forgetful at z from v*). Fix an arena  $\mathcal{A}$ , a vertex  $v \in V$ , a Player  $\sigma \in \{0, 1\}$ , and a vertex  $z \in V_\sigma$  belonging to Player  $\sigma$ . In the game  $(\mathcal{A}, O_{z\text{-first}}(Y))$ , if Player  $\sigma$  has a strategy  $S$  that is winning from  $v$ , then Player  $\sigma$  has a strategy  $T$  that is winning from  $v$  and that is forgetful at  $z$  from  $v$ .

**Proof.** We begin with the intuition.

**Sketch.** The second time  $z$  appears on a play, the winner is already determined, and so the strategy is free to repeat the first move it made at  $z$ . On the other hand, when a play visits  $z$  the first time, the strategy can make the same move regardless of the history of the play before  $z$ , because the winning condition ignores this prefix.

**Details.** We may suppose that  $z$  appears on some play of  $\text{plays}(S, v)$ , otherwise we can take  $T$  to be  $S$ . Let  $\rho$  be a simple path from  $v$  to  $z$  that is consistent with  $S$ . Let  $z' := S(\rho)$ . Define the strategy  $T$  as follows:

$$T(u) := \begin{cases} S(u) & \text{if } z \text{ does not appear on } u, \\ z' & \text{if } \text{end}(u) = z, \\ S(\rho \cdot \text{tail}_z(u)) & \text{otherwise.} \end{cases}$$

By definition,  $T$  is forgetful at  $z$  from  $v$ . It remains to show that every  $\pi \in \text{plays}(T, v)$  is won by Player  $\sigma$ .

First consider the case that  $\text{head}_z(\pi)$  is not a simple path. By definition,  $S$  and  $T$  agree on  $\text{head}_z(\pi)$ , and since  $S$  is winning, the first cycle on  $\text{head}_z(\pi)$  (and thus also on  $\pi$ ) satisfies  $Y^\sigma$ , and  $\pi$  is won by Player  $\sigma$ .

Now consider the case that  $\text{head}_z(\pi)$  is a simple path. We need to show that  $\text{first}(\text{tail}_z(\pi))$  is in  $Y^\sigma$ . Define  $\pi' := \rho \cdot \text{tail}_z(\pi)$ . It is easy to see that  $\pi'$  is consistent with  $T$ . We claim that  $\pi' \in \text{plays}(S, v)$ . Indeed, the prefix  $\rho \cdot (z, z')$  is consistent with  $S$ ; and for every  $j \geq |\rho| + 1$ , such that  $\text{end}(\pi'_j) \in V_\sigma$ , we have, by the third case in the definition of  $T$ , that  $T(\pi'[1, j]) = S(\rho \cdot \text{tail}_z(\pi'[1, j])) = S(\pi'[1, j])$  (the second equality holds since, by the definition of  $\text{tail}_z$ ,  $\rho \cdot \text{tail}_z(\pi'[1, j]) = \pi'[1, j]$ ). Now, since  $\pi'$  is consistent with  $T$ , we have that  $T(\pi'[1, j]) = \text{end}(\pi'_{j+1})$ , and thus  $S(\pi'[1, j]) = \text{end}(\pi'_{j+1})$ . This completes the proof of the claim.

To finish the proof, note that  $\text{head}_z(\pi')$  is a simple path (it is  $\rho$ ), and that  $\text{tail}_z(\pi') = \text{tail}_z(\pi)$ . Hence, since  $\text{head}_z(\pi')$  is a simple path, and  $S$  is winning from  $v$ , we know that  $\text{first}(\text{tail}_z(\pi'))$  satisfies  $Y^\sigma$ . Thus, since  $\text{head}_z(\pi)$  is a simple path and  $\text{first}(\text{tail}_z(\pi))$  satisfies  $Y^\sigma$ , we can conclude that  $\pi$  is won by Player  $\sigma$ .  $\square$

We now define the basic notion behind our necessary and sufficient condition for games in  $\text{FCG}(Y)$  to be memoryless determined.

**Definition 3.** For an arena  $\mathcal{A}$ , and a node  $v$  in  $\mathcal{A}$ , say that  $\mathcal{A}$  is  $Y$ -resettable from  $v$  if for every node  $z$ , the same player wins from  $v$  in both  $(\mathcal{A}, O_{\text{first}}(Y))$  and  $(\mathcal{A}, O_{z\text{-first}}(Y))$ .

First, we show that  $Y$ -resetability is a sufficient condition for having memoryless strategies.

**Theorem 4.** Given an arena  $\mathcal{A}$ , if a node  $v$  is such that every sub-arena of  $\mathcal{A}$  is  $Y$ -resettable from  $v$ , then the game  $(\mathcal{A}', O_{\text{first}}(Y))$  is memoryless from  $v$  for every sub-arena  $\mathcal{A}'$  of  $\mathcal{A}$ .

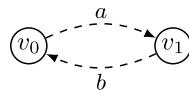
**Proof.** A node  $z \in V$  is a *choice node* of an arena  $\mathcal{B} = (V_0, V_1, E^\mathcal{B}, \mathbb{U}, \lambda)$ , if there are at least two distinct vertices  $v', v'' \in V$  such that  $(z, v') \in E^\mathcal{B}$  and  $(z, v'') \in E^\mathcal{B}$ .

**Sketch.** Fix a sub-arena  $\mathcal{A}'$  of  $\mathcal{A}$ . Suppose Player  $\sigma$  has a winning strategy from  $v$  in  $\mathcal{A}'$ . We induct on the number of choice nodes of Player  $\sigma$ . Let  $z$  be a choice node for Player  $\sigma$  (if there are none, the result is immediate). Since  $\mathcal{A}'$  is  $Y$ -resettable from  $v$ , Player  $\sigma$  also wins the game with objective  $O_{z\text{-first}}(Y)$  from  $v$ . By Lemma 5, Player  $\sigma$  has a strategy  $S$  that is winning from  $v$  and that is also forgetful at  $z$  from  $v$ . Thus we may form a sub-arena  $\mathcal{B}$  of  $\mathcal{A}'$  (and hence of  $\mathcal{A}$ ) by removing all edges from  $z$  that are not taken by  $S$ . Observe that  $S$  is winning from  $v$  in  $(\mathcal{B}, O_{z\text{-first}}(Y))$ . Since the sub-arena  $\mathcal{B}$  is  $Y$ -resettable from  $v$ , Player  $\sigma$  also wins  $(\mathcal{B}, O_{\text{first}}(Y))$  from  $v$ . But  $\mathcal{B}$  has less choice nodes for Player  $\sigma$ , and thus, by induction, Player  $\sigma$  has a memoryless winning strategy from  $v$  in  $(\mathcal{B}, O_{\text{first}}(Y))$ . This memoryless strategy is also winning from  $v$  in  $\mathcal{A}'$ .

**Details.** Fix a sub-arena  $\mathcal{A}'$  of  $\mathcal{A}$ , and let  $\sigma \in \{0, 1\}$  be such that  $v \in \text{WR}^\sigma(\mathcal{A}', O_{\text{first}}(Y))$ . The  $n$ th inductive hypothesis states: for every sub-arena  $\mathcal{B}$  (of  $\mathcal{A}'$ ), in which Player  $\sigma$  has exactly  $n$  choice nodes, if Player  $\sigma$  has a winning strategy from  $v$  in  $(\mathcal{B}, O_{\text{first}}(Y))$ , then Player  $\sigma$  has a *memoryless* winning strategy from  $v$  in  $(\mathcal{B}, O_{\text{first}}(Y))$ .

The base case is when  $n = 0$ , i.e., Player  $\sigma$  has no choice nodes in  $\mathcal{B}$ . In this case, Player  $\sigma$  has a single strategy: given a history  $u \in H_\sigma(\mathcal{B})$  with  $\text{end}(u) = w$ , then take the unique edge  $(w, w') \in E^\mathcal{B}$ . Note that this strategy is memoryless.

Let  $n > 0$ , and suppose the inductive hypothesis holds for  $n - 1$ . We will prove it holds for  $n$ . To this end, let  $\mathcal{B} = (V_0, V_1, E^\mathcal{B}, \mathbb{U}, \lambda)$  be a sub-arena (of  $\mathcal{A}'$ ) with  $n$  choice nodes for Player  $\sigma$ . Fix one such choice node, and call it  $z$ . Suppose Player  $\sigma$  has a winning strategy (not necessarily memoryless) from  $v$  in  $(\mathcal{B}, O_{\text{first}}(Y))$ . Since, by assumption,  $\mathcal{B}$  is  $Y$ -resettable from  $v$ , there is also a winning strategy  $S$  for Player  $\sigma$  from  $v$  in  $(\mathcal{B}, O_{z\text{-first}}(Y))$ . We will use  $S$  to prove Player  $\sigma$  has a memoryless winning strategy from  $v$  in  $(\mathcal{B}, O_{\text{first}}(Y))$ . By Lemma 5, we may assume that  $S$  is forgetful at



**Fig. 5.** The dashed lines represent simple paths.

$z$  from  $v$ , i.e., there exists  $z' \in V$  such that  $(z, z') \in E^{\mathcal{B}}$  and for all  $\pi \in \text{plays}_{\mathcal{B}}(S, v)$ , and all  $i \in \mathbb{N}$ , if  $\text{start}(\pi_i) = z$  then  $\text{end}(\pi_i) = z'$ .

Define the sub-arena  $\mathcal{B}_z$  to be the same as  $\mathcal{B}$  but with all edges out of  $z$  removed except for  $(z, z')$ . That is,  $\mathcal{B}_z := (V_0, V_1, E_z, \mathbb{U}, \lambda')$  where  $E_z := E^{\mathcal{B}} \setminus \{(z, x) : x \neq z'\}$  and  $\lambda'$  is  $\lambda$  restricted to  $E_z$ . Then  $S$  is well-defined on  $\mathcal{B}_z$  (i.e., by our assumption it never says to move from  $z$  to a node other than  $z'$ ). Since  $S$  is winning for Player  $\sigma$  from  $v$  in the game  $(\mathcal{B}, O_{z\text{-first}}(Y))$  and node  $z$  belongs to Player  $\sigma$ , conclude that  $S$  is winning for Player  $\sigma$  from  $v$  in  $(\mathcal{B}_z, O_{z\text{-first}}(Y))$ . Being a sub-arena of  $\mathcal{A}$ , by assumption, the arena  $\mathcal{B}_z$  is  $Y$ -resettable from  $v$ . Hence, Player  $\sigma$  wins from  $v$  in  $(\mathcal{B}_z, O_{\text{first}}(Y))$ . Since  $\mathcal{B}_z$  is a sub-arena of  $\mathcal{A}'$  and has  $n - 1$  choice nodes for Player  $\sigma$ , we can apply the induction hypothesis to  $\mathcal{B}_z$  and obtain that Player  $\sigma$  has a memoryless strategy  $S_{\text{mem}}$  winning from  $v$  in  $(\mathcal{B}_z, O_{\text{first}}(Y))$ . Recall that  $\mathcal{B}_z$  was obtained from  $\mathcal{B}$  by removing outgoing edges from  $z \in V_{\sigma}$  (i.e., by providing Player  $\sigma$  with less freedom of movement), and conclude that  $S_{\text{mem}}$  must be winning for Player  $\sigma$  from  $v$  in the game  $(\mathcal{B}, O_{\text{first}}(Y))$ . This completes the inductive step.  $\square$

It is worth noting that the assumption in [Theorem 4](#), that every sub-arena of  $\mathcal{A}$  is  $Y$ -resettable from  $v$ , cannot be replaced by the weaker requirement that only  $\mathcal{A}$  is  $Y$ -resettable from  $v$ . Consider for example the arena  $\mathcal{A}$  in [Fig. 3](#) (page 10), and a cycle property  $Y \subseteq \mathbb{U}^*$ , such that  $ab \in Y$  but  $ba \notin Y$ , for some  $a, b \in \mathbb{U}^*$ . As argued in [Theorem 3](#), the game  $(\mathcal{A}, O_{\text{first}}(Y))$  is not memoryless starting at  $v_3$ . While the sub-arena  $\mathcal{A}'$ , obtained by dropping the edge  $(v_0, v_1)$ , is not  $Y$ -resettable from  $v_3$  (since for  $z = v_0$  Player 0 wins  $(\mathcal{A}', O_{z\text{-first}}(Y))$  but  $(\mathcal{A}', O_{\text{first}}(Y))$  is won by Player 1) the arena  $\mathcal{A}$  is  $Y$ -resettable from  $v_3$ .<sup>9</sup>

Before we provide a converse for [Theorem 4](#), we show that an additional assumption, namely that  $Y$  is closed under cyclic permutations, is needed:

**Lemma 6.** *If  $Y$  is not closed under cyclic permutations then there is an arena  $\mathcal{A}$ , and a node  $v$ , for which the game  $(\mathcal{A}', O_{\text{first}}(Y))$  is memoryless from  $v$  for every sub-arena  $\mathcal{A}'$  of  $\mathcal{A}$ , but  $\mathcal{A}$  is not  $Y$ -resettable from  $v$ .*

**Proof.** Assume that  $Y$  is not closed under cyclic permutations, and let  $a, b \in \mathbb{U}^*$  be such that  $ab \in Y$  but  $ba \notin Y$ . Consider the arena in [Fig. 5](#), containing only Player 0 nodes, where the path from  $v_0$  to  $v_1$  is labelled by  $a$ , and the path from  $v_1$  to  $v_0$  is labelled by  $b$ . Observe that  $\mathcal{A}$  has only one sub-arena (itself), and there is a single strategy possible in the game  $(\mathcal{A}, O_{\text{first}}(Y))$ , and it is memoryless. However,  $\mathcal{A}$  is not  $Y$ -resettable from  $v_0$  since, starting at  $v_0$ , Player 0 wins the game  $(\mathcal{A}, O_{\text{first}}(Y))$  but not the game  $(\mathcal{A}, O_{z\text{-first}}(Y))$  for  $z = v_1$ .  $\square$

We now prove that if  $Y$  is closed under cyclic permutations then the converse of [Theorem 4](#) is also true.

**Theorem 5.** *Let  $Y$  be closed under cyclic permutations, let  $\mathcal{A}$  be an arena, and let  $v$  be a node such that the game  $(\mathcal{B}, O_{\text{first}}(Y))$  is memoryless from  $v$  for every sub-arena  $\mathcal{B}$  of  $\mathcal{A}$ . Then every sub-arena  $\mathcal{B}$  of  $\mathcal{A}$  is  $Y$ -resettable from  $v$ .*

**Proof.** We prove that, for every arena  $\mathcal{B}$  and a node  $v$  in it, if  $(\mathcal{B}, O_{\text{first}}(Y))$  is memoryless from  $v$  then  $\mathcal{B}$  is  $Y$ -resettable from  $v$ . The theorem follows by taking  $\mathcal{B}$  to be any sub-arena of  $\mathcal{A}$ .

Let  $z$  be a node in  $\mathcal{B}$ . [Lemma 3](#) (Page 6) implies that FCGs are determined. Thus, it is enough to show that if Player  $\sigma$  wins from  $v$  in the game  $(\mathcal{B}, O_{\text{first}}(Y))$ , then he wins from  $v$  in the game  $(\mathcal{B}, O_{z\text{-first}}(Y))$ . So, fix a Player  $\sigma \in \{0, 1\}$  and assume that Player  $\sigma$  wins from  $v$  in  $(\mathcal{B}, O_{\text{first}}(Y))$ . Since by our assumption this game is memoryless determined, there is a memoryless strategy  $S$  for Player  $\sigma$  winning from  $v$  in  $(\mathcal{B}, O_{\text{first}}(Y))$ . Consider the sub-arena  $\mathcal{B}^{\parallel S}$  induced by  $S$ , and recall that every path in  $\mathcal{B}^{\parallel S}$  is consistent with  $S$ . We claim that every simple cycle  $\pi = \pi_1 \dots \pi_k$  (of some length  $k$ ) in  $\mathcal{B}^{\parallel S}$ , that is reachable from  $v$ , satisfies  $Y^{\sigma}$ . Let  $\rho$  be a path in  $\mathcal{B}^{\parallel S}$  of minimal length that starts in  $v$  and ends in  $\text{start}(\pi_i)$  for some  $1 \leq i \leq k$ . Consider the path  $\pi' = \rho \pi_i \dots \pi_k \pi_1 \dots \pi_{i-1}$ . Since  $\pi' \in \text{plays}_{\mathcal{B}}(S, v)$ , and  $S$  is winning from  $v$ , we have that the first cycle  $c$  on  $\pi'$  satisfies  $Y^{\sigma}$ . Observe that (by our choice of  $\rho$ )  $c = \pi_i \dots \pi_k \pi_1 \dots \pi_{i-1}$ , and is thus a cyclic permutation of  $\pi$ . Since  $Y$  is closed under cyclic permutations (recall that if  $Y$  is closed under cyclic permutations then so is  $\neg Y$ ) then  $\pi$  satisfies  $Y^{\sigma}$ , and the claim is true. In other words, for every play  $\rho' \in \text{plays}_{\mathcal{B}}(S, v)$ , and every simple cycle  $c' = \rho'_i \dots \rho'_j$  (for some  $i, j \in \mathbb{N}$ ) on  $\rho'$ , we have that  $c'$  satisfies  $Y^{\sigma}$ . Hence,  $S$  is also winning from  $v$  in the game  $(\mathcal{B}, O_{z\text{-first}}(Y))$ .  $\square$

If an arena  $\mathcal{A}$  is  $Y$ -resettable from  $v$ , for every node  $v$ , then we simply say that it is  $Y$ -resettable. In other words:

<sup>9</sup> One can also come up with such an example (albeit a more complicated one) for  $Y = \text{cyc-EvenLen}$  which is closed under cyclic permutations.

**Definition 4.** An arena  $\mathcal{A}$  is  $Y$ -resettable if for every  $\sigma \in \{0, 1\}$ , and every node  $z$ , we have that  $\text{WR}^\sigma(\mathcal{A}, O_{z\text{-first}}(Y)) = \text{WR}^\sigma(\mathcal{A}, O_{\text{first}}(Y))$ .

We conclude with this full characterisation:

**Theorem 6 (Memoryless determinacy characterisation of FCGs).** *The following are equivalent for every cycle property  $Y$ :*

1.  $Y$  is closed under cyclic permutations, and every arena  $\mathcal{A}$  is  $Y$ -resettable.
2. Every game in  $\text{FCG}(Y)$  is uniform memoryless determined.

**Proof.** Suppose  $Y$  is closed under cyclic permutations. **Theorem 3** (part 3) says that every game in  $\text{FCG}(Y)$  that is point-wise memoryless determined is uniform memoryless determined. But since every arena is assumed  $Y$ -resettable from every  $v$ , **Theorem 4** implies that every game in  $\text{FCG}(Y)$  is point-wise memoryless determined.

Conversely, suppose every game in  $\text{FCG}(Y)$  is uniform memoryless determined. By **Theorem 3** (part 1),  $Y$  must be closed under cyclic permutations. This also means we can apply **Theorem 5**, and conclude that every arena is  $Y$ -resettable.  $\square$

## 6. Strategy Transfer Theorem: infinite-duration cycle games and first-cycle games

In this section we define the connection between first-cycle games and games of infinite duration (such as parity games, etc.), namely the concept of  $Y$ -greedy games. We then prove the Strategy Transfer Theorem, which says, roughly, that for every arena, the winning regions of the First-Cycle Game over  $Y$  and a  $Y$ -greedy game coincide, and that memoryless winning strategies transfer between these two games.

**Definition 5 (Greedy).** Say that a game  $(\mathcal{A}, O)$  is  $Y$ -greedy if

$$O_{\text{all}}^{\mathcal{A}}(Y) \subseteq O \text{ and } O_{\text{all}}^{\mathcal{A}}(\neg Y) \subseteq E^\omega \setminus O.$$

Intuitively, a game  $(\mathcal{A}, O)$  is  $Y$ -greedy means that Player 0 can win the game  $(\mathcal{A}, O)$  if he ensures that every cycle in the cycles-decomposition of the play is in  $Y$ , and Player 1 can win if she ensures that every cycle in the cycles-decomposition of the play is not in  $Y$ .

An equivalent formulation is that  $(\mathcal{A}, O)$  is  $Y$ -greedy if

$$O_{\text{all}}^{\mathcal{A}}(Y) \subseteq O \subseteq O_{\text{exist}}^{\mathcal{A}}(Y),$$

where  $O_{\text{exist}}^{\mathcal{A}}(Y)$  consists of all plays  $\pi$  such that some cycle in  $\text{cycles}(\pi)$  satisfies  $Y$ .

Here are some examples.

1. Every all-cycles game  $(\mathcal{A}, O_{\text{all}}(Y))$  is  $Y$ -greedy.
2. Every parity game is cyc-Parity-greedy.
3. Every game with  $v$ -mean-payoff winning condition is cyc-MeanPayoff $_v$ -greedy.

Before proving the Strategy Transfer Theorem we need a lemma that states that one can pump a strategy  $S$  that is winning for the first-cycle game to get a strategy  $S^\circlearrowright$  that is winning for the all-cycles game by following  $S$  until a cycle is formed, removing that cycle from the history, and continuing. The fact that every winning strategy in the first-cycle game of  $Y$  can be pumped to obtain a winning strategy in a  $Y$ -greedy game, is why we call such games “greedy”.

Recall from the Definitions that for a finite path  $\pi \in E^*$ , the stack content at the end of  $\text{CycDEC}(\epsilon, \pi)$  (**Algorithm 1**) is denoted  $\text{stack}(\pi)$ . Recall our notational abuse (Page 4) that for a finite path  $\rho$ , we may write  $S(\rho)$  instead of  $S(\text{nodes}(\rho))$ .

**Definition 6 (Pumping strategy).** Fix an arena  $\mathcal{A}$ , a Player  $\sigma \in \{0, 1\}$ , and a strategy  $S$  for Player  $\sigma$ . Let the pumping strategy of  $S$  be the strategy  $S^\circlearrowright$  for Player  $\sigma$ , defined, on any input  $u = v_1 \dots v_k \in H_\sigma(\mathcal{A})$ , as follows:

- a.  $S^\circlearrowright(u) := S(v_k)$  if  $k = 1$  or  $\text{stack}(\pi) = \epsilon$ , and otherwise
- b.  $S^\circlearrowright(u) = S(\text{stack}(\pi))$ ,

where  $\pi \in E^*$  is the path corresponding to  $u$ , i.e.,  $\text{nodes}(\pi) = u$ .

Note that  $S^\circlearrowright$  is well-defined since if  $\text{stack}(\pi) \neq \epsilon$  then  $\text{stack}(\pi)$  ends with  $v_k \in V_\sigma$  and so  $\text{stack}(\pi)$  is in the domain of  $S$ . Also note that if  $S$  is memoryless then the pumping strategy  $S^\circlearrowright = S$ .

**Lemma 7.** Let  $\mathcal{A}, \sigma, S$  and  $S^\circlearrowright$  be as in **Definition 6**, and let  $v \in V$ . If  $\pi \in \text{plays}(S^\circlearrowright, v)$  then for every cycle  $C$  in  $\text{cycles}(\pi)$  there exists a finite path  $\rho$  consistent with  $S$  and starting with  $v$  such that:

- The first cycle on  $\rho$  is  $C$  (thus, in particular, if  $S$  is winning from  $v$  in the game  $(\mathcal{A}, O_{\text{first}}(Y))$  then  $C$  satisfies  $Y^\sigma$ );
- $\rho$  only contains edges from  $\pi$ .

**Proof. Sketch.** The strategy  $S^\circlearrowleft$  says to follow  $S$ , and when a cycle is popped by CycDEC, remove that cycle from the history and continue. Thus, for every cycle  $C$  that is popped, let  $l$  be the time at which the first edge of  $C$  is being pushed, and note that the stack up to time  $l$  followed by  $C$  is a path consistent with  $S$  whose first cycle is  $C$ .

**Details.** Fix  $\pi \in \text{plays}(S^\circlearrowleft, v)$ . Every cycle  $C \in \text{cycles}(\pi)$  was output by the run of CycDEC( $\epsilon, \pi$ ) at some time  $j \in \mathbb{N}$ , and is thus of the form  $C = \text{cycle}^j(\pi)$ . Let  $\rho := \text{stack}_{j-1}(\pi) \cdot \pi_j$ , and observe that  $C$  is the first cycle on  $\rho$ , and the second item in the statement of the lemma is true.

For the first item, it is sufficient to prove, for all  $k \in \mathbb{N}$ , that  $\text{stack}_{k-1}(\pi) \cdot \pi_k$  is consistent with  $S$  and starts with  $v$ . We remind the reader that, by definition,  $\rho$  is consistent with  $S$  iff:

1.  $S(\text{start}(\rho_1)) = \text{end}(\rho_1)$  if  $\text{start}(\rho_1) \in V_\sigma$ , and
2.  $S(\rho[1, n]) = \text{end}(\rho_{n+1})$  for  $1 \leq n < |\rho|$  with  $\text{end}(\rho[1, n]) \in V_\sigma$ .

We prove that  $\text{stack}_{k-1}(\pi) \cdot \pi_k$  is consistent with  $S$  and starts with  $v$ , by induction on  $k \in \mathbb{N}$ .

The base is when  $k = 1$ . Since  $\text{stack}_0(\pi) = \epsilon$ , we show that the path consisting of the single edge  $\pi_1$ , which starts with  $v$  by definition, is consistent with  $S$ . Note that we are in item 1. of the consistency condition with  $\rho = \pi_1$ . So suppose  $\text{start}(\pi_1) \in V_\sigma$ . Then:

$$\begin{aligned} \text{end}(\pi_1) &= S^\circlearrowleft(\text{start}(\pi_1)) && \text{since } \pi \text{ is consistent with } S^\circlearrowleft, \\ &= S(\text{start}(\pi_1)) && \text{by definition of } S^\circlearrowleft, \text{ part a).} \end{aligned}$$

For the inductive step, let  $k > 1$  and suppose the inductive hypothesis holds for  $k - 1$ . We prove it holds for  $k$ . There are two cases.

i) Suppose  $\text{stack}_{k-1}(\pi) = \epsilon$ . To show  $\pi_k$  is consistent with  $S$ , note that we are again in item 1. of the consistency condition, but this time with  $\rho = \pi_k$ . Repeat the argument in the base case with  $\pi_k$  replacing  $\pi_1$ . To show that  $\pi_k$  starts with  $v$  argue as follows. The fact that  $\text{stack}_{k-1}(\pi) = \epsilon$  means that after pushing  $\pi_{k-1}$  onto the stack during step  $k - 1$  of the algorithm CycDEC( $\epsilon, \pi$ ), the resulting stack forms a cycle. In other words,  $\text{end}(\pi_{k-1}) = \text{start}(\text{stack}_{k-2}(\pi) \cdot \pi_{k-1})$ . But  $\text{start}(\pi_k) = \text{end}(\pi_{k-1})$  since  $\pi$  is a path, and  $\text{start}(\text{stack}_{k-2}(\pi) \cdot \pi_{k-1}) = v$  by the induction hypothesis. Thus  $\text{start}(\pi_k) = v$ .

ii) Suppose  $\text{stack}_{k-1}(\pi) \neq \epsilon$ . The induction hypothesis states that the path  $\text{stack}_{k-2}(\pi) \cdot \pi_{k-1}$  is consistent with  $S$  and starts with  $v$ . Observe that  $\text{stack}_{k-1}(\pi)$  is a (non-empty) prefix of  $\text{stack}_{k-2}(\pi) \cdot \pi_{k-1}$ . So,  $\text{stack}_{k-1}(\pi)$  starts with  $v$ , and, being a prefix of a path consistent with  $S$ , is consistent with  $S$ . Thus we only need to consider  $\pi_k$ . That is, we are in item 2. of the consistency condition, with  $\rho = \text{stack}_{k-1}(\pi) \cdot \pi_k$  and  $n = |\text{stack}_{k-1}(\pi)|$ . If  $\text{end}(\rho[1, n]) \in V_\sigma$  then

$$\begin{aligned} \text{end}(\rho_{n+1}) &= \text{end}(\pi_k) && \text{since } \rho_{n+1} = \pi_k, \\ &= S^\circlearrowleft(\pi[1, k-1]) && \text{since } \pi \text{ is consistent with } S^\circlearrowleft, \\ &= S(\text{stack}_{k-1}(\pi)) && \text{by definition of } S^\circlearrowleft, \text{ part b),} \\ &= S(\rho[1, n]) && \text{since } \rho[1, n] = \text{stack}_{k-1}(\pi). \end{aligned}$$

This completes the inductive step and the proof.  $\square$

**Corollary 2.** Fix Player  $\sigma \in \{0, 1\}$  and let  $(\mathcal{A}, O)$  be a  $Y$ -greedy game. If  $S$  is a strategy for Player  $\sigma$  that is winning from  $v$  in  $(\mathcal{A}, O_{\text{first}}(Y))$  then  $S^\circlearrowleft$  is winning from  $v$  in  $(\mathcal{A}, O)$ .

**Proof.** Suppose  $S$  is winning from  $v$  in the game  $(\mathcal{A}, O_{\text{first}}(Y))$ . Then, by Lemma 7, for every play  $\pi \in \text{plays}(S^\circlearrowleft, v)$ , every cycle in  $\text{cycles}(\pi)$  satisfies  $Y^\sigma$ , i.e.,  $\pi \in O_{\text{all}}^{\mathcal{A}}(Y^\sigma)$ . By definition of  $Y$ -greedy, this means that  $S^\circlearrowleft$  is winning from  $v$  in the game  $(\mathcal{A}, O)$ .  $\square$

We now have the ingredients for the proof of the Strategy Transfer Theorem:

**Theorem 7 (Strategy transfer).** Let  $(\mathcal{A}, O)$  be a  $Y$ -greedy game, and let  $\sigma \in \{0, 1\}$ .

1. The winning regions for Player  $\sigma$  in the games  $(\mathcal{A}, O)$  and  $(\mathcal{A}, O_{\text{first}}(Y))$  coincide.
2. For every memoryless strategy  $S$  for Player  $\sigma$ , and vertex  $v \in V$  in arena  $\mathcal{A}$ :  $S$  is winning from  $v$  in the game  $(\mathcal{A}, O)$  if and only if  $S$  is winning from  $v$  in the game  $(\mathcal{A}, O_{\text{first}}(Y))$ .

**Proof.** Let  $Y \subseteq \mathbb{U}^*$  be a cycle property and  $\mathcal{A}$  an arena. Suppose that  $(\mathcal{A}, O)$  is  $Y$ -greedy. We begin by proving the first item. Use Corollary 2 to get that for  $\sigma \in \{0, 1\}$ ,

$$\text{WR}^\sigma(\mathcal{A}, O_{\text{first}}(Y)) \subseteq \text{WR}^\sigma(\mathcal{A}, O).$$

Since first-cycle games are determined (Lemma 3), the winning regions  $\text{WR}^0(\mathcal{A}, O_{\text{first}}(Y))$  and  $\text{WR}^1(\mathcal{A}, O_{\text{first}}(Y))$  partition  $V$ . Thus, since  $\text{WR}^0(\mathcal{A}, O)$  and  $\text{WR}^1(\mathcal{A}, O)$  are disjoint, the containments above are equalities, as required for item 1.

We prove the second item. Since  $S = S^\circlearrowleft$  if  $S$  is memoryless, conclude by Corollary 2: if  $S$  is winning from  $v$  in the game  $(\mathcal{A}, O_{\text{first}}(Y))$  then it is winning from  $v$  in the game  $(\mathcal{A}, O)$ . For the other direction, assume by contraposition that  $S$  is not winning from  $v$  in the game  $(\mathcal{A}, O_{\text{first}}(Y))$ . Since  $S$  is memoryless, plays of  $\mathcal{A}$  consistent with  $S$  are exactly infinite paths in the induced sub-arena  $\mathcal{A}^{\parallel S}$ . Hence, there is a path  $\pi$  in the induced solitaire arena  $\mathcal{A}^{\parallel S}$  for which the first cycle, say  $\pi[i, j]$ , satisfies  $Y^{1-\sigma}$ . Define the infinite path  $\pi' := \pi[1, i-1] \cdot (\pi[i, j])^\omega$  and note that, being a path in  $\mathcal{A}^{\parallel S}$ , it is a play of  $\mathcal{A}$  consistent with  $S$ . Moreover,  $\pi'$  has the property that every cycle in its cycles-decomposition (i.e.,  $\pi[i, j]$ ) satisfies  $Y^{1-\sigma}$ . Since  $(\mathcal{A}, O)$  is  $Y$ -greedy,  $S$  is not winning from  $v$  in the game  $(\mathcal{A}, O)$ .  $\square$

Since FCGs are determined (Lemma 3, Page 6) use Theorem 7 to get:

**Corollary 3.** Every  $Y$ -greedy game  $(\mathcal{A}, O)$  is determined, has the same winning regions as  $(\mathcal{A}, O_{\text{first}}(Y))$ , and is point-wise (uniform) memoryless determined if and only if the game  $(\mathcal{A}, O_{\text{first}}(Y))$  is point-wise (uniform) memoryless determined.

We now state our main result concerning  $Y$ -greedy games.

**Theorem 8** (Memoryless determinacy characterisation of greedy games). For every cycle property  $Y$ , the following are equivalent:

1.  $Y$  is closed under cyclic permutations, and every arena is  $Y$ -resettable.
2. Every  $Y$ -greedy game is uniform memoryless determined.

**Proof.** This follows from Corollary 3, Theorem 6, and the fact that for every arena  $\mathcal{A}$  there is a  $Y$ -greedy game with arena  $\mathcal{A}$ , for example, the game  $(\mathcal{A}, O_{\text{all}}(Y))$ .  $\square$

## 7. An easy to check sufficient condition on $Y$ ensuring memoryless determinacy of FCGs

As we have seen  $Y$ -resetability together with closure under cyclic permutations provides a full characterization of those cycle properties  $Y$  for which the games in  $\text{FCG}(Y)$ , as well as any  $Y$ -greedy games, are memoryless determined. Even though, in many cases, checking whether  $Y$  is such that every arena  $\mathcal{A}$  (or just the arena(s) of interest) is  $Y$ -resettable is not too difficult, we believe that in practice it is much easier to use the following condition on  $Y$  which avoids reasoning about arenas altogether. The condition we suggest is the following:

$$Y \text{ and } \neg Y \text{ are closed under concatenation.}$$

As we later show, this condition (together with closure under cyclic permutations) ensures that every arena  $\mathcal{A}$  is  $Y$ -resettable, and thus by Theorem 6, that all games in  $\text{FCG}(Y)$  are memoryless determined. On the other hand, as we discuss in Section 8, there is a cycle property  $Y$  which is closed under cyclic permutations and for which every arena  $\mathcal{A}$  is  $Y$ -resettable, even though  $Y$  does not satisfy the condition. Thus, this condition cannot replace  $Y$ -resetability as a necessary condition for memoryless determinacy of all games in  $\text{FCG}(Y)$ . However, it is applicable in a wide variety of cases, and is usually very easy to check. Consider for example the cycle properties given in Section 2. For each of these properties  $Y$ , while proving that every arena is  $Y$ -resettable may not be hard, checking whether  $Y$  satisfies the condition above is almost completely trivial. Note that cyc-EvenLen, which is the only property among these which is closed under cyclic permutations but fails to satisfy this condition, would also fail to satisfy any other condition that guarantees memoryless determinacy since it actually admits FCGs that are not memoryless determined (cf. Theorem 3, Page 10).

Our goal in the rest of this section is to prove the following theorem:

**Theorem 9** (Easy to check). Let  $Y \subseteq \mathbb{U}^*$  be a cycle property. If  $Y$  is closed under cyclic permutations, and both  $Y$  and  $\neg Y$  are closed under concatenation, then every arena  $\mathcal{A}$  is  $Y$ -resettable.

By Theorem 6 we get:

**Corollary 4.** Let  $Y \subseteq \mathbb{U}^*$  be a cycle property. If  $Y$  is closed under cyclic permutations, and both  $Y$  and  $\neg Y$  are closed under concatenation, then every game in  $\text{FCG}(Y)$  is uniform memoryless determined.

**Remark 2.** Consider the cycle property  $Y = \text{cyc-GoodForEnergy}$  (recall from the definitions this means that either the energy level is positive, or it is zero and the largest priority occurring is even). Note that  $Y$  is closed under cyclic permutations, and both  $Y$  and  $\neg Y$  are closed under concatenation. Conclude that every game in  $\text{FCG}(Y)$  is pointwise-memoryless determined. This fact allows one to obtain a proof of Lemma 4 in [5] that no longer relies on the incorrect result from [4].

We begin by introducing a new kind of objective  $O_{\text{tail}}^{\mathcal{A}}(Y)$ :

**Definition 7.** For an arena  $\mathcal{A}$  and cycle property  $Y$ , define the objective  $O_{\text{tail}}^{\mathcal{A}}(Y)$  to consist of all plays  $\pi$  such that there is a suffix  $\pi'$  of  $\pi$  with the property that every cycle in  $\text{cycles}(\pi')$  satisfies  $Y$ .

Note that this is not the same as saying that  $\lambda(C) \in Y$  for all but finitely many cycles  $C$  in  $\text{cycles}(\pi)$ .<sup>10</sup>

**Definition 8.** An arena  $\mathcal{A}$  is  $Y$ -unambiguous if  $O_{\text{tail}}^{\mathcal{A}}(Y) \cap O_{\text{tail}}^{\mathcal{A}}(\neg Y) = \emptyset$ .

In other words,  $\mathcal{A}$  is  $Y$ -unambiguous if no play in  $\mathcal{A}$  has two suffixes,  $\pi, \pi'$  such that every cycle in the cyclic decomposition of  $\pi$  is in  $Y$ , and every cycle in the cyclic decomposition of  $\pi'$  is not in  $Y$ .

In order to prove [Theorem 9](#) we have to show that, for  $Y$  satisfying the assumptions of the theorem, for every arena  $\mathcal{A}$ , every Player  $\sigma \in \{0, 1\}$ , and every node  $z$  in  $\mathcal{A}$ , we have that  $\text{WR}^{\sigma}(\mathcal{A}, O_{\text{first}}(Y)) = \text{WR}^{\sigma}(\mathcal{A}, O_{z\text{-first}}(Y))$ . The proof is in three parts:

- Part 1. If  $Y$  is closed under cyclic permutations, and both  $Y$  and  $\neg Y$  are closed under concatenation, then every arena  $\mathcal{A}$  is  $Y$ -unambiguous.
- Part 2. If  $\mathcal{A}$  is  $Y$ -unambiguous then  $\text{WR}^{\sigma}(\mathcal{A}, O_{\text{first}}(Y)) = \text{WR}^{\sigma}(\mathcal{A}, O_{\text{tail}}(Y))$ .
- Part 3. If  $\mathcal{A}$  is  $Y$ -unambiguous then  $\text{WR}^{\sigma}(\mathcal{A}, O_{z\text{-first}}(Y)) = \text{WR}^{\sigma}(\mathcal{A}, O_{\text{tail}}(Y))$ .

Observe that parts 2 and 3 imply that if an arena  $\mathcal{A}$  is  $Y$ -unambiguous then it is  $Y$ -resettable.

We first need a couple of definitions and a few easy lemmas. Say that  $Y$  is *closed under insertions* if it is closed under concatenation and:  $ac \in Y$  and  $b \in Y$  imply that  $abc \in Y$ , for all  $a, b, c \in \mathbb{U}^*$ . The *closure of  $Y$  under insertions*, is defined to be the smallest subset of  $\mathbb{U}^*$  (with respect to set containment) that contains  $Y$  and is closed under insertions.

**Lemma 8.** *If  $Y$  is closed under cyclic permutations and under concatenation then  $Y$  is closed under insertions.*

**Proof.** Assume that  $ac, b \in Y$ . We have that  $ac \in Y \implies ca \in Y \implies cab \in Y \implies abc \in Y$ . The middle implication is since  $Y$  is closed under concatenation, and the other two implications are since  $Y$  is closed under cyclic permutations.  $\square$

Fix an arena  $\mathcal{A} = (V_0, V_1, E, \mathbb{U}, \lambda)$ . Given a simple path  $s \in E^*$ , and a path  $u \in E^*$  such that  $\text{end}(s) = \text{start}(u)$ , write  $\text{labels}(s, u) = \{\lambda(C) \mid C \in \text{cycles}(s, u)\}$  for the set of the labels of the cycles output by  $\text{CycDEC}(s, u)$ .

**Lemma 9.** *Given an arena  $\mathcal{A} = (V_0, V_1, E, \mathbb{U}, \lambda)$ , let  $s, u$  be paths in  $\mathcal{A}$  such that  $u$  is a cycle and  $\text{end}(s) = \text{start}(u) = v$ . We have that:*

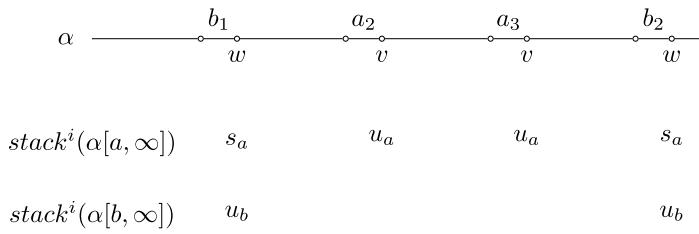
1. *if  $\text{stack}(s, u) = s$  then the insertion closure of  $\text{labels}(s, u)$  contains  $\lambda(u)$ ;*
2. *if  $v$  is the only node that appears on both  $s$  and  $u$  then  $\text{stack}(s, u) = s$ .*

**Proof. Sketch.** For the first item, note that the assumption  $\text{stack}(s, u) = s$  implies that the cycles in  $\text{cycles}(s, u)$  contain exactly the edges of  $u$ . Hence, it is not hard to see that  $\lambda(u)$  is in the insertion closure of  $\text{labels}(s, u)$ . Intuitively, the algorithm output the cycles in  $\text{cycles}(s, u)$  by “popping them out” of  $u$ . Thus, we can reverse this process and reconstruct  $u$  using insertion operations on the elements of  $\text{cycles}(s, u)$ .

**Details.** More formally, let  $C^1, \dots, C^m$  be the elements of  $\text{cycles}(s, u)$  in the order they were output. We iteratively construct a string  $w \in E^*$  until we get  $w = u$ . We start with  $w$  being the empty string, and count down from  $m$  to 1. At step  $i$  in the count, we insert the cycle  $C^i$  into the correct position in  $w$ , as follows: let  $j$  be such that  $u_j = C_0^i$  (i.e., the first edge of  $C^i$ ), and set  $w := w_1 \dots w_h \cdot C^i \cdot w_{h+1} \dots w_{|w|}$ , where  $h$  is the maximal index such that all the edges  $w_1, \dots, w_h$  are in  $\{u_1, \dots, u_{j-1}\}$ .

It is easy to see that when the construction ends  $w$  contains exactly the edges appearing in  $C^1, \dots, C^m$  and thus, as noted before, exactly the edges of  $u$ . We claim that the edges are also ordered correctly (i.e., if  $a < b$  then  $w_a$  appears in  $u$  before  $w_b$ ) and thus,  $w = u$  as required. Assume by way of contradiction that the correct ordering in  $w$  was violated for the first time when  $C^i$  was inserted. Let  $u_j$  be the first edge of  $C^i$  and  $h$  be the position where  $C^i$  was inserted into  $w$ , as defined before. Recall that after the insertion the constructed string is  $w_1 \dots w_h \cdot C^i \cdot w_{h+1} \dots w_{|w|}$ . Observe that since all edges of  $C_i$  are internally ordered correctly, and all of them correctly appear (by our choice of  $h$ ) after  $w_1 \dots w_h$ , the only possible violation is that some edge  $u_t$  in  $C^i$  incorrectly appears before the edge  $w_{h+1} = u_r$ , i.e., that  $j < r < t$ . Also note that at the step where  $C^i$  was output, all edges above  $u_j$  that were on the stack were popped, and all edges up to  $u_t$  were already processed. Hence, it can not be that the edge  $u_r$  was output by the algorithm after  $C^i$ , which is a contradiction to the fact that  $u_r$  is already in  $w$  before the insertion of  $C^i$ . This completes the proof of the claim.

<sup>10</sup> For instance, consider the arena in [Fig. 4](#), take  $Y$  to be cyc-EvenLen, and let  $\pi := [(v_1, v_2)(v_2, v_1)(v_1, v_3)(v_3, v_2)(v_2, v_4)(v_4, v_1)]^\omega$ . Note that i) decomposing the suffix  $\pi'$  starting with the second edge results in all cycles having odd length, and ii) it is not the case that almost all cycles in the cycles-decomposition of  $\pi$  (i.e., starting with the first edge) have odd length – in fact, they all have even length.

**Fig. 6.** Visualisation of the processing of the play  $\alpha$ .

For the second item, observe that the assumption made there implies that, while processing  $u$ , the algorithm CycDec( $s, u$ ) cannot pop any edge in  $s$ . Hence,  $stack(s, u) = s$  does not hold only if there exists  $j$  such that the edge  $u_j = (v, v')$  is pushed on top of  $s$  but never popped. Observe that  $v' \neq v = src(u) = trg(u)$  (the first inequality is since a self-loop is always popped, the rest by our assumption that  $u$  is a cycle that starts in  $v$ ), and thus  $u_j$  is not the last edge of  $u$ . But this is a contradiction since the (non-empty) suffix  $u_{j+1} \dots u_{|u|}$  of  $u$  contains at least one edge  $e'$  with  $end(e') = v$  (namely  $u_{|u|}$ ), and the algorithm would have popped the edge  $u_j$  when it encountered the first edge that ends in  $v$ .  $\square$

We are now ready to show part 1 in the proof of [Theorem 9](#).

**Proposition 2.** Let  $Y \subseteq \mathbb{U}^*$  be a cycle property. If  $Y$  is closed under cyclic permutations, and both  $Y$  and  $\neg Y$  are closed under concatenation, then every arena  $\mathcal{A}$  is  $Y$ -unambiguous.

**Proof.** First, note that since  $Y$  is closed under cyclic permutations then so is  $\neg Y$ . By [Lemma 8](#), we have that  $Y$ , as well as  $\neg Y$ , are closed under insertions.

Assume by way of contradiction that there is an arena  $\mathcal{A} = (V_0, V_1, E, \mathbb{U}, \lambda)$  which is not  $Y$ -unambiguous, and take a play  $\alpha$  in the intersection of  $O_{tail}^{\mathcal{A}}(Y)$  and  $O_{tail}^{\mathcal{A}}(\neg Y)$ . Let  $a, b \in \mathbb{N}$  be such that every cycle in  $cycles(\alpha[a, \infty])$  satisfies  $Y$ , and every cycle in  $cycles(\alpha[b, \infty])$  satisfies  $\neg Y$ .

For  $i, j$  such that  $a \leq i < j$ , write  $Out(i, j) = \{cycle^l(\alpha[a, \infty]) \mid i \leq l \leq j\} \setminus \{\epsilon\}$  to be the set of cycles output by CycDec( $\epsilon, \alpha[a, \infty]$ ) while processing the edges  $\alpha_i \dots \alpha_j$ , and note that for every  $C \in Out(i, j)$  we have that  $\lambda(C) \in Y$ . Similarly, for  $i, j$  such that  $b \leq i < j$ , write  $Out^-(i, j) = \{cycle^l(\alpha[b, \infty]) \mid i \leq l \leq j\} \setminus \{\epsilon\}$  to be the set of cycles output by CycDec( $\epsilon, \alpha[b, \infty]$ ) while processing the edges  $\alpha_i \dots \alpha_j$ , and note that for every  $C \in Out^-(i, j)$  we have that  $\lambda(C) \in \neg Y$ .

Since, for every  $i \geq 1$ , the stack content  $stack^i(\alpha[a, \infty])$  is of length at most  $|V| - 1$ , there is at least one stack content that appears infinitely often. Thus, let  $u_a \in E^*$  be a path of minimal length such that the set  $A = \{i \in \mathbb{N} \mid u_a = stack^i(\alpha[a, \infty])\}$  is infinite. Similarly, let  $u_b \in E^*$  be a path of minimal length such that  $B = \{i \in \mathbb{N} \mid u_b = stack^i(\alpha[b, \infty])\}$  is infinite.

We assume w.l.o.g. (otherwise we simply replace  $A$  and  $B$  by appropriate infinite subsets of themselves until each of the following conditions is satisfied) that  $\dagger$ :

- (i)  $u_a = stack^i(\alpha[a, \infty])$  for every  $i \in A$ , and  $u_b = stack^i(\alpha[b, \infty])$  for every  $i \in B$ ;
- (ii)  $end(\alpha_i) = end(\alpha_j)$  for all  $i, j \in A$  and  $end(\alpha_i) = end(\alpha_j)$  for all  $i, j \in B$  (recall that the nodes on  $\alpha$  come from the finite set  $V$ );
- (iii) there exists  $s_a \in E^*$  such that  $s_a = stack^i(\alpha[a, \infty])$  for all  $i \in B$ .

Pick indices  $b_1, b_2 \in B$  and  $a_2, a_3 \in A$  in such a way that  $b_1 < a_2 < a_3 < b_2$ . [Fig. 6](#) may aid in visualising the current state of affairs. In this figure,  $w := end(\alpha_{b_1}) = end(u_b) = end(\alpha_{b_2})$  and  $v := end(\alpha_{a_2}) = end(u_a) = end(\alpha_{a_3})$ .

Our aim now is to show that the above state of affairs leads to a contradiction, and thus deduce that it can not be that  $\alpha \in O_{tail}^{\mathcal{A}}(Y) \cap O_{tail}^{\mathcal{A}}(\neg Y)$ . The contradiction we will show is that  $\lambda(\alpha[b_1 + 1, b_2])$  is both in  $Y$  and in its complement  $\neg Y$ , which is impossible. We will do that by showing how to build  $\lambda(\alpha[b_1 + 1, b_2])$  in two different ways: the first by using cyclic permutations and concatenations of strings that are the labels of cycles that satisfy  $Y$ , and the second by doing the same but with cycles that satisfy  $\neg Y$ . Since by the assumption of the proposition  $Y$  and  $\neg Y$  are closed under these string operations, the contradiction is reached.

We first show that  $\alpha[b_1 + 1, b_2]$  (which by  $\dagger(ii)$  is a cycle) satisfies  $\neg Y$ . Intuitively (refer to [Fig. 6](#)), this follows from the fact that all cycles output by the algorithm CycDec( $\epsilon, \alpha[b, \infty]$ ) while processing  $\alpha[b_1 + 1, b_2]$  satisfy  $\neg Y$ , and the fact that before and after processing  $\alpha[b_1 + 1, b_2]$  the stack of this algorithm is  $u_b$ , and thus by [Lemma 9](#) we have that  $\lambda(\alpha[b_1 + 1, b_2])$  is in the insertion closure of the labels of these cycles, and thus also in  $\neg Y$  (recall that  $\neg Y$  is closed under insertions).

We next show that  $\lambda(\alpha[b_1 + 1, b_2]) \in Y$ . Observe (refer to [Fig. 6](#)) that  $\alpha[b_1 + 1, b_2] = \alpha[b_1 + 1, a_2]\alpha[a_2 + 1, a_3]\alpha[a_3 + 1, b_2]$ . Since  $Y$  is closed under concatenation and cyclic permutations, to show that  $\lambda(\alpha[b_1 + 1, b_2]) \in Y$  it is enough to show that  $\lambda(\alpha[a_2 + 1, a_3]) \in Y$  and  $\lambda(x) \in Y$ , where  $x := \alpha[a_3 + 1, b_2]\alpha[b_1 + 1, a_2]$ . The fact that  $\lambda(\alpha[a_2 + 1, a_3]) \in Y$  follows from a symmetric argument that mimics the one used above to prove that  $\alpha[b_1 + 1, b_2]$  satisfies  $\neg Y$ . It remains to show that  $\lambda(x) \in Y$ .

To see that  $\lambda(x) \in Y$ , note that when the algorithm  $\text{CycDEC}(\epsilon, \alpha[a, \infty])$  finishes processing  $\alpha[a_3 + 1, b_2]$  it has the same stack content (namely  $s_a$ ) that it had when it previously started processing  $\alpha[b_1 + 1, a_2]$ . Thus, if we imagine that after processing  $\alpha[a_3 + 1, b_2]$ , instead of continuing to process  $\alpha[b_2 + 1, \infty]$ , we feed the algorithm again with  $\alpha[b_1 + 1, a_2]$  (i.e., we let it process the second part of  $x$ ), we will get (by Lemma 2) that it will behave exactly the same as when it first processed  $\alpha[b_1 + 1, a_2]$  as part of the prefix  $\alpha[a, a_2]$ . We thus obtain that while processing  $x$  this way starting with stack  $u_a$ , the algorithm ends with stack  $u_a$  and outputs only cycles that satisfy  $Y$  (recall that all cycles output by  $\text{CycDEC}(\epsilon, \alpha[a, \infty])$  do). Thus, by Lemma 9, we have that  $\lambda(x)$  is in the insertion closure of the labels of cycles that satisfy  $Y$ , and thus also in  $Y$  (recall that  $Y$  is closed under insertions).  $\square$

The following Proposition deals with Part 2 in the proof of Theorem 9.

**Proposition 3.** Fix a  $Y$ -unambiguous arena  $\mathcal{A}$ . Then for  $\sigma \in \{0, 1\}$ ,

$$\text{WR}^\sigma(\mathcal{A}, O_{\text{first}}(Y)) = \text{WR}^\sigma(\mathcal{A}, O_{\text{tail}}(Y)).$$

**Proof.** By Theorem 7 (item 1) it is sufficient to show that if  $\mathcal{A}$  is  $Y$ -unambiguous then the game  $(\mathcal{A}, O_{\text{tail}}(Y))$  is  $Y$ -greedy. So, fix a play  $\pi$  in  $\mathcal{A}$ . If every cycle in the cycles-decomposition of  $\pi$  satisfies  $Y$  then certainly  $\pi \in O_{\text{tail}}(Y)$  (just take  $\pi$  itself as the required suffix). On the other hand, if every cycle in the cycles-decomposition of  $\pi$  satisfies  $\neg Y$  then for the same reason  $\pi \in O_{\text{tail}}(\neg Y)$ . However, since  $\mathcal{A}$  is  $Y$ -unambiguous,  $\pi \notin O_{\text{tail}}(Y)$ , as required.  $\square$

The following Proposition deals with Part 3.

**Proposition 4.** Fix a  $Y$ -unambiguous arena  $\mathcal{A}$  and vertex  $z \in V$ . Then for  $\sigma \in \{0, 1\}$ ,

$$\text{WR}^\sigma(\mathcal{A}, O_{z\text{-first}}(Y)) = \text{WR}^\sigma(\mathcal{A}, O_{\text{tail}}(Y)).$$

**Proof.** Let  $\mathcal{A} = (V_0, V_1, E, \mathbb{U}, \lambda)$ . We first claim that for every  $\sigma \in \{0, 1\}$ , we have that  $\text{WR}^\sigma(\mathcal{A}, O_{z\text{-first}}(Y)) \subseteq \text{WR}^\sigma(\mathcal{A}, O_{\text{tail}}(Y))$ . To see that the Proposition follows from this claim note that  $\text{WR}^0(\mathcal{A}, O_{z\text{-first}}(Y))$  and  $\text{WR}^1(\mathcal{A}, O_{z\text{-first}}(Y))$  partition  $V$  since the game  $(\mathcal{A}, O_{z\text{-first}}(Y))$ , being finitary, is determined (by Lemma 3). Since  $\text{WR}^0(\mathcal{A}, O_{\text{tail}}(Y))$  and  $\text{WR}^1(\mathcal{A}, O_{\text{tail}}(Y))$  are disjoint, the containments above must be equalities.

To prove the claim, recall that since  $\mathcal{A}$  is  $Y$ -unambiguous then  $O_{\text{tail}}^A(Y) \cap O_{\text{tail}}^A(\neg Y) = \emptyset$ . Hence, a play  $\pi$  in the game  $(\mathcal{A}, O_{\text{tail}}(Y))$  is won by Player 0 (resp. Player 1) if  $\pi$  has a suffix  $\pi'$  for which every cycle in  $\text{cycles}(\pi')$  is in  $Y$  (resp.  $\neg Y$ ). It is thus enough to show that: (†) for every  $\sigma \in \{0, 1\}$ , and every node  $v$  in  $\mathcal{A}$ , given a strategy  $S$  that is winning from  $v$  for Player  $\sigma$  in the game  $(\mathcal{A}, O_{z\text{-first}}(Y))$ , we can construct a strategy  $T$  for Player  $\sigma$  in the game  $(\mathcal{A}, O_{\text{tail}}(Y))$ , such that every play  $\pi \in \text{plays}(T, v)$  has a suffix  $\pi'$  for which every cycle in  $\text{cycles}(\pi')$  satisfies  $Y^\sigma$ .

Consider first the case where  $z \notin \text{reach}(S, v)$ , or that  $z \in \text{reach}(S, v)$  but that every path in  $\text{plays}(S, v)$  that visits  $z$  contains a cycle before the first occurrence of  $z$  on the path. Observe that this implies that  $S$  is winning from  $v$  in the game  $(\mathcal{A}, O_{\text{first}}(Y))$ . In this case we let  $T = S^\circlearrowright$ , where  $S^\circlearrowright$  is the pumping strategy of  $S$  (Definition 6). By Lemma 7 we have that, for every  $\pi \in \text{plays}(T, v)$ , all the cycles in  $\text{cycles}(\pi)$  satisfy  $Y^\sigma$ , and thus (†) holds with  $\pi' = \pi$ .

Consider now the remaining case that  $z \in \text{reach}(S, v)$  and that there is a simple path  $\rho$  from  $v$  to  $z$  which is consistent with  $S$ . Define a strategy  $S_z$  for Player  $\sigma$  in the game  $(\mathcal{A}, O_{\text{first}}(Y))$  as follows: for every  $u \in V^*V_\sigma$ , let  $S_z(u) := S(\rho u)$  if  $u$  starts in  $z$ , and otherwise define it arbitrarily (i.e.,  $S_z(u) = w$ , where  $w$  is some node with  $(\text{end}(u), w) \in E$ ). Observe that, for every  $\pi \in \text{plays}(S_z, z)$ , the play  $\rho\pi$  is in  $\text{plays}(S, v)$ , and since  $\rho$  is a simple path that ends in  $z$ , Player  $\sigma$  wins the play  $\rho\pi$  in the game  $(\mathcal{A}, O_{z\text{-first}}(Y))$  iff the first cycle on  $\pi$  satisfies  $Y^\sigma$ . Thus, since  $S$  is winning from  $v$ , we have that  $S_z$  is winning from  $z$  in the game  $(\mathcal{A}, O_{\text{first}}(Y))$ .

We can now construct the desired strategy  $T$  for Player  $\sigma$  in the game  $(\mathcal{A}, O_{\text{tail}}(Y))$ . The strategy  $T$  works as follows: as long as a play does not touch the node  $z$  the strategy  $T$  behaves like the pumping strategy  $S^\circlearrowright$  of  $S$ ; however, once (and if)  $z$  is reached,  $T$  erases its memory and switches to behave like the pumping strategy  $(S_z)^\circlearrowright$  of  $S_z$  (starting from  $z$ ). Recall that we have to show that every play  $\pi \in \text{plays}(T, v)$  has a suffix  $\pi'$  for which every cycle in  $\text{cycles}(\pi')$  satisfies  $Y^\sigma$ . Informally, if  $z$  does not appear on  $\pi$  then  $\pi$  is consistent with  $S^\circlearrowright$ , and by Lemma 7 every cycle in  $\text{cycles}(\pi)$  satisfies  $Y^\sigma$ , and we can take  $\pi' = \pi$ . On the other hand, if  $z$  appears on  $\pi$  then  $\text{tail}_z(\pi)$  is consistent with  $(S_z)^\circlearrowright$ , and thus by Lemma 7 every cycle in  $\text{cycles}(\text{tail}_z(\pi))$  satisfies  $Y^\sigma$ , and we can take  $\pi' = \text{tail}_z(\pi)$ .

Formally, define the strategy  $T$  as follows: for every  $u \in V^*V_\sigma$ ,

$$T(u) := \begin{cases} S^\circlearrowright(u) & \text{if } z \text{ does not appear on } u, \\ (S_z)^\circlearrowright(\text{tail}_z(u)) & \text{otherwise.} \end{cases}$$

Given  $\pi \in \text{plays}(T, v)$ , either  $z$  appears on  $\pi$  or not. If it does not, then  $\pi \in \text{plays}(S^\circlearrowright, v)$ , and by Lemma 7 every cycle  $C$  in  $\text{cycles}(\pi)$  is the first cycle of some path  $\rho$  consistent with  $S$  and starting with  $v$ , that only uses edges from  $\pi$ . Thus,  $z$  does not appear on  $\rho$ , and since  $S$  is winning from  $v$  in the game  $(\mathcal{A}, O_{z\text{-first}}(Y))$ , we have that  $C$  satisfies  $Y^\sigma$ . If  $z$  does appear

on  $\pi$ , we argue that  $\text{tail}_z(\pi) \in \text{plays}((S_z)^\circlearrowleft, z)$ , i.e., that  $\text{tail}_z(\pi)$  is consistent with  $(S_z)^\circlearrowleft$ . Indeed, if  $m = |\text{head}(\pi)|$ , then for every  $j \geq 1$  for which  $\text{start}(\text{tail}_z(\pi)_j) \in V_\sigma$  we have:

$$\begin{aligned}\text{end}(\text{tail}_z(\pi)_j) &= T(\pi[1, m+j]) \\ &= (S_z)^\circlearrowleft(\text{tail}_z(\pi[1, m+j])) \\ &= (S_z)^\circlearrowleft(\pi[m+1, m+j]) \\ &= (S_z)^\circlearrowleft((\text{tail}_z(\pi))[1, j]).\end{aligned}$$

By Lemma 7, since  $\text{tail}_z(\pi) \in \text{plays}((S_z)^\circlearrowleft, z)$ , every cycle in  $\text{cycles}(\text{tail}_z(\pi))$  satisfies  $Y^\sigma$ . Overall, in the first case (†) holds with  $\pi' = \pi$ , and in the second with  $\pi' = \text{tail}_z(\pi)$ , which completes the proof of the claim.  $\square$

This concludes the proof of Theorem 9.

## 8. A recipe for proving that a game is memoryless determined

We synthesise the results of the previous section and provide a practical way of deducing uniform memoryless determinacy of many infinite-duration games.

First, we get the following sufficient condition for memoryless determinacy from Theorems 8 and 9.

**Theorem 10.** Let  $Y$  be a cycle property that is closed under cyclic permutations, and such that both  $Y$  and  $\neg Y$  are closed under concatenation. Let  $W$  be a winning condition. Every  $Y$ -greedy game  $(\mathcal{A}, O(W))$  is uniform memoryless determined.

Second, many winning conditions for infinite-duration games (for example parity and mean-payoff) are such that the outcome of a play does not depend on any finite prefix of the play, but only on some “infinite” property of the play. Such winning conditions are usually called *prefix-independent*. Formally:

**Definition 9.** Say that a winning condition  $W \subseteq \mathbb{U}^\omega$  is *prefix-independent* if  $c_1c_2\dots \in W$  implies that the suffix  $c_ic_{i+1}\dots \in W$  for every  $i \in \mathbb{N}$ .

In practice, showing whether or not a given  $W$  is prefix-independent is usually very easy. The relevance of prefix-independence to our work is captured by the following theorem.

**Theorem 11.** Let  $W$  be a prefix-independent winning condition, and let  $Y$  be a cycle-property that is closed under cyclic permutations. For every arena  $\mathcal{A}$ , if the game  $(\mathcal{A}, O(W))$  is  $Y$ -greedy then it is uniform memoryless determined.

**Proof.** By Theorem 8 it is sufficient to prove that every arena  $\mathcal{A}$  is  $Y$ -resettable. By Propositions 3 and 4 it is thus sufficient to prove that  $\mathcal{A}$  is  $Y$ -unambiguous. So, assume by way of contradiction there is a play  $\pi$  and indices  $a, b \in \mathbb{N}$  be such that every cycle in  $\text{cycles}(\pi[a, \infty])$  satisfies  $Y$ , and every cycle in  $\text{cycles}(\pi[b, \infty])$  satisfies  $\neg Y$ . Since  $(\mathcal{A}, O(W))$  is  $Y$ -greedy on  $\mathcal{A}$  we get that, in the game  $(\mathcal{A}, O(W))$ , Player 0 wins  $\pi[a, \infty]$ , and Player 1 wins  $\pi[b, \infty]$ . But this is a contradiction to the assumption that  $W$  is prefix-independent.  $\square$

Given a winning condition  $W$  and a set of arenas of interest  $\mathcal{C}$  (in many cases  $\mathcal{C}$  is taken to be all arenas), Theorems 10 and 11 suggest the following recipe for proving that the game  $(\mathcal{A}, O(W))$  is uniform memoryless determined for every  $\mathcal{A} \in \mathcal{C}$ .

- Step 1. Finitise the winning condition  $W \subseteq \mathbb{U}^\omega$  to get a cycle property  $Y \subseteq \mathbb{U}^*$  that is closed under cyclic permutations.
- Step 2. Show that the game  $(\mathcal{A}, O(W))$  is  $Y$ -greedy for every  $\mathcal{A} \in \mathcal{C}$ .
- Step 3. Show that either:
  - (a) Both  $Y$  and  $\neg Y$  are closed under concatenation; or that:
  - (b)  $W$  is prefix independent.

We illustrate the use of the recipe with some examples.

**Example 1.** We will use the recipe to prove that every parity game is memoryless determined. For Step 1, a natural finitisation of the parity condition  $W \subset \mathbb{Z}^\omega$  — which says that the largest priority occurring infinitely often is even — is the cycle property  $\text{cyc-Parity} \subset \mathbb{Z}^*$  which says that the largest priority occurring is even. This property is clearly closed under

cyclic permutations. For Step 2, it is easy to verify that every parity game is cyc-Parity-greedy, and for Step 3, it is immediate that both cyc-Parity and  $\neg$ cyc-Parity are closed under concatenation (as it happens, it is also easy to check that  $W$  is prefix-independent).

**Example 2.** We now consider a slightly more subtle application of the recipe. Consider the following game,<sup>11</sup> played on an arena  $\mathcal{A}$  whose vertices are labelled<sup>12</sup> using the alphabet  $\mathbb{U} = \{a, b\}$ . The winning condition  $W \subset \mathbb{U}^\omega$  consists of those infinite sequences that contain infinitely many  $a$ 's and infinitely many  $b$ 's. The natural finite version of  $W$  is the cycle property  $Y \subset \mathbb{U}^*$  consisting of strings which contain at least one  $a$  and at least one  $b$ , and it is clearly closed under cyclic permutations. To see that  $W$  is  $Y$ -greedy on  $\mathcal{A}$ , observe that if all cycles in  $\text{cycles}(\pi)$  satisfy  $Y$  then certainly  $\pi \in W$ . On the other hand, if all cycles in  $\text{cycles}(\pi)$  satisfy  $\neg Y$  then no edge that appears on such a cycle goes from a node labelled  $a$  to a node labelled  $b$ . Thus by Lemma 1 (Page 5)  $\pi$  does not satisfy  $W$ .

Unfortunately,  $\neg Y$  is not closed under concatenation, which prevents us from applying Step 3(a). However, since  $W$  is clearly prefix-independent, we can apply Step 3(b) and conclude that  $(\mathcal{A}, O(W))$  is memoryless determined.

**Example 3.** We conclude with a more sophisticated use of the recipe, applied to the initial credit problem of energy games. We show that either there is an initial credit with which Player 0 (the “energy” player) wins, or that for every initial credit Player 1 wins. In both cases, we show that the winner can use a memoryless strategy. A natural finitisation of the energy condition  $W_r \subset \mathbb{Z}^\omega$  – which says that at every point along a play, the sum of the initial credit  $r$  and the labels of the edges already traversed is not negative – is the cycle property cyc-Energy  $\subset \mathbb{Z}^*$  which says that the sum of the numbers is non-negative. This property is clearly closed under cyclic permutations. Given an arena  $\mathcal{A}$ , consider the game  $(\mathcal{A}, O_{\text{all}}(\text{cyc-Energy}))$ . We first claim that if the initial credit is at least  $r = -t(|V| - 1)$ , where  $t$  is the minimum amongst the negative weights of the arena, the winning regions of the energy game and the game  $(\mathcal{A}, O_{\text{all}}(\text{cyc-Energy}))$  coincide and winning strategies transfer from the latter to the former. To see that, observe that a play won by Player 1 in  $(\mathcal{A}, O_{\text{all}}(\text{cyc-Energy}))$  is also won by her in the energy game since the energy along the play tends to  $-\infty$ . On the other hand, let  $\pi$  be a play won by Player 0 in  $(\mathcal{A}, O_{\text{all}}(\text{cyc-Energy}))$ , and consider some prefix  $\pi'$  of it. Recall that, by Lemma 1, at most  $|V| - 1$  edges are not on  $\text{cycles}(\pi')$ , and thus the energy level at the end of  $\pi'$  is at least the initial credit plus  $t(|V| - 1)$ . Hence, an initial credit of  $r$  suffices for  $\pi$  to be winning for Player 0 also in the energy game. It remains to show, using the recipe, that  $(\mathcal{A}, O_{\text{all}}(\text{cyc-Energy}))$  is memoryless determined. As noted before (see Example 1 after Definition 5) every  $(\mathcal{A}, O_{\text{all}}(Y))$  is  $Y$ -greedy, which completes Step 2. Since step 3a is immediate for cyc-Energy we are done.

## 9. Discussion and future work

The central algorithmic problem for a class of determined graph games is to decide, given an arena and a starting vertex, which of the players has a winning strategy. We have seen a PSPACE upper bound on the complexity of solving games in FCG( $Y$ ) (assuming  $Y$  is computable in PSPACE), and that there are very simple cycle properties  $Y$  for which the complexity is PSPACE-complete. What about other  $Y$ 's such as cyc-Parity and cyc-MeanPayoff<sub>v</sub>? Our Strategy Transfer result (Theorem 7) implies that the complexity of solving FCG( $Y$ ) is the same as that of solving  $Y$ -greedy games. For instance, since parity games are cyc-Parity-greedy, and the complexity of solving them is not known to be in PTIME – except on restricted classes of arenas, e.g., bounded DAG-width ([2]) and bounded trap-depth ([9]) – the same is true of the complexity of solving games from FCG(cyc-Parity).

On the other hand, since there are cycle properties  $Y$  such that some games in FCG( $Y$ ) are memoryless determined and some are not (for instance, take  $Y = \text{cyc-EvenLen}$ ), the following algorithmic problem naturally presents itself: what is the complexity of deciding, given (a finite description of)  $Y$  and an arena  $\mathcal{A}$ , whether or not the game  $(\mathcal{A}, O_{\text{first}}(Y))$  is memoryless determined? We believe that this problem can be addressed using techniques developed in this paper.

Finally, this paper has dealt with qualitative games (i.e., a player either wins or loses). The exact nature of the theory of quantitative first-cycle games over general cycle properties  $Y$  is still to be explored. To what extent do our techniques generalise to quantitative games? or to stochastic ones?

## Acknowledgments

We thank Erich Grädel for stimulating feedback which led to an improvement of the recipe in Section 8. We thank the referees for their useful suggestions and careful reading. Benjamin Aminof was supported by the Austrian National Research Network S11403-N23 (RiSE) of the Austrian Science Fund (FWF) and by the Vienna Science and Technology Fund (WWTF) through grant ICT12-059.

<sup>11</sup> This example was kindly brought to our attention by Erich Grädel, and it prompted us to enhance the recipe that was published in the preliminary work [1] to include Step 3(b).

<sup>12</sup> Recall Remark 1 that states that all the results in this paper also apply to vertex labelling.

## References

- [1] B. Aminof, S. Rubin, First cycle games, in: Proceedings 2nd International Workshop on Strategic Reasoning, SR 2014, in: EPTCS, vol. 146, 2014, pp. 83–90.
- [2] D. Berwanger, A. Dawar, P. Hunter, S. Kreutzer, Dag-width and parity games, in: B. Durand, W. Thomas (Eds.), Symposium on Theoretical Aspects of Computer Science, STACS 2006, in: Lecture Notes in Computer Science, vol. 3884, Springer, 2006, pp. 524–536.
- [3] A. Bianco, M. Faella, F. Mogavero, A. Murano, Exploring the boundary of half-positionality, Ann. Math. Artif. Intell. 62 (1–2) (2011) 55–77.
- [4] H. Björklund, S. Sandberg, S.G. Vorobyov, Memoryless determinacy of parity and mean payoff games: a simple proof, Theor. Comput. Sci. 310 (1–3) (2004) 365–378.
- [5] K. Chatterjee, L. Doyen, Energy parity games, Theor. Comput. Sci. 458 (2012) 49–60.
- [6] A. Ehrenfeucht, J. Mycielski, Positional strategies for mean payoff games, Int. J. Game Theory 8 (2) (1979) 109–113.
- [7] D. Gale, F.M. Stewart, Infinite games with perfect information, in: H. Kuhn, A. Tucker (Eds.), Contributions to the Theory of Games, Volume II, in: Annals of Mathematics Studies, vol. AM-28, Princeton University Press, 1953, pp. 245–266.
- [8] H. Gimbert, W. Zielonka, Games where you can play optimally without any memory, in: M. Abadi, L. de Alfaro (Eds.), International Conference on Concurrency Theory, CONCUR 2005, in: Lecture Notes in Computer Science, vol. 3653, 2005, pp. 428–442.
- [9] A. Grinshpun, P. Phalitnonkiat, S. Rubin, A. Tarfulea, Alternating traps in Muller and parity games, Theor. Comput. Sci. 521 (2014) 73–91.
- [10] E. Kopczynski, Half-positional determinacy of infinite games, in: International Colloquium on Automata, Languages and Programming, ICALP 2006, 2006, pp. 336–347.
- [11] M. Sipser, Introduction to the Theory of Computation, PWS Publishing Company, 1997.
- [12] U. Zwick, M. Paterson, The complexity of mean payoff games on graphs, Theor. Comput. Sci. 158 (1&2) (1996) 343–359.

#### **4.10 Publication**

The rest of this page is intentionally left blank

# Imperfect-Information Games and Generalized Planning

**Giuseppe De Giacomo**

SAPIENZA Università di Roma  
Rome, Italy  
degiacomo@dis.uniroma1.it

**Aniello Murano, Sasha Rubin, Antonio Di Stasio**

Università degli Studi di Napoli “Federico II”  
Naples, Italy  
first.last@unina.it

## Abstract

We study a generalized form of planning under partial observability, in which we have multiple, possibly infinitely many, planning domains with the same actions and observations, and goals expressed over observations, which are possibly temporally extended. By building on work on two-player (non-probabilistic) games with imperfect information in the Formal Methods literature, we devise a general technique, generalizing the belief-state construction, to remove partial observability. This reduces the planning problem to a game of perfect information with a tight correspondence between plans and strategies. Then we instantiate the technique and solve some generalized-planning problems.

## 1 Introduction

Automated planning is a fundamental problem in Artificial Intelligence. Given a deterministic dynamic system with a single known initial state and a goal condition, automated planning consists of finding a sequences of actions (the plan) to be performed by agents in order to achieve the goal [M. Ghallab and Traverso, 2008]. The application of this notion in real-dynamic worlds is limited, in many situations, by three facts: i) the number of objects is neither small nor predetermined, ii) the agent is limited by its observations, iii) the agent wants to realize a goal that extends over time. For example, a preprogrammed driverless car cannot know in advance the number of obstacles it will enter in a road, or the positions of the other cars not in its view, though it wants to realize, among other goals, that every time it sees an obstacle it avoids it. This has inspired research in recent years on *generalized forms of planning* including conditional planning in partially observable domains [Levesque, 1996; Rintanen, 2004], planning with incomplete information for temporally extended goals [De Giacomo and Vardi, 1999; Bertoli and Pistore, 2004] and generalized planning for multiple domains or infinite domains [Levesque, 2005; Srivastava et al., 2008; Bonet et al., 2009; Hu and Levesque, 2010; Hu and De Giacomo, 2011; Srivastava et al., 2011; Felli et al., 2012; Srivastava et al., 2015].

We use the following running example, taken from [Hu and Levesque, 2010], to illustrate a generalized form of planning:

*Example 1* (Tree Chopping). The goal is to chop down a tree, and store the axe. The number of chops needed to fell the tree is unknown, but a look-action checks whether the tree is up or down. Intuitively, a plan solving this problem alternates looking and chopping until the tree is seen to be down, and then stores the axe. This scenario can be formalized as a partially-observable planning problem on a single infinite domain (Example 2, Figure 1), or on the disjoint union of infinitely many finite domains (Section 3).

The standard approach to solve planning under partial observability for *finite* domains is to reduce them to planning under complete observability. This is done by using the *belief-state construction* that removes partial observability and passes to the *belief-space* [Goldman and Boddy, 1996; Bertoli et al., 2006]. The motivating problem of this work is to solve planning problems on infinite domains, and thus we are naturally lead to the problem of removing partial-observability from infinite domains.

In this paper we adopt the Formal Methods point of view and consider generalized planning as a game  $\mathbf{G}$  of imperfect information, i.e., where the player under control has partial observability. The game  $\mathbf{G}$  may be infinite, i.e., have infinitely many states.

Our technical contribution (Theorem 4.5) is a general sound and complete mathematical technique for removing imperfect information from a possibly infinite game  $\mathbf{G}$  to get a game  $\mathbf{G}^\beta$ , possibly infinite, of perfect information. Our method builds on the classic belief-state construction [Reif, 1984; Goldman and Boddy, 1996; Raskin et al., 2007], also adopted in POMDPs [Kaelbling et al., 1998; LaValle, 2006].<sup>1</sup> The classic belief-state construction fails for certain infinite games, see Example 3. We introduce a new component to the classic belief-state construction that isolates only those plays in the belief-space that correspond to plays in  $\mathbf{G}$ . This new component is necessary and sufficient to solve the general case and capture all infinite games  $\mathbf{G}$ .

We apply our technique to the decision problem that asks, given a game of imperfect information, if the player under control has a winning strategy (this corresponds to deciding if there is a plan for a given planning instance). We remark that we consider strategies and plans that may de-

<sup>1</sup>However, our work considers nondeterminism rather than probability, and qualitative objectives rather than quantitative objectives.

pend on the history of the observations, not just the last observation. Besides showing how to solve the running Tree Chopping example, we report two cases. The first case is planning under partial observability for temporally extended goals expressed in LTL in finite domains (or a finite set of infinite domains sharing the same observations). This case generalizes well-known results in the AI literature [De Giacomo and Vardi, 1999; Rintanen, 2004; Bertoli *et al.*, 2006; Hu and De Giacomo, 2011; Felli *et al.*, 2012]. The second case involves infinite domains. Note that because game solving is undecidable for computable infinite games (simply code the configuration space of a Turing Machine), solving games with infinite domains requires further computability assumptions. We focus on games generated by pushdown automata; these are infinite games that recently attracted the interest of the AI community [Murano and Perelli, 2015; Chen *et al.*, 2016]. In particular these games have been solved assuming perfect information. By applying our technique, we extend their results to deal with imperfect information under the assumption that the stack remains observable (it is known that making the stack unobservable leads to undecidability [Azhar *et al.*, 2001]).

## 2 Generalized-Planning Games

In this section we define generalized-planning (GP) games, known as games of imperfect information in the Formal Methods literature [Raskin *et al.*, 2007], that capture many generalized forms of planning.

Informally, two players (agent and environment) play on a transition-system. Play proceeds in rounds. In each round, from the current state  $s$  of the transition-system, the agent observes  $obs(s)$  (some information about the current state), and picks an action  $a$  from the set of actions  $Ac$ , and then the environment picks an element of  $tr(s, a)$  ( $tr$  is the transition function of the transition-system) to become the new current state. Note that the players are asymmetric, i.e., the agent picks actions and the environment resolves non-determinism.

*Notation.* Write  $X^\omega$  for the set of infinite sequences whose elements are from the set  $X$ , write  $X^*$  for the finite sequences, and  $X^+$  for the finite non-empty sequences. If  $\pi$  is a finite sequence then  $Last(\pi)$  denotes its last element. The positive integers are denoted  $\mathbb{N}$ , and  $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$ .

**Linear-temporal logic.** We define LTL over a finite set of letters  $\Sigma$ .<sup>2</sup> The *formulas of LTL (over  $\Sigma$ )* are generated by the following grammar:  $\varphi ::= x \mid \varphi \wedge \varphi \mid \neg \varphi \mid X \varphi \mid \varphi \vee \varphi$  where  $x \in \Sigma$ . We introduce the usual abbreviations for, e.g.,  $\vee, F$ . Formulas of LTL (over  $\Sigma$ ) are interpreted over infinite words  $\alpha \in \Sigma^\omega$ . Define the satisfaction relation  $\models$  as follows: i)  $(\alpha, n) \models x$  iff  $\alpha_n = x$ ; ii)  $(\alpha, n) \models \varphi_1 \wedge \varphi_2$  iff  $(\alpha, n) \models \varphi_i$  for  $i = 1, 2$ ; iii)  $(\alpha, n) \models \neg \varphi$  iff it is not the case that  $(\alpha, n) \models \varphi$ ; iv)  $(\alpha, n) \models X \varphi$  iff  $(\alpha, n+1) \models \varphi$ ; v)  $(\alpha, n) \models \varphi_1 \vee \varphi_2$  iff there exists  $i \geq n$  such that  $(\alpha, i) \models \varphi_2$  and for all  $j \in [n, i)$ ,  $(\alpha, j) \models \varphi_1$ . Write  $\alpha \models \varphi$  if  $(\alpha, 0) \models \varphi$  and say that  $\alpha$  *satisfies* the LTL formula  $\varphi$ .

<sup>2</sup>This is without loss of generality, since if LTL were defined over a set of atomic propositions AP we let  $\Sigma = 2^{AP}$  and replace atoms  $p \in AP$  by  $\bigvee_{p \in x} x$  to get equivalent LTL formulas over  $\Sigma$ .

**Arenas.** An *arena of imperfect information*, or simply an arena, is a tuple  $\mathbf{A} = (S, I, Ac, tr, Obs, obs)$ , where  $S$  is a (possibly infinite) set of states,  $I \subseteq S$  is the set of *initial states*,  $Ac$  is a finite set of *actions*, and  $tr : S \times Ac \rightarrow 2^S \setminus \{\emptyset\}$  is the *transition function*,  $Obs$  is a (possibly infinite) set of *observations*, and  $obs : S \rightarrow Obs$ , the *observation function*, maps each state to an observation. We extend  $tr$  to sets of states: for  $\emptyset \neq Q \subseteq S$ , let  $tr(Q, a)$  denote the set  $\bigcup_{q \in Q} tr(q, a)$ .

Sets of the form  $obs^{-1}(x)$  for  $x \in Obs$  are called *observation sets*. The set of all observation sets is denoted  $ObsSet$ . Non-empty subsets of observation sets are called *belief-states*. Informally, a belief-state is a subset of the states of the game that the play could be in after a given finite sequence of observations and actions.

**Finite and finitely-branching.** An arena is *finite* if  $S$  is finite, and *infinite* otherwise. An arena is *finitely-branching* if i)  $I$  is finite, and ii) for every  $s, a$  the cardinality of  $tr(s, a)$  is finite. Clearly, being finite implies being finitely-branching.

**Strategies.** A *play* in  $\mathbf{A}$  is an infinite sequence  $\pi = s_0 a_0 s_1 a_1 s_2 a_2 \dots$  such that  $s_0 \in I$  and for all  $i \in \mathbb{N}_0$ ,  $s_{i+1} \in tr(s_i, a_i)$ . A *history*  $h = s_0 a_0 \dots s_{n-1} a_{n-1} s_n$  is a finite prefix of a play ending in a state. The set of plays is denoted  $Ply(\mathbf{A})$ , and the set of histories is denoted  $Hist(\mathbf{A})$  (we drop  $\mathbf{A}$  when it is clear from the context). For a history or play  $\pi = s_0 a_0 s_1 a_1 \dots$  write  $obs(\pi)$  for the sequence  $obs(s_0) a_0 obs(s_1) a_1 \dots$ . A *strategy* (for the agent) is a function  $\sigma : Hist(\mathbf{A}) \rightarrow Ac$ . A strategy is *observational* if  $obs(h) = obs(h')$  implies  $\sigma(h) = \sigma(h')$ . In Section 3 we will briefly mention an alternative (but essentially equivalent) definition of observational strategy, i.e., as a function  $Obs^+ \rightarrow Ac$ . We do not define strategies for the environment. A play  $\pi = s_0 a_0 s_1 a_1 \dots$  is *consistent* with a strategy  $\sigma$  if for all  $i \in \mathbb{N}$  we have that if  $h \in Hist(\mathbf{A})$  is a prefix of  $\pi$ , say  $h = s_0 a_0 \dots s_{n-1} a_{n-1} s_n$ , then  $\sigma(h) = a_{n+1}$ .

**GP Games.** A *generalized-planning (GP) game*, is a tuple  $\mathbf{G} = \langle \mathbf{A}, W \rangle$  where the *winning objective*  $W \subseteq Obs^\omega$  is a set of infinite sequences of observation sets. A *GP game with restriction* is a tuple  $\mathbf{G} = \langle \mathbf{A}, W, F \rangle$  where, in addition,  $F \subseteq S^\omega$  is the *restriction*. Note that unlike the winning objective, the restriction need not be closed under observations. A GP game is *finite* (resp. *finitely branching*) if the arena  $\mathbf{A}$  is finite (resp. finitely branching).

**Winning.** A strategy  $\sigma$  is *winning* in  $\mathbf{G} = \langle \mathbf{A}, W \rangle$  if for every play  $\pi \in Ply(\mathbf{A})$  consistent with  $\sigma$ , we have that  $obs(\pi) \in W$ . Similarly, a strategy is *winning* in  $\mathbf{G} = \langle \mathbf{A}, W, F \rangle$  if for every play  $\pi \in Ply(\mathbf{A})$  consistent with  $\sigma$ , if  $\pi \in F$  then  $obs(\pi) \in W$ . Note that a strategy is winning in  $\langle \mathbf{A}, W, Ply(\mathbf{A}) \rangle$  if and only if it is winning in  $\langle \mathbf{A}, W \rangle$ .

**Solving a GP game.** A central decision problem is the following, called *solving a GP game*: given a (finite representation of a) GP game of imperfect information  $\mathbf{G}$ , decide if the agent has a winning observational-strategy.

**GP games of perfect information.** An arena/GP game has *perfect information* if  $Obs = S$  and  $obs(s) = s$  for all  $s$ . We thus suppress mentioning  $Obs$  and  $obs$  completely, e.g., we write  $\mathbf{A} = (S, I, Ac, tr)$  and  $W, F \subseteq S^\omega$ . Note that in a GP game of perfect information every strategy is observational.

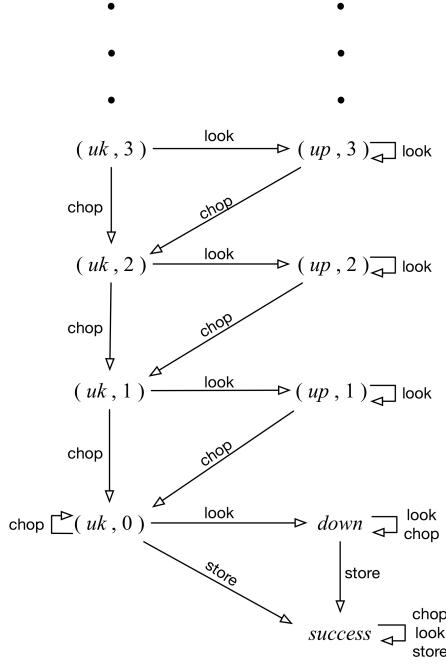


Figure 1: Part of the arena  $A_{\text{chop}}$  (missing edges go to the *failure* state). The numbers correspond to the number of chops required to fell the tree. The arena is not finitely-branching since it has infinitely many initial states  $\{uk\} \times \mathbb{N}_0$ .

*Example 2* (continued). We formalize the tree-chopping planning problem. Define the GP game  $G_{\text{chop}} = \langle A_{\text{chop}}, W \rangle$  where  $A_{\text{chop}} = \langle S, I, Ac, tr, Obs, obs \rangle$ , and:

- $S = \{down, success, failure\} \cup (\{uk\} \times \mathbb{N}_0) \cup (\{up\} \times \mathbb{N})$ ,
- $Ac = \{\text{chop}, \text{look}, \text{store}\}$ ,  $I = \{uk\} \times \mathbb{N}$ ,
- $tr$  is illustrated in Figure 1,
- $Obs = \{\text{DN}, \checkmark, \times, \text{UK}, \text{UP}\}$ ,
- $obs$  maps  $down \mapsto \text{DN}$ ,  $(up, i) \mapsto \text{UP}$  for  $i \in \mathbb{N}$ ,  $(uk, i) \mapsto \text{UK}$  for  $i \in \mathbb{N}_0$ ,  $failure \mapsto \times$ , and  $success \mapsto \checkmark$ , and
- the objective  $W$  is defined as  $\alpha \in W$  iff  $\alpha \models F \checkmark$ .

The mentioned plan is formalized as the observational-strategy  $\sigma_{\text{chop}}$  that maps any history ending in  $(uk, i)$  to look (for  $i \in \mathbb{N}_0$ ),  $(up, i)$  to chop (for  $i \in \mathbb{N}$ ),  $down$  to store, and all others arbitrarily (say to store).

Note:  $\sigma_{\text{chop}}$  is a winning strategy, i.e., no matter which initial state the environment chooses, the strategy ensures that the play (it is unique because the rest of the GP game is deterministic) reaches the state *success* having observation  $\checkmark$ .

### 3 Generalized-Planning Games and Generalized Forms of Planning

In this section we establish that generalized-planning (GP) games can model many different types of planning from the

AI literature, including a variety of generalized forms of planning:

1. planning on finite transition-systems, deterministic actions, actions with conditional effects, partially observable states, incomplete information on the initial state, and temporally extended goals [De Giacomo and Vardi, 1999];
2. planning under partial observability with finitely many state variables, nondeterministic actions, reachability goals, and partial observability [Rintanen, 2004];
3. planning on finite transition systems, nondeterministic actions, looking for strong plans (i.e., adversarial nondeterminism) [Bertoli *et al.*, 2006];
4. generalized planning, consisting of multiple (possibly infinitely many) related finite planning instances [Hu and Levesque, 2010; Hu and De Giacomo, 2011].

We discuss the latter in detail. Following [Hu and De Giacomo, 2011], a *generalized-planning problem*  $\mathfrak{P}$  is defined as a sequence of related classical planning problems. In our terminology, fix finite sets  $Ac, Obs$  and let  $\mathfrak{P}$  be a countable sequence  $G_1, G_2, \dots$  where each  $G_n$  is a finite GP game of the form  $\langle S_n, \{\iota_n\}, Ac, tr_n, Obs, obs_n, W_n \rangle$ . In [Hu and De Giacomo, 2011], a plan is an observational-strategy  $p : Obs^+ \rightarrow Ac$ , and a solution is a single plan that solves all of the GP games in the sequence. Now, we view  $\mathfrak{P}$  as a single infinite GP game as follows. Let  $G_{\mathfrak{P}}$  denote the disjoint union of the GP games in  $\mathfrak{P}$ . Formally,  $G_{\mathfrak{P}} = \langle S, I, Ac, tr, Obs, obs, W \rangle$  where

- $S = \{(s, n) : s \in S_n, n \in \mathbb{N}\}$ ,
- $I = \{(\iota_n, n) : n \in \mathbb{N}\}$ ,
- $tr((s, n), a) = \{(t, n) : t \in tr_n(s, a)\}$ ,
- $obs(s, n) = obs_n(s)$ ,
- $W = \bigcup_n W_n$ .

Then: there is a correspondence between solutions for  $\mathfrak{P}$  and winning observational-strategies in  $G_{\mathfrak{P}}$ .

For example, consider the tree-chopping problem as formalized in [Hu and Levesque, 2010; Hu and De Giacomo, 2011]: there are infinitely many planning instances which are identical except for an integer parameter denoting the number of chops required to fell the tree. The objective for all instances is to fell the tree. Using the translation above we get a GP game with an infinite arena which resembles (and, in fact, can be transformed to) the GP game in Example 2.

### 4 Generalized Belief-State Construction

In this section we show how to remove imperfect information from generalized-planning (GP) games  $G$ . That is, we give a transformation of GP games of imperfect information  $G$  to GP games of perfect information  $G^{\beta}$  such that the agent has a winning observational-strategy in  $G$  if and only if the agent has a winning strategy in  $G^{\beta}$ . The translation is based on the classic belief-state construction [Reif, 1984; Raskin *et al.*, 2007]. Thus, we begin with a recap of that construction.

**Belief-state Arena.**<sup>3</sup> Let  $\mathbf{A} = (S, I, Ac, \text{tr}, \text{Obs}, obs)$  be an arena (not necessarily finite). Recall from Section 2 that observation sets are of the form  $obs^{-1}(x)$  for  $x \in \text{Obs}$ , and are collectively denoted ObsSet. Define the arena of perfect information  $\mathbf{A}^\beta = (S^\beta, I^\beta, Ac, \text{tr}^\beta)$  where,

- $S^\beta$  is the set of belief-states, i.e., the non-empty subsets of the observation-sets,
- $I^\beta$  consists of all belief-states of the form  $I \cap X$  for  $X \in \text{ObsSet}$ ,
- $\text{tr}^\beta(Q, a)$  consists of all belief-states of the form  $\text{tr}(Q, a) \cap X$  for  $X \in \text{ObsSet}$ .

The idea is that  $Q \in S^\beta$  represents a refinement of the observation set: the agent, knowing the structure of  $G$  ahead of time, and the sequence of observations so far in the game, may deduce that it is in fact in a state from  $Q$  which may be a strict subset of its corresponding observation set  $X$ .<sup>4</sup>

**NB.** Since  $\mathbf{A}^\beta$  is an arena, we can talk of its histories and plays. Although we defined  $S^\beta$  to be the set of all belief-states, only those belief-states that are reachable from  $I^\beta$  are relevant. Thus, overload notation and write  $S^\beta$  for the set of reachable belief-states, and  $\mathbf{A}^\beta$  for the corresponding arena. This notation has practical relevance since if  $\mathbf{A}$  is countable there are uncountably many belief-states; but in many cases only countably many (or, as in the running example, finitely many) reachable belief-states.

The intuition for the rest of this section is illustrated in the next example.

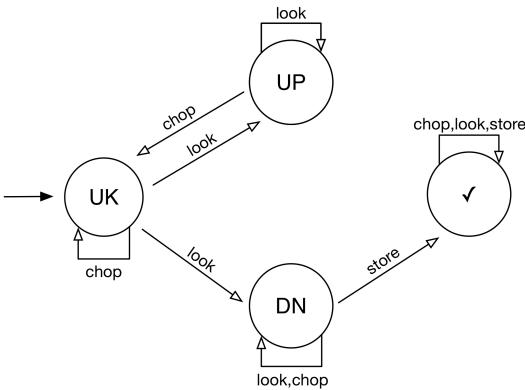


Figure 2: Part of the arena  $\mathbf{A}_{\text{chop}}^\beta$  (missing edges go to the failure state). Each circle is a belief-state. The winning objective is  $F \checkmark$ , and the restriction is  $\neg G F[\text{UK} \wedge X \text{ look} \wedge X \text{ UP}]$ .

*Example 3 (continued).* Figure 2 shows the arena  $\mathbf{A}_{\text{chop}}^\beta$  corresponding to the arena from tree-chopping game  $\mathbf{G}_{\text{chop}}$ , i.e.,

<sup>3</sup>In the AI literature, this is sometimes called the *belief-space*.

<sup>4</sup>To illustrate simply, suppose there is a unique initial state  $s$ , and that it is observationally equivalent to other states. At the beginning of the game the agent can deduce that it must be in  $s$ . Thus, its initial belief-state is  $\{s\}$  and not its observation-set  $obs^{-1}(obs(s))$ . This belief can (and, in general, must) be exploited if the agent is to win.

- $S^\beta$  are the following belief-states:  $\{(uk, n) \mid n \in \mathbb{N}_0\}$ , denoted UK;  $\{(up, n) \mid n \in \mathbb{N}\}$ , denoted UP;  $\{down\}$ , denoted DN;  $\{\text{success}\}$ , denoted  $\checkmark$ ; and  $\{\text{failure}\}$ .
- $I^\beta$  is the belief-state UK,
- and  $\text{tr}^\beta$  is shown in the figure.

Note that the agent does not have a winning strategy in the GP game with arena  $\mathbf{A}_{\text{chop}}^\beta$  and winning condition  $F \checkmark$ . The informal reason is that the strategy  $\sigma_{\text{chop}}$  (which codifies “alternately look and chop until the tree is sensed to be down, and then store the axe”), which wins in  $\mathbf{G}$ , does not work. The reason is that after every look the opponent can transition to UP (and never DN), resulting in the play  $\rho = (\text{UK look UP chop})^\omega$ , i.e., the repetition of (UK look UP chop) forever. Such a play of  $\mathbf{A}_{\text{chop}}^\beta$  does not correspond to any play in  $\mathbf{A}_{\text{chop}}$ . This is a well known phenomena of the standard belief-set construction [Sardiña *et al.*, 2006], which our construction overcomes by adding a restriction that removes from consideration plays such as  $\rho$  (as discussed in Example 5).

The following definition is central. It maps a history  $h \in \text{Hist}(\mathbf{A})$  to the corresponding history  $h^\beta \in \text{Hist}(\mathbf{A}^\beta)$  of belief-states.

**Definition 4.1.** For  $h \in \text{Hist}(\mathbf{A})$  define  $h^\beta \in \text{Hist}(\mathbf{A}^\beta)$  inductively as follows.

- For  $s \in I$ , define  $s^\beta \in I^\beta$  to be  $I \cap obs^{-1}(obs(s))$ . In words,  $s^\beta$  is the set of initial states the GP game could be in given the observation  $obs(s)$ .
- If  $h \in \text{Hist}(\mathbf{A})$ ,  $a \in Ac$ ,  $s \in S$ , then  $(has)^\beta := h^\beta a B$  where  $B := \text{tr}(Last(h^\beta), a) \cap obs^{-1}(obs(s))$ . In words,  $B$  is the set of possible states the GP game could be in given the observation sequence  $obs(has)$ .

In the same way, for  $\pi \in \text{Ply}(\mathbf{A})$  define  $\pi^\beta \in \text{Ply}(\mathbf{A}^\beta)$ . Extend the map pointwise to sets of plays  $P \subseteq \text{Ply}(\mathbf{A})$ , i.e., define  $P^\beta := \{\pi^\beta \in \text{Ply}(\mathbf{A}^\beta) \mid \pi \in P\}$ . Finally, we give notation to the special case that  $P = \text{Ply}(\mathbf{A})$ : write  $\text{Im}(\mathbf{A})$  for the set  $\{\pi^\beta \mid \pi \in \text{Ply}(\mathbf{A})\}$ , called the *image* of  $\mathbf{A}$ .

By definition,  $\text{Im}(\mathbf{A}) \subseteq \text{Ply}(\mathbf{A}^\beta)$ . However, the converse is not always true.

*Example 4 (continued).* There is a play of  $\mathbf{A}_{\text{chop}}^\beta$  that is not in  $\text{Im}(\mathbf{A}_{\text{chop}})$ , e.g.,  $\rho = (uk \text{ look } up \text{ chop})^\omega$ . Indeed, suppose  $\pi^\beta = \rho$  and consider the sequence of counter values of  $\pi$ . Every look action establishes that the current counter value in  $\pi$  is positive (this is the meaning of the tree being *up*), but every chop action reduces the current counter value by one. This contradicts that counter values are always non-negative.

*Remark 4.2.* If  $\mathbf{A}$  is finitely-branching then  $\text{Im}(\mathbf{A}) = \text{Ply}(\mathbf{A}^\beta)$ . To see this, let  $\rho$  be a play in  $\mathbf{A}^\beta$ , and consider the forest whose nodes are the histories  $h$  of  $\mathbf{A}$  such that  $h^\beta$  is a prefix of  $\rho$ . Each tree in the forest is finitely branching (because  $\mathbf{A}$  is), and at least one tree in this forest is infinite. Thus, by König’s lemma, the tree has an infinite path  $\pi$ . But  $\pi$  is a play in  $\mathbf{A}$  and  $\pi^\beta = \rho$ .

**Definition 4.3.** For  $\rho \in \text{Ply}(\mathbf{A}^\beta)$ , say  $\rho = B_0 a_0 B_1 a_1 \dots$ , define  $obs(\rho)$  to be the sequence  $obs(q_0) a_0 obs(q_1) a_1 \dots$

where  $q_i \in B_i$  for  $i \in \mathbb{N}_0$  (this is well defined since, by definition of the state set  $S^\beta$ , each  $B_i$  is a subset of a unique observation-set).

The classic belief-state construction transforms  $\langle \mathbf{A}, W \rangle$  into  $\langle \mathbf{A}^\beta, W \rangle$ . Example 3 shows that this transformation may not preserve the agent having a winning strategy if  $\mathbf{A}$  is infinite. We now define the generalized belief-state construction and the main technical theorem of this work.

**Definition 4.4.** Let  $\mathbf{G} = \langle \mathbf{A}, W \rangle$  be a GP game. Define  $\mathbf{G}^\beta = \langle \mathbf{A}^\beta, W, \text{Im}(\mathbf{A}) \rangle$ , a GP game of perfect information with restriction. The restriction  $\text{Im}(\mathbf{A}) \subseteq \text{Ply}(\mathbf{A}^\beta)$  is the image of  $\text{Ply}(\mathbf{A})$  under the map  $\pi \mapsto \pi^\beta$ .

**Theorem 4.5.** Let  $\mathbf{A}$  be a (possibly infinite) arena of imperfect information,  $\mathbf{A}^\beta$  the corresponding belief-state arena of perfect information, and  $\text{Im}(\mathbf{A}) \subseteq \text{Ply}(\mathbf{A}^\beta)$  the image of  $\mathbf{A}$ . Then, for every winning objective  $W$ , the agent has a winning observational-strategy in the GP game  $\mathbf{G} = \langle \mathbf{A}, W \rangle$  if and only if the agent has a winning strategy in the GP game  $\mathbf{G}^\beta = \langle \mathbf{A}^\beta, W, \text{Im}(\mathbf{A}) \rangle$ . Moreover, if  $\mathbf{A}$  is finitely-branching then  $\mathbf{G}^\beta = \langle \mathbf{A}^\beta, W \rangle$ .<sup>5</sup>

*Proof.* The second statement follows from the first statement and Remark 4.2. For the first statement, we first need some facts that immediately follow from Definition 4.1.

1.  $(h_1)^\beta = (h_2)^\beta$  if and only if  $\text{obs}(h_1) = \text{obs}(h_2)$ .
2. For every  $h \in \text{Hist}(\mathbf{A}^\beta)$  that is also a prefix of  $\pi^\beta$  there is a history  $h' \in \text{Hist}(\mathbf{A})$  that is also a prefix of  $\pi$  such that  $(h')^\beta = h$ . Also, for every  $h' \in \text{Hist}(\mathbf{A})$  that is also a prefix of  $\pi$  there is a history  $h \in \text{Hist}(\mathbf{A}^\beta)$  that is also a prefix of  $\pi^\beta$  such that  $(h')^\beta = h$ .

Second, there is a natural correspondence between observational strategies of  $\mathbf{A}$  and strategies of  $\mathbf{A}^\beta$ .

- If  $\sigma$  is a strategy in  $\mathbf{A}^\beta$  then define the strategy  $\omega(\sigma)$  of  $\mathbf{A}$  as mapping  $h \in \text{Hist}(\mathbf{A})$  to  $\sigma(h^\beta)$ . Now,  $\omega(\sigma)$  is observational by Fact 1. Also, if  $\pi$  is consistent with  $\omega(\sigma)$  then  $\pi^\beta$  is consistent with  $\sigma$ . Indeed, let  $h$  be a history that is also a prefix of  $\pi^\beta$ . We need to show that  $h\sigma(h)$  is a prefix of  $\pi^\beta$ . Suppose that  $\sigma(h) = a$ . Take prefix  $h'$  of  $\pi$  such that  $(h')^\beta = h$  (Fact 2). Then  $\omega(\sigma)(h') = \sigma((h')^\beta) = \sigma(h) = a$ . Since  $\pi$  is assumed consistent with  $\omega(\sigma)$ , conclude that  $h'a$  is a prefix of  $\pi$ . Thus  $ha$  is a prefix of  $\pi^\beta$ .
- If  $\sigma$  is an observational strategy in  $\mathbf{A}$  then define the strategy  $\kappa(\sigma)$  of  $\mathbf{A}^\beta$  as mapping  $h \in \text{Hist}(\mathbf{A}^\beta)$  to  $\sigma(h')$  where  $h'$  is any history such that  $h'^\beta = h$ . This is well-defined by (†) and the fact that  $\sigma$  is observational. Also, if  $\rho$  is consistent with  $\kappa(\sigma)$ , then every  $\pi$  with  $\pi^\beta = \rho$  (if there are any) is consistent with  $\sigma$ . Indeed, let  $h'$  be a history of  $\pi$  and take a prefix  $h$  of  $\pi^\beta$  such that  $(h')^\beta = h$  (Fact 2). Then  $\kappa(\sigma)(h) = \sigma(h')$ , call this action  $a$ . But  $\pi^\beta$  is assumed consistent with  $\kappa(\sigma)$ , and thus  $ha$  is a prefix of  $\pi^\beta$ . Thus  $h'a$  is a prefix of  $\pi$ .

We now put everything together and show that the agent has a winning observational-strategy in  $\mathbf{G}$  iff the agent has a winning strategy in  $\mathbf{G}^\beta$ .

<sup>5</sup>The case that  $\mathbf{A}$  is finite appears in [Raskin *et al.*, 2007].

Suppose  $\sigma$  is a winning strategy in  $\mathbf{G}^\beta$ . Let  $\pi \in \text{Ply}(\mathbf{A})$  be consistent with the observational strategy  $\omega(\sigma)$  of  $\mathbf{G}$ . Then  $\pi^\beta \in \text{Im}(\mathbf{A})$  is consistent with  $\sigma$ . But  $\sigma$  is assumed to be winning, thus  $\text{obs}(\pi^\beta) \in W$ . But  $\text{obs}(\pi) = \text{obs}(\pi^\beta)$ . Conclude that  $\omega(\sigma)$  is a winning strategy in  $\mathbf{G}$ .

Conversely, suppose  $\sigma$  is a winning strategy in  $\mathbf{G}$ . Let  $\rho \in \text{Im}(\mathbf{A})$  be consistent with the strategy  $\kappa(\sigma)$  of  $\mathbf{G}^\beta$ , and take  $\pi \in \text{Ply}(\mathbf{A})$  be such that  $\pi^\beta = \rho$  (such a  $\pi$  exists since we assumed  $\rho \in \text{Im}(\mathbf{A})$ ). Then  $\pi$  is consistent with  $\sigma$ . But  $\sigma$  is assumed to be winning, thus  $\text{obs}(\pi) \in W$ . But  $\text{obs}(\rho) = \text{obs}(\pi^\beta) = \text{obs}(\pi)$ . Conclude that  $\kappa(\sigma)$  is a winning strategy in  $\mathbf{G}^\beta$ .  $\square$

**Remark 4.6.** The proof of Theorem 4.5 actually shows how to transform strategies between the GP games, i.e.,  $\sigma \mapsto \omega(\sigma)$  and  $\sigma \mapsto \kappa(\sigma)$ , and moreover, these transformations are inverses of each other.

We end with our running example:

*Example 5 (Continued).* Solving  $\mathbf{G}_{\text{chop}}$  (Figure 1) is equivalent to solving the finite GP game  $\mathbf{G}_{\text{chop}}^\beta$  of perfect information, i.e.,  $\langle \mathbf{A}_{\text{chop}}^\beta, W, \text{Im}(\mathbf{A}_{\text{chop}}) \rangle$ , where the arena  $\mathbf{A}$  is shown in Figure 2. To solve this we should understand the structure of  $\text{Im}(\mathbf{A}_{\text{chop}})$ . It is not hard to see that a play  $\rho \in \text{Ply}(\mathbf{A}_{\text{chop}}^\beta)$  is in  $\text{Im}(\mathbf{A}_{\text{chop}})$  if and only if it contains only finitely many infixes of the form “UK look UP”. This property is expressible in LTL by the formula  $\neg G F [U K \wedge X \text{look} \wedge X X \text{UP}]$ . Thus we can apply the algorithm for solving finite games of perfect information with LTL objectives (see, e.g., [Pnueli and Rosner, 1989; de Alfaro *et al.*, 2001]) to solve  $\mathbf{G}_{\text{chop}}^\beta$ , and thus the original GP game  $\mathbf{G}_{\text{chop}}$ .

## 5 Application of the Construction

We now show how to use generalized-planning (GP) games and our generalized belief-state construction to obtain effective planning procedures for sophisticated problems. For the rest of this section we assume Obs is finite ( $\mathbf{A}$  may be infinite) so that we can consider LTL temporally extended goals over the alphabet Obs. For instance, LTL formulas specify persistent surveillance missions such as “get items from region A, drop items at region B, infinitely often, and always avoid region C”.

**Definition 5.1.** Let  $\varphi$  be an LTL formula over  $\text{Obs} \times \text{Ac}$ . For an arena  $\mathbf{A}$ , define  $[[\varphi]] = \{\pi \in \text{Ply}(\mathbf{A}) \mid \text{obs}(\pi) \models \varphi\}$ .

The following is immediate from Theorem 4.5 and the fact that solving finite LTL games of perfect information is decidable [Pnueli and Rosner, 1989; de Alfaro *et al.*, 2001]:

**Theorem 5.2.** Let  $\mathbf{G} = \langle \mathbf{A}, [[\varphi]] \rangle$  be a GP game with a finite arena (possibly obtained as the disjoint union of several arenas sharing the same observations), and  $\varphi$  be an LTL winning objective. Then solving  $\mathbf{G}$  can be reduced to solving the finite GP game  $\mathbf{G}^\beta = \langle \mathbf{A}^\beta, [[\varphi]] \rangle$  of perfect information, which is decidable.

Although we defined winning objectives to be observable, one may prefer general winning conditions, i.e.,  $W \subseteq S^\omega$ . In this case, for finite arenas there is a translation from parity-objectives to observable parity-objectives [Chatterjee

and Doyen, 2010]; moreover, for reachability objectives, a plan reaches a goal  $T \subseteq S$  iff it reaches a belief-state  $B \subseteq T$  [Bertoli *et al.*, 2006].

Next we look a case where the arena is actually infinite. Recently, the AI community has considered games generated by pushdown-automata [Murano and Perelli, 2015; Chen *et al.*, 2016]. However, the games considered are of perfect information and cannot express generalized-planning problems or planning under partial observability. In contrast, our techniques can solve these planning problems on pushdown domains assuming that the stack is not hidden (we remark that if the stack is hidden, then game-solving becomes undecidable [Azhar *et al.*, 2001]):

**Theorem 5.3.** *Let  $\mathbf{G} = \langle \mathbf{A}, [[\varphi]] \rangle$  be a GP game with a pushdown-arena with observable stack, and  $\varphi$  is an LTL formula. Then solving  $\mathbf{G}$  can be reduced to solving  $\mathbf{G}^\beta = \langle \mathbf{A}^\beta, [[\varphi]] \rangle$ , a GP game with pushdown-arena with perfect information, which is decidable.*

*Proof.* Let  $P$  be a pushdown-automaton with states  $Q$ , initial state  $q_0$ , finite input alphabet  $Ac$ , and finite stack alphabet  $\Gamma$ . We call elements of  $\Gamma^*$  stacks, and denote the empty stack by  $\epsilon$ . Also, fix an observation function on the states, i.e.,  $f : Q \rightarrow \text{Obs}$  for some set  $\text{Obs}$  (we do not introduce notation for the transition function of  $P$ ). A pushdown-arena  $\mathbf{A}_P = \langle S, I, Ac, \text{tr}, \text{Obs}, obs \rangle$  is generated by  $P$  as follows: the set of states  $S$  is the set of configurations of  $P$ , i.e., pairs  $(q, \gamma)$  where  $q \in Q$  and  $\gamma \in \Gamma^*$  is a stack-content of  $P$ ; the initial state of  $\mathbf{A}$  is the initial configuration, i.e.,  $I = \{(q_0, \epsilon)\}$ ; the transition function of  $\mathbf{A}$  is defined as  $\text{tr}((q, \gamma), a) = (q', \gamma')$  if  $P$  can move in one step from state  $q$  and stack content  $\gamma$  to state  $q'$  and stack content  $\gamma'$  by consuming the input letter  $a$ ; the observation function  $obs$  maps a configuration  $(q, \gamma)$  to  $f(q)$  (i.e., this formalizes the statement that the stack is observable). Observe now that: (1) the GP game  $\mathbf{A}$  is finitely-branching; (2) the GP game  $\mathbf{A}^\beta$  is generated by a pushdown automaton (its states are subsets of  $Q$ ). Thus we can apply Theorem 4.5 to reduce solving  $\mathbf{A}$ , a GP-game with imperfect information and pushdown arena, to  $\mathbf{A}^\beta$ , a GP-game with perfect information and pushdown arena. The latter is decidable [Walukiewicz, 2001].  $\square$

## 6 Related work in Formal Methods

Games of imperfect information on *finite* arenas have been studied extensively. Reachability winning-objectives were studied in [Reif, 1984] from a complexity point of view: certain games were shown to be universal in the sense that they are the hardest games of imperfect information, and optimal decision procedures were given. More generally,  $\omega$ -regular winning-objectives were studied in [Raskin *et al.*, 2007], and symbolic algorithms were given (also for the case of randomized strategies).

To solve (imperfect-information) games on infinite arenas one needs a finite-representation of the infinite arena. One canonical way to generate infinite arenas is by parametric means. In this line, [Jacobs and Bloem, 2014] study the synthesis problem for distributed architectures with a parametric number of finite-state components. They leverage re-

sults from the Formal Methods literature that say that for certain types of token-passing systems there is a cutoff [Emerson and Namjoshi, 1995], i.e., an upper bound on the number of components one needs to consider in order to synthesize a protocol for any number of components. Another way to generate infinite arenas is as configuration spaces of pushdown automata. These are important in analysis of software because they capture the flow of procedure calls and returns in reactive programs. Module-checking pushdown systems of imperfect information [Bozzelli *et al.*, 2010; Aminof *et al.*, 2013] can be thought of as games in which the environment plays non-deterministic strategies. Although undecidable, by not hiding the stack (cf. Theorem 5.3) decidability of module-checking is regained.

Finally, we note that synthesis of distributed systems has been studied in the Formal Methods literature using the techniques of games, starting with [Pnueli and Rosner, 1990]. Such problems can be cast as multi-player games of imperfect information, and logics such as ATL with knowledge can be used to reason about strategies in these games. However, even for three players, finite arenas, and reachability goals, the synthesis problem (and the corresponding model checking problem for ATL $K$ ) is undecidable [Dima and Tiplea, 2011].

## 7 Critical Evaluation and Conclusions

Although our technique for removing partial observability is sound and complete, it is, necessarily, not algorithmic: indeed, no algorithm can always remove partial observability from computable infinite domains and result in a solvable planning problem (e.g, one with a finite domain).<sup>6</sup>

The main avenue for future technical work is to establish natural classes of generalized-planning problems that can be solved algorithmically. We believe the methodology of this paper will be central to this endeavor. Indeed, as we showed in Section 5, we can identify  $\text{Im}(\mathbf{A})$  in a number of cases. We conjecture that one can do the same for all of the one-dimensional planning problems of [Hu and Levesque, 2010; Hu and De Giacomo, 2011].

The framework presented in this paper is non-probabilistic, but extending it with probabilities and utilities associated to agent choices [Kaelbling *et al.*, 1998; LaValle, 2006; Bonet and Geffner, 2009; Geffner and Bonet, 2013] is of great interest. In particular, POMDPs with temporally-extended winning objectives (e.g., LTL, Büchi, parity) have been studied for finite domains [Chatterjee *et al.*, 2010]. We leave for future work the problem of dealing with such POMDPs over infinite domains.

## Acknowledgments

We thank the reviewers for their constructive comments. This research was partially supported by the Sapienza project “Immersive Cognitive Environments”. Aniello Murano was supported in part by GNCS 2016 project: Logica, Automi e Giochi per Sistemi Auto-adattivi. Sasha Rubin is a Marie Curie fellow of the Istituto Nazionale di Alta Matematica.

<sup>6</sup>In fact, there is no algorithm solving games of perfect observation on computable domains with reachability objectives.

## References

- [Aminof *et al.*, 2013] B. Aminof, A. Legay, A. Murano, O. Serre, and M. Y. Vardi. Pushdown module checking with imperfect information. *Inf. Comput.*, 223, 2013.
- [Azhar *et al.*, 2001] S. Azhar, G. Peterson, and J. Reif. Lower bounds for multiplayer non-cooperative games of incomplete information. *J. Comp. Math. Appl.*, 41, 2001.
- [Bertoli and Pistore, 2004] P. Bertoli and M. Pistore. Planning with extended goals and partial observability. In *Proc. of ICAPS 2004*, 2004.
- [Bertoli *et al.*, 2006] P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Strong planning under partial observability. *Artif. Intell.*, 170(4-5), 2006.
- [Bonet and Geffner, 2009] B. Bonet and H. Geffner. Solving POMDPs: Rtdp-bel vs. point-based algorithms. In *Proc. of IJCAI 2009*, 2009.
- [Bonet *et al.*, 2009] B. Bonet, H. Palacios, and H. Geffner. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *Proc. of ICAPS 2009*, 2009.
- [Bozzelli *et al.*, 2010] L. Bozzelli, A. Murano, and A. Peron. Pushdown module checking. *FMSD*, 36(1), 2010.
- [Chatterjee and Doyen, 2010] K. Chatterjee and L. Doyen. The complexity of partial-observation parity games. In *Proc. of LPAR 2010*, 2010.
- [Chatterjee *et al.*, 2010] K. Chatterjee, L. Doyen, and T.A. Henzinger. Qualitative analysis of partially-observable markov decision processes. In *Proc. of MFCS*, 2010.
- [Chen *et al.*, 2016] T. Chen, F. Song, and Z. Wu. Global model checking on pushdown multi-agent systems. In *Proc. of AAAI 2016*, 2016.
- [de Alfaro *et al.*, 2001] L. de Alfaro, T. A. Henzinger, and R. Majumdar. From verification to control: Dynamic programs for omega-regular objectives. In *Proc. of LICS 2001*, 2001.
- [De Giacomo and Vardi, 1999] G. De Giacomo and M. Y. Vardi. Automata-theoretic approach to planning for temporally extended goals. In *Proc. of ECP 1999*, 1999.
- [Dima and Tiplea, 2011] C. Dima and F. L. Tiplea. Model-checking ATL under imperfect information and perfect recall semantics is undecidable. *CoR*, abs/1102.4225, 2011.
- [Emerson and Namjoshi, 1995] E. A. Emerson and K. S. Namjoshi. Reasoning about rings. In *Proc. of POPL 1995*, 1995.
- [Felli *et al.*, 2012] P. Felli, G. De Giacomo, and A. Lomuscio. Synthesizing agent protocols from LTL specifications against multiple partially-observable environments. In *Proc. of KR 2012*, 2012.
- [Geffner and Bonet, 2013] H. Geffner and B. Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool, 2013.
- [Goldman and Boddy, 1996] R. P. Goldman and M. S. Boddy. Expressive planning and explicit knowledge. In *Proc. of AIPS 1996*, 1996.
- [Hu and De Giacomo, 2011] Y. Hu and G. De Giacomo. Generalized planning: Synthesizing plans that work for multiple environments. In *Proc. of IJCAI 2011*, 2011.
- [Hu and Levesque, 2010] Y. Hu and H. J. Levesque. A correctness result for reasoning about one-dimensional planning problems. In *Proc. of KR 2010*, 2010.
- [Jacobs and Bloem, 2014] S. Jacobs and R. Bloem. Parameterized synthesis. *LMCS*, 10(1), 2014.
- [Kaelbling *et al.*, 1998] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101(1-2), 1998.
- [LaValle, 2006] S. M. LaValle. *Planning algorithms*. Cambridge, 2006.
- [Levesque, 1996] H. J. Levesque. What is planning in the presence of sensing. In *Proc. of AAAI 1996*, 1996.
- [Levesque, 2005] H. Levesque. Planning with loops. In *Proc. of IJCAI 2005*, 2005.
- [M. Ghallab and Traverso, 2008] D. Nau M. Ghallab and P. Traverso. *Automated Planning: Theory & Practice*. Elsevier, 2008.
- [Murano and Perelli, 2015] A. Murano and G. Perelli. Pushdown multi-agent system verification. In *Proc. of IJCAI 2015*, 2015.
- [Pnueli and Rosner, 1989] A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In *Proc. of ICALP 1989*, 1989.
- [Pnueli and Rosner, 1990] A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *Proc. of STOC 1990*, 1990.
- [Raskin *et al.*, 2007] J. Raskin, K. Chatterjee, L. Doyen, and T. A. Henzinger. Algorithms for omega-regular games with imperfect information. *LMCS*, 3(3), 2007.
- [Reif, 1984] J. H. Reif. The complexity of two-player games of incomplete information. *JCSS*, 29(2), 1984.
- [Rintanen, 2004] J. Rintanen. Complexity of planning with partial observability. In *Proc. of ICAPS 2004*, 2004.
- [Sardiña *et al.*, 2006] S. Sardiña, G. De Giacomo, Y. Lespérance, and H. J. Levesque. On the limits of planning over belief states under strict uncertainty. In *Proc. of KR 2016*, 2006.
- [Srivastava *et al.*, 2008] S. Srivastava, N. Immerman, and S. Zilberstein. Learning generalized plans using abstract counting. In *Proc. of AAAI 2008*, 2008.
- [Srivastava *et al.*, 2011] S. Srivastava, N. Immerman, and S. Zilberstein. A new representation and associated algorithms for generalized planning. *Artif. Intell.*, 175(2), 2011.
- [Srivastava *et al.*, 2015] S. Srivastava, S. Zilberstein, A. Gupta, P. Abbeel, and S. J. Russell. Tractability of planning with loops. In *Proc. of AAAI 2015*, 2015.
- [Walukiewicz, 2001] I. Walukiewicz. Pushdown processes: Games and model-checking. *Inf. Comput.*, 164(2), 2001.

## **4.11 Publication**

The rest of this page is intentionally left blank

# Verification of Multi-agent Systems with Imperfect Information and Public Actions

F. Belardinelli

Laboratoire IBISC, UVEVE – IRIT Toulouse  
belardinelli@ibisc.fr

A. Lomuscio

Department of Computing  
Imperial College London  
a.lomuscio@imperial.ac.uk

A. Murano and S. Rubin

DIETI  
Università degli Studi di Napoli  
murano@na.infn.it  
rubin@unina.it

## ABSTRACT

We analyse the verification problem for synchronous, perfect recall multi-agent systems with imperfect information against a specification language that includes strategic and epistemic operators. While the verification problem is undecidable, we show that if the agents' actions are public, then verification is 2EXPTIME-complete. To illustrate the formal framework we consider two epistemic and strategic puzzles with imperfect information and public actions: the muddy children puzzle and the classic game of battleships.

## 1. INTRODUCTION

Synchronous, perfect-recall multi-agent systems (MAS) are an important class of MAS that can be used to model a wide variety of scenarios including communication protocols, security protocols and games [8]. Reasoning about the knowledge and the strategic ability of agents in these systems remains of particular importance. Traditionally, epistemic logic [8] has been used to express the states of knowledge of the agents, whereas ATL has provided a basis for the agents' strategic abilities [1]. ATL and epistemic logic have been combined in a number of ways to obtain specification languages capable of expressing both concepts (see below). A popular method for establishing properties of MAS is verification via model checking [4].

However, verifying synchronous, perfect recall MAS under incomplete information against specifications in ATL is undecidable [1, 6] (hence it remains undecidable when epistemic modalities are added); it is therefore of interest to identify cases in which reasoning about MAS is decidable. These restrictions typically take three forms: restricting the syntax of the logic (e.g., by removing strategic abilities and consider, instead,  $LTL_{\mathbb{K}}$ , the extension of LTL with individual-knowledge operators, as in [32]), restricting the semantics (e.g., by requiring strategy quantifiers to vary over memoryless-strategies [31]), or by restricting the class of MAS under consideration. In this paper we follow the third option.

**Contribution.** We identify a class of imperfect-information concurrent game structures (iCGS) that we call public-action iCGS (PA-iCGS). In contrast to general iCGS [6], we prove

that model-checking the full logic  $ATL_{\mathbb{K},C,D}^*$  on PA-iCGS is decidable, specifically 2EXPTIME-complete. Thus, the joint complexity of model-checking is the same as that of  $ATL^*$  with perfect information [1]. Moreover, we show that the class models MAS in which agents have imperfect information, synchronous perfect recall, and whose actions are public, i.e., all actions are visible to all agents. As we explain, the class PA-iCGS captures games of imperfect information in which the agents have uncertainty about the initial configuration but all moves are observable to all agents. This has applications to, among others, games (e.g., Bridge, Poker, Battleships, etc.), fair division protocols (e.g., classic cake cutting algorithms), selected broadcast protocols [33], blackboard systems in which a public database is read and written by agents [26], auctions and auction-based mechanisms [7].

The rest of the paper is organised as follows. In the remainder of this section we discuss related work. In Section 2 we define iCGS with public actions and the logic  $ATL_{\mathbb{K},C,D}^*$ , that we will use as specification language and illustrate the formalism. In Section 3 we present the main result of the paper, i.e., we show the decidability of the verification problem, by means of an automata-theoretic approach, and analyse the resulting complexity. In Section 4 we compare our approach to that of Broadcast Environments [33]. We conclude in Section 5.

**Related Work.** In order to reason formally about multi-agent systems, temporal logics such as LTL, CTL, CTL\* have been extended with strategy quantifiers [1] and epistemic modalities [14]. The extended syntax has been combined with a number of different assumptions on the underlying MAS: perfect vs. imperfect information, perfect vs. imperfect recall, state-based vs. history-based semantics [1, 14, 10, 31, 15, 5, 11, 25].

Assuming imperfect information and perfect recall, as we do in this paper, often results in intractable model-checking. For instance, the model-checking problem for ATL in this setting is undecidable [6], as is the model-checking problem for the extension  $LTL_{\mathbb{K},C}$  of LTL with epistemic operators, including common knowledge [32]. Given this difficulty, finding decidable or tractable fragments remains of interest.

As expected, restricting the logic lowers the complexity. We list some notable examples: the model-checking problem for  $LTL_{\mathbb{K}}$  with only individual knowledge is non-elementary complete [32], model-checking ATL with only “communicating coalitions” (i.e., coalitions use their distributed knowledge instead of their individual knowledge) is decidable and non-elementary [5, 11]; and, model-checking ATL in which all coalitions operate with a single indistinguishability relation

tion reduces ATL to its singleton-coalition fragment [17].

Also, restricting the class of structures (iCGS) over which these languages are interpreted lowers the complexity. Such restrictions typically take one of two forms: i) on the observation or information sets of the agents; ii) on the architectures that govern communication. Notable examples of i) may require that: all agents have the same observation sets [21]; that the information sets form a hierarchy [27], or that, over time, they infinitely often form a hierarchy [2]. A notable example of ii) are characterisations of the architectures for which distributed synthesis is decidable [9, 30], thus generalising earlier results on linear architectures [27, 18].

More closely related to the present contribution are broadcast environments (which restrict the underlying iCGS) and that can capture epistemic puzzles and games of imperfect information such as Bridge [33]. The most relevant result for broadcast environments is that synthesis of linear-temporal logic with individual-knowledge operators is decidable [33]. Not only can our language express this synthesis problem, but it is strictly more expressive, as it can alternate strategic quantifiers mentioning overlapping coalitions. A detailed discussion of the significance of [33] is given in Section 4.

Actions that constitute public announcements have been studied in depth (Dynamic Epistemic Logic, Public Announcement Logic, epistemic protocols [34]). However, this line of research differs semantically and syntactically from our work. In particular, in these works modal operators are model transformers, and coalitions are not explicitly named in the language.

## 2. GAMES WITH PUBLIC ACTIONS AND STRATEGIC-EPISTEMIC LOGIC

In this section we define the game model and the logic. The model is the subclass of imperfect information concurrent game structures (iCGS) that only have public actions (PA-iCGS). The logic is  $\text{ATL}_{\mathbb{K}, \mathbb{C}, \mathbb{D}}^*$ , an extension of Alternating Time Temporal Logic (ATL\*) which includes strategic operators ( $\langle\!\langle A \rangle\!\rangle$  for  $A \subseteq Ag$ ) as well as epistemic operators for individual-knowledge ( $\mathbb{K}_a$  for  $a \in Ag$ ), common-knowledge ( $\mathbb{C}_A$  for  $A \subseteq Ag$ ), and distributed-knowledge ( $\mathbb{D}_A$  for  $A \subseteq Ag$ ).

**Notation.** For an infinite or non-empty finite sequence  $u \in X^\omega \cup X^+$  write  $u_i$  for the  $i$ th element of  $u$ , i.e.,  $u = u_0 u_1 \dots$ . The empty sequence is denoted  $\epsilon$ . The length of a finite sequence  $u \in X^*$  is denoted  $|u|$ , its last (resp. first) element is denoted  $last(u)$  (resp.  $first(u)$ ). Note that  $last(\epsilon) = first(\epsilon) = \epsilon$ . For  $i < |u|$  write  $u_{\leq i}$  for the prefix  $u_0 \dots u_i$ . For a vector  $v \in \prod_i X_i$  we denote the  $i$ -th co-ordinate of  $v$  by  $v(i)$ . In particular, for  $F \in \prod_i (X_i)^Y$  we may write  $F(i) \in X^Y$  and  $F(i)(y) \in X_i$ .

### 2.1 iCGS with only Public Actions

We begin with the standard definition of imperfect information concurrent game structures [3, 6].

**DEFINITION 1** (iCGS). *An imperfect information concurrent game structure (iCGS) is a tuple*

$$S = \langle Ag, AP, \{Act_a\}_{a \in Ag}, S, S_0, \delta, \{\sim_a\}_{a \in Ag}, \lambda \rangle$$

where:

- $Ag$  is the finite non-empty set of agent names;

- $AP$  is the finite non-empty set of atomic propositions;
- $Act_a$  is the finite non-empty set of actions for  $a \in Ag$ ;
- $S$  is the finite non-empty set of states;
- $S_0 \subseteq S$  is the non-empty set of initial states;
- $\delta : S \times ACT \rightarrow S$  is the transition function, where the set  $ACT$  of joint-actions is the set of all functions  $J : Ag \rightarrow \bigcup_a Act_a$  such that  $J(a) \in Act_a$ . The transition function assigns to every state  $s$  and joint-action  $J$ , a successor state  $\delta(s, J)$ ;
- $\sim_a \subseteq S^2$  is the indistinguishability relation for agent  $a$ , which is an equivalence relation; the equivalence class  $[s]_a$  of  $s \in S$  under  $\sim_a$  is called the observation set of agent  $a$ ;
- $\lambda : AP \rightarrow 2^S$  is the labeling function that assigns to each atom  $p$  the set of states  $\lambda(p)$  in which  $p$  holds.

Perfect-information is treated as a special case:

**DEFINITION 2** (PERFECT-INFORMATION). *A concurrent game structure (CGS) is an iCGS for which  $\sim_a = \{(s, s) : s \in S\}$  for all  $a \in Ag$ .*

We now give a brief and to-the-point definition of what it means for an iCGS to only have public actions, i.e., all actions are visible to all agents. This determines a subclass of iCGS that we call PA-iCGS.

**DEFINITION 3** (PA-iCGS). *An iCGS only has public actions if for every agent  $a \in Ag$ , states  $s, s' \in S$ , and joint actions  $J, J' \in ACT$ , if  $J \neq J'$  and  $s \sim_a s'$  then  $\delta(s, J) \not\sim_a \delta(s', J')$ . We write PA-iCGS for the class of iCGS that only have public actions.*

This definition says that if an agent  $a$  cannot distinguish between two states, but different joint actions are performed in each of these states (because, for instance, some other agent can distinguish them), then the agent can distinguish between the resulting successor states.

One way to generate an iCGS only having public actions is to ensure that i) the state records the last joint-action played, thus  $S$  is of the form  $T \times (ACT \cup \{\epsilon\})$ , where  $\epsilon$  refers to the situation that no actions have yet been played, and ii) the indistinguishability relations  $\sim_a$  satisfy that if  $(t, J) \sim_a (t', J')$  then  $J = J'$ . Similar conditions have been considered in the literature, e.g., *recording contexts* in [8].

In the remainder of this section we define what it means for an agent to have *synchronous perfect-recall* [8].

**Synchronous perfect-recall under imperfect information.** A path in  $S$  is a non-empty infinite or finite sequence  $\pi_0 \pi_1 \dots \in S^\omega \cup S^+$  such that for all  $i$  there exists a joint-action  $J(i) \in ACT$  such that  $\pi_{i+1} \in \delta(\pi_i, J(i))$ . Paths that start with initial states are called *histories* if they are finite and *computations* if they are infinite. The set of computations in  $S$  is written  $\text{comp}(S)$ , and the set of computations in  $S$  that start with history  $h$  is written  $\text{comp}(S, h)$ . We define  $\text{hist}(S)$  and  $\text{hist}(S, h)$  similarly.

We use the following notation: if  $\sim$  is a binary relation on  $S$  we define the extension of  $\sim$  to histories as the binary relation  $\equiv$  on  $\text{hist}(S)$  defined by  $h \equiv h'$  iff  $|h| = |h'|$  (i.e., synchronicity) and  $h_j \sim h'_j$  for all  $j \leq |h|$  (i.e., perfect recall).

We give three particular instantiations. If  $\sim_a$  is the indistinguishability relation for agent  $a$ , then we say that two histories  $h, h'$  are *indistinguishable to agent a*, if  $h \equiv_a h'$ . For  $A \subseteq Ag$ , let  $\sim_A^C = (\cup_{a \in A} \sim_a)^*$ , where  $*$  denotes the reflexive transitive closure (wrt. composition of relations), and its extension to histories is denoted  $\equiv_A^C$ . For  $A \subseteq Ag$ , let  $\sim_A^D = \cap_{a \in A} \sim_a$ , and its extension to histories is denoted  $\equiv_A^D$ . **Strategies.** A *deterministic memoryfull strategy*, or simply a *strategy*, for agent  $a$  is a function  $\sigma_a : \text{hist}(S) \rightarrow \text{Act}_a$ . A strategy  $\sigma_a$  is *uniform* if for all  $h \equiv_a h'$ , we have  $\sigma(h) = \sigma(h')$ . The set of uniform strategies is denoted  $\Sigma(S)$ . All strategies in the rest of the paper are uniform (although sometimes we will stress this fact and write “uniform strategy”).

For  $A \subseteq Ag$ , let  $\sigma_A : A \rightarrow \Sigma(S)$  denote a function that associates a uniform strategy  $\sigma_a$  with each agent  $a \in A$ . We write  $\sigma_A(a) = \sigma_a$ , and call  $\sigma_A$  a *joint strategy*.

For  $h \in \text{hist}(S)$  write  $\text{out}(S, h, \sigma_A)$ , called the *outcomes of  $\sigma_A$  from  $h$* , for the set of computations  $\pi \in \text{comp}(S, h)$  such that  $\pi$  is consistent with  $\sigma_A$ , that is,  $\pi \in \text{out}(S, h, \sigma_A)$  iff (i)  $\pi_{\leq|h|-1} = h$ ; (ii) for every position  $i \geq |h|$ , there exists a joint-action  $J_i \in \text{ACT}$  such that  $\pi_{i+1} \in \delta(\pi_i, J_i)$ , and for every  $a \in A$ ,  $J_i(a) = \sigma_A(a)(\pi_{\leq i})$ . We may drop  $S$  and write simply  $\text{out}(h, \sigma_A)$ . Notice that, if  $A = \emptyset$ , then  $\text{out}(h, \sigma_A)$  is the set of all paths starting with  $h$  (this is because  $\sigma_A$  is the empty function and (ii) above places no additional restrictions on the computations).

## 2.2 The Logic $\text{ATL}_{K,C,D}^*$

In this section we define the logic  $\text{ATL}_{K,C,D}^*$ . Its syntax has been called  $\text{ATEL}^*$  (cf. [14]), and we interpret it on iCGS with history-based semantics and imperfect information.

**Syntax.** Fix a finite set of *atomic propositions (atoms)*  $AP$  and a finite set of *agents*  $Ag$ . The *history ( $\varphi$ ) and path ( $\psi$ ) formulas over  $AP$  and  $Ag$*  are built using the following grammar:

$$\begin{aligned}\varphi &::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbb{K}_a \varphi \mid \mathbb{C}_A \varphi \mid \langle\langle A \rangle\rangle \psi \\ \psi &::= \varphi \mid \neg\psi \mid \psi \wedge \psi \mid \mathbf{X} \psi \mid \psi \mathbf{U} \psi\end{aligned}$$

where  $p \in AP$ ,  $a \in Ag$ , and  $A \subseteq Ag$ .

The class of  $\text{ATL}_{K,C,D}^*$  *formulas* is the set of history formulas generated by the grammar. The *temporal operators* are  $\mathbf{X}$  (read “next”) and  $\mathbf{U}$  (read “until”). The *strategy quantifier* is  $\langle\langle A \rangle\rangle$  (“the agents in  $A$  can enforce  $\psi$ ”), and the *epistemic operators* are  $\mathbb{K}_a$  (“agent  $a$  knows that”),  $\mathbb{C}_A$  (“it is common-knowledge amongst  $A$  that”), and  $\mathbb{D}_A$  (“the agents in  $A$  distributively know that”).

**Semantics.** Fix an iCGS  $S$ . We simultaneously define, by induction on the formulas,  $(S, h) \models \varphi$  where  $h \in \text{hist}(S)$  and  $\varphi$  is a history formula, and  $(S, \pi, m) \models \psi$  where  $\pi \in \text{comp}(S)$ ,  $m \geq 0$ , and  $\psi$  is a path formula:

$$\begin{aligned}(S, h) \models p &\quad \text{iff } \text{last}(h) \in \lambda(p), \text{ for } p \in AP. \\ (S, h) \models \neg\varphi &\quad \text{iff } (S, h) \not\models \varphi. \\ (S, h) \models \varphi_1 \wedge \varphi_2 &\quad \text{iff } (S, h) \models \varphi_i \text{ for } i \in \{1, 2\}. \\ (S, h) \models \langle\langle A \rangle\rangle \psi &\quad \text{iff for some joint strategy } \sigma_A \in \Sigma(S), \\ &\quad \quad (S, \pi, |h|-1) \models \psi \text{ for all } \pi \in \text{out}(h, \sigma_A). \\ (S, h) \models \mathbb{K}_a \varphi &\quad \text{iff for every history } h' \in \text{hist}(S), \\ &\quad \quad h' \equiv_a h \text{ implies } (S, h') \models \varphi. \\ (S, h) \models \mathbb{C}_A \varphi &\quad \text{iff for every history } h' \in \text{hist}(S), \\ &\quad \quad h' \equiv_A^C h \text{ implies } (S, h') \models \varphi. \\ (S, h) \models \mathbb{D}_A \varphi &\quad \text{iff for every history } h' \in \text{hist}(S), \\ &\quad \quad h' \equiv_A^D h \text{ implies } (S, h') \models \varphi.\end{aligned}$$

$$\begin{aligned}(S, \pi, m) \models \varphi &\quad \text{iff } (S, \pi_{\leq m}) \models \varphi, \text{ for } \varphi \text{ a history formula.} \\ (S, \pi, m) \models \neg\psi &\quad \text{iff } (S, \pi, m) \not\models \psi. \\ (S, \pi, m) \models \psi_1 \wedge \psi_2 &\quad \text{iff } (S, \pi, m) \models \psi_i \text{ for } i \in \{1, 2\}. \\ (S, \pi, m) \models \mathbf{X} \psi &\quad \text{iff } (S, \pi, m+1) \models \psi. \\ (S, \pi, m) \models \psi_1 \mathbf{U} \psi_2 &\quad \text{iff for some } j \geq m, (S, \pi, j) \models \psi_2, \text{ and} \\ &\quad \quad \text{for all } k \text{ with } m \leq k < j, \text{ we have} \\ &\quad \quad (S, \pi, k) \models \psi_1.\end{aligned}$$

For a history formula  $\varphi$ , write  $S \models \varphi$  to mean that  $(S, s) \models \varphi$  for every  $s \in S_0$ .

We isolate some important fragments.

1. The fragment  $\text{ATL}_{K,C,D}$  consists of history formulas  $\varphi$  defined by the grammar above, except with the following path formulas:  $\psi ::= \mathbf{X} \varphi \mid \varphi \mathbf{U} \varphi$
2. The fragment  $\text{ATL}$  (resp.  $\text{ATL}^*$ ) consists of formulas of  $\text{ATL}_{K,C,D}$  (resp.  $\text{ATL}_{K,C,D}^*$ ) that do not mention epistemic operators.
3. The CTL operator  $\mathbf{E}$  (resp.  $\mathbf{A}$ ) is definable in  $\text{ATL}^*$  by  $[[\emptyset]]$  (resp.  $\langle\langle \emptyset \rangle\rangle$ ). In particular,  $\text{CTL}_{K,C,D}^*$  is a syntactic fragment of  $\text{ATL}_{K,C,D}^*$ . The fragment of  $\text{CTL}_{K,C,D}^*$  consisting of formulas of the form  $\mathbf{A} \psi$ , where  $\psi$  is a path formula, is denoted  $\text{LTL}_{K,C,D}$ . Finally,  $\text{LTL}$  is the fragment of  $\text{LTL}_{K,C,D}$  that does not mention epistemic operators.

**REMARK 1.** *The definition of the semantics of  $\langle\langle A \rangle\rangle \psi$  is the “objective” semantics of  $\langle\langle A \rangle\rangle$ , and captures the idea that a designer is reasoning about the existence of strategies. On the other hand, “subjective” semantics capture the idea that agents themselves are reasoning about the existence of strategies [31]. In Section 3.1 we define subjective semantics and extend our main result to deal with these.*

**Model Checking.** We state the main decision problem of this work.

**DEFINITION 4 (MODEL CHECKING).** Let  $\mathcal{C}$  be a class of iCGS and  $\mathcal{F}$  a sublanguage of  $\text{ATL}_{K,C,D}^*$ . Model checking  $\mathcal{C}$  against  $\mathcal{F}$  specifications is the following decision problem: given  $S \in \mathcal{C}$  and  $\varphi \in \mathcal{F}$  as input, decide whether  $S \models \varphi$ .

Model checking is undecidable in general. Actually, it is undecidable even if  $\mathcal{C}$  consists of all iCGS with  $|Ag| = 3$  and  $\mathcal{F}$  consists of the single formula  $\langle\langle \{1, 2\} \rangle\rangle G p$ , see [6]. In Section 3 we prove that model checking PA-iCGS against  $\text{ATL}_{K,C,D}^*$  specifications is decidable.

## 2.3 Examples

We illustrate this definition with two scenarios: the epistemic puzzle of the muddy children [8], and a generalisation of the game Battleships to multiple players. We model (or sketch) the scenario as an iCGS only having public actions, and supply representative formulas of  $\text{ATL}_{K,C,D}^*$ .

**Muddy Children.** We express this classic puzzle, as presented, e.g., in [8], in slightly different terms. There are  $n$  children,  $k$  of them with mud on their foreheads. Each child can see the forehead (and thus the muddy state) of all the other children, but not their own. Then the father enters the scene. The father can see the foreheads of all the muddy children. Each person can only make truthful statements about what they know.

The classic question is: what one statement can the father make that will lead each muddy child to learn that she

is indeed muddy. The answer is that the father declares, assuming that  $k \geq 1$ , that “at least one of you is muddy”.

There are a number of modeling decisions one can make to formalise this scenario, e.g., we must decide on the exact set of actions. Let  $Ag = \{F\} \cup \{1, 2, \dots, n\}$ . We give the father two actions, i.e.,  $Act_F = \{\exists m, \neg \exists m\}$ . We give the  $i$ th child two actions, i.e.,  $Act_i = \{\text{know\_muddy}, \neg \text{know\_muddy}\}$ . The set of states is  $S = \{0, 1\}^n \times (\text{ACT} \cup \{\epsilon\})$ . If the current state is  $(v, J)$  then  $v_i = 1$  (resp.  $v_i = 0$ ) means that the  $i$ th child is (resp. is not) muddy, and  $J$  is the most recent joint action ( $\epsilon$  means that no joint action has yet taken place). The set of initial states is  $S_0 = \{0, 1\}^n \times \{\epsilon\}$ . The transition relation simply updates the last joint action:  $\delta((v, J), J') = (v, J')$ . The father is perfectly informed so  $\sim_F = \{(s, s) : s \in S\}$ , and each child only does not see herself, so  $\sim_i = \{((v, J), (v', J')) : \bigwedge_{j \neq i} v_j = v'_j, J = J'\}$ . Finally, the atoms are  $AP = \{m_i : i \leq n\} \cup \{\alpha_a : \alpha \in Act_a, a \in Ag\}$ . Finally, define  $\lambda(m_i) = \{(s, J) : s_i = 1\}$  and  $\lambda(\alpha_a) = \{(s, J) : J(a) = \alpha\}$ . So  $m_i$  means that the  $i$ th child is muddy, and  $\alpha_a$  means that agent  $a$ 's last action was  $\alpha \in Act_a$ . It is straightforward to check that we have defined an iCGS only having public actions.

Define the shorthand formula  $\text{Kw}_a \phi$ , read “agent  $a$  knows whether  $\phi$ ”, by the formula  $\text{K}_a \phi \vee \text{K}_a \neg \phi$ . Consider the formula:

$$\langle\langle \{f, 1, \dots, n\} \rangle\rangle [(\bigwedge_{i \leq n} G(X \alpha_a \rightarrow \widehat{\alpha_a})) \wedge F(\bigwedge_{i \leq n} \text{Kw}_i m_i)]$$

where the first conjunction is over  $a \in Ag, \alpha \in Act_a$ , and  $\widehat{\alpha_a}$  is a formula representing the intended meaning of  $\alpha_a$ , e.g., if  $\alpha_a = \text{know\_muddy}$  then  $\widehat{\alpha_a} = \text{Kw}_a m_a$ . This expresses that all the agents have a truthful strategy so that eventually each child will know whether or not she is muddy.

The reader might wonder what would happen if we don't explicitly express that the actions are truthful. Consider the following formula that says that the father has a strategy such that eventually the children know their muddy state:

$$\langle\langle \{f\} \rangle\rangle F \bigwedge_{i \leq n} \text{Kw}_i m_i$$

This formula is true. Indeed, a strategy for the father is to play action  $\exists m$  at the  $i$ th step iff the  $i$ th child is muddy.

In the next scenario, what agents see changes over time (unlike in the muddy children scenario). In both scenarios, what agents hear gets updated over time.

**Battleships.** We consider a game of battleships with three players. Each player has a  $10 \times 10$ -board with numeric coordinates from bottom-left  $(0, 0)$  to top-right  $(9, 9)$ . Initially, each player can only see her own board. On her board each player displays her battleships: one carrier of size 5, two battleships of size 4, three cruisers of size 3, four submarines also of size 3, and five destroyers of size 2. We assume that ships are displayed either horizontally or vertically. As is standard, overlapping is not allowed.

Here we consider a synchronous version of battleships, structured in 2-phase rounds, where in phase 1 every player broadcasts the name  $p$  of a player and cell  $(n, m)$  of  $p$ 's board to attack. Then, in phase 2, the players truthfully state whether they have been hit or missed and the boards are updated accordingly. A player is eliminated once all her ships have been destroyed. The last player standing, if there is one, wins the game. The relevant atoms in this game are

$win_i$  and  $lose_i$  which state whether player  $i$  has won or lost. It is a routine exercise to build an iCGS only having public actions from this description. We only mention that the assumption that players are truthful can be built in to the iCGS, i.e., by limiting the available actions a player has in state 2.

Now consider the formula:

$$\langle\langle \{1, 2\} \rangle\rangle F \langle\langle \{1, 3\} \rangle\rangle F (lose_2 \vee \langle\langle \{1\} \rangle\rangle F win_1)$$

This expresses that player 1 can collude with each of her enemies, in order to weaken player 2 or win the game.

### 3. DECIDABILITY OF PA-iCGS

In this section we prove that model checking PA-iCGS against  $\text{ATL}_{\mathbb{K}, \mathbb{C}, \mathbb{D}}^*$  specifications is decidable. This should be contrast with the fact that model checking arbitrary iCGS against  $\text{ATL}$  specifications is undecidable [6].

**THEOREM 1.** *Model checking PA-iCGS against  $\text{ATL}_{\mathbb{K}, \mathbb{C}, \mathbb{D}}^*$  specifications is 2EXPTIME-complete.*

The bulk of this section is devoted to proving decidability. We then establish the complexity, and discuss further extensions of the result. Before giving the proof, we introduce an encoding  $\mu$  of histories.

**DEFINITION 5.** *Let  $\mathbf{S}$  be an iCGS. Let  $\mu : S_0 \times \text{ACT}^* \rightarrow \text{hist}(\mathbf{S})$  denote the function mapping  $(s_0, u)$  to the history starting at the initial state  $s_0$  that results from the sequence of joint actions  $u \in \text{ACT}^*$ . That is,  $\mu(s_0, u)$  is the history  $h$  such that  $h_0 = s_0$ ,  $h_j = \delta(h_{j-1}, u_{j-1})$  for  $1 \leq j \leq |u|$ .*

For PA-iCGS, the encoding is actually a bijection:

**REMARK 2.** *Let  $\mathbf{S}$  be a PA-iCGS. Since each  $\sim_a$  is reflexive,  $\delta(s, \cdot) : \text{ACT} \rightarrow S$  is injective for every  $s \in S$ . Thus,  $\mu : S_0 \times \text{ACT}^* \rightarrow \text{hist}(\mathbf{S})$  is a bijection. In particular, for every  $h \in \text{hist}(\mathbf{S})$  and  $s \in S_0$  there exists a unique  $u \in \text{ACT}^*$  such that  $\mu(s, u) = h$ . This bijection allows us to encode histories of  $\mathbf{S}$  by (unique) elements of  $S_0 \times \text{ACT}^*$ .*

An immediate consequence of having only public actions, but one that forms the foundation of our decidability proof, is that the moment different joint actions are taken, two histories become distinguishable.

**LEMMA 1.** *Let  $\mathbf{S}$  be a PA-iCGS. For all  $a \in Ag$ ,  $u, u' \in \text{ACT}^*$  and  $s, s' \in S_0$ , if  $\mu(s, u) \equiv_a \mu(s', u')$  then  $u = u'$ .*

**PROOF.** If  $\mu(s, u) \equiv_a \mu(s', u')$  then  $|u| = |u'|$  and, for all  $0 \leq j \leq |u|$ ,  $\mu(s, u)_j \sim_a \mu(s', u')_j$ . By the definition of having only public actions,  $u_j = u'_j$  for all  $j < |u|$ .  $\square$

We prove Theorem 1 in the rest of this section. We use an automata-based marking algorithm. Such algorithms have been successfully applied to a number of logics, including  $\text{CTL}^*$  [19] and  $\text{ATL}^*$  [1] in the perfect information setting.

**Automata theory.** Since our proof uses an automata-theoretic approach we now fix notations of word and tree automata. We remark that we only make use of standard properties of automata operating on finite words, infinite words, and infinite trees [20].

A *deterministic finite-word automaton* (DFW) is a tuple  $M = (\Sigma, S, s_0, \rho, F)$  where  $\Sigma$  is the *input alphabet*,  $S$  is the finite set of *states*,  $s_0 \in S$  the *initial state*,  $\rho : S \times \Sigma \rightarrow S$

the deterministic transition function, and  $F \subseteq S$  the set of final states. The run of  $M$  on  $u \in \Sigma^*$  is the finite sequence  $s_0 s_1 \cdots s_{|u|}$  where  $\rho(s_i, u_i) = s_{i+1}$  for all  $i < |u|$ . The automaton accepts a word  $u \in \Sigma^*$  iff the run of  $M$  on  $u$  ends in a final state. A DFW is empty if it accepts no word. A set of strings  $X \subseteq \Sigma^*$  is called regular if there is a DFW  $M$  that accepts  $u \in \Sigma^*$  iff  $u \in X$ .

We also make use of automata operating on infinite words  $\alpha \in \Sigma^\omega$ : a deterministic parity word automata (DPW) is a tuple  $P = (\Sigma, S, s_0, \rho, c)$  where all components are as for DFW except that  $c : S \rightarrow \mathbb{Z}$  is the colouring function. The run of  $P$  on  $\alpha \in \Sigma^\omega$  is the infinite sequence  $s_0 s_1 \cdots$  such that  $\rho(s_i, \alpha_i) = s_{i+1}$  for all  $i$ . The automaton accepts a word  $\alpha$  iff the smallest color  $k$  for which there are infinitely many  $i$  with  $c(s_i) = k$  is even (where  $s_0 s_1 \cdots$  is the run of  $M$  on  $\alpha$ ).

We also make use of automata operating on trees. A deterministic parity tree automata (DPW) is a tuple  $T = (\Sigma, D, S, s_0, \rho, c)$  where all components are as for a DPW except that  $D$  is the finite set of directions, and  $\rho : S \times \Sigma \rightarrow S^D$ . The automaton operates on  $\Sigma$ -labelled  $D$ -ary branching trees, i.e., functions  $f : D^* \rightarrow \Sigma$ . A branch of  $t$  is an infinite sequence  $\beta \in D^\omega$ . The run of  $T$  in input  $t$  is the  $Q$ -labeled  $D$ -ary branching tree  $g : D^* \rightarrow Q$  such that  $g(\epsilon) = s_0$  and  $g(u\beta) = \rho(g(u), t(u))$ . The automaton accepts the tree  $f$  iff for every  $\beta \in D^\omega$  (called a branch), the smallest color  $k$  for which there are infinitely  $i$  with  $g(\beta_i) = k$  is even.

The classes of DFW and DPW are effectively closed under the Boolean operations (complementation and intersection). Also, DFW, DPW and DPT can be effectively tested for emptiness. Finally, we make use of the following important fact connecting linear temporal logic with automata:

**PROPOSITION 1** ([35, 29]). *Every LTL formula  $\psi$  over atoms AP can be effectively converted into a DPW  $P_\psi$  with input alphabet  $2^{AP}$  that accepts a word  $\alpha \in 2^{AP}$  iff  $\alpha \models \psi$ . Moreover, the DPW has double-exponentially many states and single-exponentially many colours.*

**Proof outline.** We proceed by induction on the formula  $\varphi$  to be checked. We build a DFW that accepts all encodings of histories  $h$  such that  $(S, h) \models \varphi$ . Precisely, we build a DFW  $M_\varphi^s$  that accepts a sequence of joint actions  $u \in \text{ACT}^*$  iff  $(S, \mu(s, u)) \models \varphi$ . The atomic case is immediate, and the Boolean cases follow from the effective closure of DFW under complementation and intersection. The  $\varphi = \mathbb{K}_a \varphi'$  case is done by simulating the DFW  $M_{\varphi'}^t$  for  $t \in S_0$ , and recording whether or not  $\mu(s, u) \sim_a \mu(t, u)$ ; the other knowledge operators are similar. The strategic operator  $\varphi = \langle\!\langle A \rangle\!\rangle \psi$  is done as follows: first we show that we can assume  $\psi$  is an LTL formula, and then we build a DPW for the formula  $\psi$ ; we build the DFW that simulates the DPW, and when the input ends we use a tree automaton to decide if there is a joint strategy that ensures that the DPW accepts all computations consistent with that joint strategy.

**Generalisation of the labeling function.** We first generalise the labeling function of iCGS so that atoms are regular sets of histories (instead of state labelings).

**DEFINITION 6.** A generalised iCGS is a tuple

$$S = \langle Ag, AP, \{Act_a\}_{a \in Ag}, S, S_0, \delta, \{\sim_a\}_{a \in Ag}, \Lambda \rangle$$

where all entries are as for iCGS, except that  $\lambda : AP \rightarrow 2^S$  is replaced by a function  $\Lambda : AP \rightarrow 2^{\text{hist}(S)}$  such that  $\Lambda(p) \subseteq$

$\text{hist}(S)$  is a regular set of histories, i.e., there exists a DFW over the alphabet  $S$  accepting  $h \in \text{hist}(S)$  iff  $h \in \Lambda(p)$ .

Then, we redefine the atomic case of the semantics of  $\text{ATL}_{\mathbb{K}, \mathbb{C}, \mathbb{D}}^*$ :  $(S, h) \models p$  iff  $h \in \Lambda(p)$ . It is immediate that generalised iCGS are indeed more general than iCGS:

**LEMMA 2.** Let  $S$  be an iCGS with labeling function  $\lambda$ . The generalised iCGS  $S'$  with labeling  $\Lambda(p) = \{h \in \text{hist}(S) : \text{last}(h) \in \lambda(p)\}$  has the property that  $S \models \varphi$  iff  $S' \models \varphi$  (for all formulas  $\varphi$  of  $\text{ATL}_{\mathbb{K}, \mathbb{C}, \mathbb{D}}^*$ ).

**PROOF.** First, note that  $\Lambda(p)$  is regular since a DFW can read the history  $h$  and store in its state whether or not the last state it read is in  $\lambda(p)$  or not. Second, the fact that  $S \models \varphi$  iff  $S' \models \varphi$  holds by a straightforward induction on the structure of  $\varphi$ .  $\square$

A generalised PA-iCGS is a generalised iCGS that only has public actions, i.e., it satisfies the condition in Definition 3 (which does not depend on the labeling). For the rest of the proof we view  $S$  as a generalised PA-iCGS.

**Inductive Statement.** Let  $S$  be a generalised PA-iCGS. For every history formula  $\varphi$  and initial state  $s \in S_0$  we will build a DFW  $M_\varphi^s$  such that for every  $u \in \text{ACT}^*$ ,

$$M_\varphi^s \text{ accepts } u \text{ iff } (S, \mu(s, u)) \models \varphi.$$

From this it is easy to get the decidability stated in the theorem: simply check that  $\epsilon$  is accepted by every  $M_\varphi^s$  with  $s \in S_0$ .

We build the DFW  $M_\varphi^s$ , simultaneously for all  $s \in S_0$ , by induction on  $\varphi$ .

**$\varphi$  is an atom.** Say  $\varphi = p \in AP$  and let  $s \in S_0$ . The required DFW  $M_\varphi^s$  should accept  $u \in \text{ACT}^*$  iff  $\mu(s, u)$  is accepted by the DFW  $\Lambda(p)$ . To do this we define  $M_\varphi^s$  to simulate  $S$  and the DFW  $R = (S, Q, q_0, \rho, F)$  for  $\Lambda(p)$  in parallel, i.e., by taking a product of  $S$  and  $R$ . Formally, define  $M_\varphi^s = (\text{ACT}, S \times Q, (\iota, q_0), \tau, F')$  where the transition function  $\tau$  maps state  $(s, q)$  and input  $a \in \text{ACT}$  to state  $(\delta(s, a), \rho(q, s))$ , and the final states  $F'$  are of the form  $(s, q)$  where  $\rho(q, s) \in F$ .

**$\varphi$  is a Boolean combination.** Let  $s \in S_0$ . The Boolean combinations follow from the effective closure of DFW under complementation and intersection. Indeed,  $M_{\neg\varphi}^s$  is formed by complementing the final states of  $M_\varphi^s$ , and  $M_{\varphi_1 \wedge \varphi_2}^s$  is the product of the  $M_{\varphi_i}^s$ 's.

**$\varphi$  is of the form  $\mathbb{K}_a \varphi'$ .** Let  $s \in S_0$ . By induction, we have DFW  $M_{\varphi'}^t$  for  $t \in S_0$ . The required DFW should accept a string  $u$  iff, for every  $t \in S_0$ , if  $\mu(s, u) \equiv_a \mu(t, u)$  then  $M_{\varphi'}^t$  accepts  $u$ .

To do this, the DFW will simulate, in parallel, each  $M_{\varphi'}^t$  for  $t \in S_0$ . This is done by forming their product, i.e., the states of the product are  $q : S_0 \rightarrow Q$  where  $Q$  is the union of the state sets of the  $M_{\varphi'}^t$  for  $t \in S_0$ , and there is a transition in the product from  $q$  to  $q'$  on input  $J \in \text{ACT}$  if for each  $t \in S_0$  there is a transition in  $M_{\varphi'}^t$  from  $q(t)$  to  $q'(t)$  on input  $J$ . Instrument this product by recording, on input  $u \in \text{ACT}^*$  a function  $f_u : S_0 \rightarrow S$  and a set  $G_u \subseteq S_0$  with the following properties:

- $f_u(t) = \text{last}(\mu(t, u))$
- $t \in G_u$  iff for every prefix  $v$  of  $u$ ,  $f_v(s) \sim_a f_v(t)$  (thus, initially  $G_\epsilon = \{t \in S_0 : t \sim_a s\}$ , and the moment  $f_u(s) \not\sim_a f_u(t)$  we remove  $t$  from  $G_v$ ).

A state  $\langle f, G, q \rangle$  is final if, for every  $t \in G$  it is also the case that  $q(t)$  is a final state of  $M_\varphi^t$ .

**$\varphi$  is of the form  $\mathbb{C}_A\varphi'$ .** This is identical to the  $\mathbb{K}_a$  case except replace  $\sim_a$  by  $\sim_A^\mathbb{C}$  and replace  $\equiv_a$  by  $\equiv_A^\mathbb{C}$ .

**$\varphi$  is of the form  $\mathbb{D}_A\varphi'$ .** This is identical to the  $\mathbb{K}_a$  case except replace  $\sim_a$  by  $\sim_A^\mathbb{D}$  and replace  $\equiv_a$  by  $\equiv_A^\mathbb{D}$ .

**$\varphi$  is of the form  $\langle\langle A\rangle\rangle\psi$ .** We proceed in two steps. First, we show how to linearise path formulas, and then we show how to encode strategies as trees so that we can build the promised DFW.

*Linearising path formulas.* One can think of an  $\text{ATL}_{\mathbb{K},\mathbb{C},\mathbb{D}}^*$  path formula  $\psi$  as an LTL formula  $\text{lin}(\psi)$  over a fresh set of atoms  $\text{max}(\psi)$ , the maximal history subformulas of  $\psi$ . This translation of  $\text{ATL}_{\mathbb{K},\mathbb{C},\mathbb{D}}^*$  path formulas to LTL formulas does not make use of the assumption that the iCGS  $S$  only has public actions, and it is analogous to the translation of  $\text{ATL}^*$  (or  $\text{CTL}^*$ ) path formulas to LTL formulas over maximal state subformulas [19, 1].

We briefly discuss the translation. Let  $\text{max}(\psi)$  be the set of history subformulas of  $\psi$  that are maximal, i.e., a history formula  $\varphi \in \text{max}(\psi)$  iff it occurs in  $\psi$  and that occurrence is not a subformula of any other occurrence of a history subformula of  $\psi$ . If  $\psi$  is a path formula, define  $\text{lin}(\psi)$  to be the LTL formula where for each  $\varphi \in \text{max}(\psi)$  there is a fresh atom  $\boxed{\varphi}$  such that every occurrence of  $\varphi$  in  $\psi$  is replaced by  $\boxed{\varphi}$ . For example, consider  $\psi = (p \mathbb{U} \langle\langle A\rangle\rangle \mathbb{K}_a p) \vee X \neg \mathbb{C}_A p$ . The history subformulas of  $\psi$  are  $\{p, \langle\langle A\rangle\rangle \mathbb{K}_a p, \mathbb{K}_a p, \neg \mathbb{C}_A p, \mathbb{C}_A p\}$ . Then  $\text{max}(\psi) = \{p, \langle\langle A\rangle\rangle \mathbb{K}_a p, \neg \mathbb{C}_A p\}$ .<sup>1</sup> Thus  $\text{lin}(\psi)$  is the LTL formula  $(\boxed{p} \mathbb{U} \boxed{\langle\langle A\rangle\rangle \mathbb{K}_a p}) \vee X \boxed{\neg \mathbb{C}_A p}$  over the atoms  $\boxed{\varphi}$  for  $\varphi \in \text{max}(\psi)$ . Since, by induction, we have built DFW for each boxed atom, for the remainder of the proof we assume that  $\psi$  is an LTL formula.

*Construction of  $M_{\langle\langle A\rangle\rangle\psi}^s$  for an LTL formula  $\psi$ .* We will build a DPW  $E_{\psi,s}$  over ACT that accepts  $\alpha \in \text{ACT}^\omega$  iff  $(S, \mu(s, \alpha)) \models \psi$ . The promised DFW  $M_{\langle\langle A\rangle\rangle\psi}^s$  reads  $u \in \text{ACT}^*$  and simulates  $E_{\psi,s}$ . Suppose after reading  $u$  the state of  $E_{\psi,s}$  is  $q$ . Then  $M_{\langle\langle A\rangle\rangle\psi}^s$  accepts, i.e.,  $q$  is defined to be a final state of  $M_{\langle\langle A\rangle\rangle\psi}^s$ , iff there exists uniform  $\sigma_A$  such that for every  $\alpha \in \text{ACT}^\omega$ , if  $\mu(s, u \cdot \alpha) \in \text{out}(S, \mu(s, u), \sigma_A)$  then  $\alpha$  is accepted by  $E_{\psi,s}$  starting from state  $q$  (and thus  $(S, \mu(s, u)) \models \langle\langle A\rangle\rangle\psi$ , as required). These latter realisability problems (one for each  $q$ ) are solved offline by constructing DPT  $F_{\psi,s,q}$  and testing them for non-emptiness. We now show how to build the automata  $E_{\psi,s}$  and  $F_{\psi,s,q}$ .

*Construction of DPW  $E_{\psi,s}$ .* To build  $E_{\psi,s}$ , begin by writing  $AP(\psi)$  for the finite set of atoms appearing in  $\psi$ . First, for each  $p \in AP(\psi)$ , let  $D^p$  be a DFW for the regular set  $\{u \in \text{ACT}^* : \mu(s, u) \in \Lambda(p)\}$ . Second, by Proposition 1, every LTL formula  $\psi$  can be converted into a DPW  $D_\psi$  over alphabet  $2^{AP}$  that accepts all word models of that formula. Let  $Q_\psi$  be the states and  $\Delta_\psi : Q_\psi \times 2^{AP(\psi)} \rightarrow Q_\psi$  the transition of the DPW. Now, the DPW  $E_{\psi,s}$  simulates  $D_\psi$  and each DFW  $D^p$ . The automaton does this by storing and updating a state  $q_u \in Q_\psi$  and a function  $f_u$  such that  $f_u(p)$  is the state of  $D^p$  (i.e.,  $f_u : AP(\psi) \rightarrow Q$  where  $Q$  is the union of states of the  $D^p$ s). Transitions of  $E_{\psi,s}$  are as follows: from state  $\langle q_u, f_u \rangle$  and input  $d \in \text{ACT}$  the next state  $\langle q_{ud}, f_{ud} \rangle$  satisfies that  $q_{ud} = \Delta_\psi(q_u, Z)$  where  $p \in Z$  iff  $f_u(p)$  is a final state of  $D^p$ , and  $f_{ud}(p)$  is the state resulting from applying

<sup>1</sup>Note that although  $p$  has a non-maximal occurrence in  $\psi$ , it is included in  $\text{max}(\psi)$  since it has at least one occurrence which is maximal, i.e., on the left-side of U.

the transition function of  $D^p$  to the state  $f_u(p)$  and input  $d$ . Define the colour of  $\langle q, f \rangle$  to be the same as the colour in  $D_\psi$  of the state  $q$ , and  $\langle q, f \rangle$  is an initial state if  $q$  is an initial state in  $D_\psi$ . The following is straightforward to prove:

LEMMA 3. *The DPW  $E_{\psi,s}$  accepts  $\alpha \in \text{ACT}^\omega$  if and only if  $(S, \mu(s, \alpha)) \models \psi$ .*

*Construction of DPT  $F_{\psi,s,q}$ .* To solve the synthesis problem we will encode strategies as trees and use tree-automata. Encode a strategy  $\sigma$  of agent  $a$  by the ACT-branching tree

$$T_\sigma : \text{ACT}^* \rightarrow (\text{Act}_a)^{S_0}$$

where, for  $u \in \text{ACT}^*$ ,  $T_\sigma(u)(t) = \sigma(\mu(t, u))$ . By Lemma 1,  $\sigma$  is uniform iff  $T_\sigma$  satisfies the property

$$\mu(t, u) \equiv_a \mu(t', u) \Rightarrow T_\sigma(u)(t) = T_\sigma(u)(t') \quad (1)$$

LEMMA 4. *The set of encodings  $T_\sigma$  of uniform strategies  $\sigma$  of agent  $a$  is recognised by a DPT.*

PROOF. To do this, it is sufficient to build a DPT that accepts a tree  $T$  iff  $T$  has property (1). Informally, for every pair of different states  $t, t' \in S_0$ , the DPT keeps a bit that is initialised to 1 (signifying that it must verify that  $T(u)(t) = T(u)(t')$ ), and as soon as it finds that  $\mu(t, u) \not\equiv_a \mu(t', u)$  it sets the bit to 0, which signals that it no longer has to ensure  $T_\sigma(u)(t) = T_\sigma(u)(t')$ .  $\square$

Encode a set of uniform strategies  $\sigma_A$  (for  $A \subseteq Ag$ ) as the convolution  $T_{\sigma_A}$  of their individual encodings, i.e.,

$$T_{\sigma_A} : \text{ACT}^* \rightarrow \prod_{a \in A} (\text{Act}_a)^{S_0}$$

where  $T_{\sigma_A}(u)(a) = T_{\sigma_a}(u)$ . Running the automata for  $\sigma_a$  in parallel yields:

LEMMA 5. *For every  $A \subseteq Ag$ , the set of encodings of joint uniform strategies  $\sigma_A$  is recognised by a DPT.*

Finally, in order to solve the synthesis problem for state  $q$ , we define a DPT  $F_{\psi,s,q}$  that accepts  $T_{\sigma_A}$  iff for every  $\alpha \in \text{ACT}^\omega$ , if  $\mu(s, \alpha)$  is consistent with  $\sigma_A$  then  $\alpha$  is accepted by  $E_{\psi,s}$  starting from  $q$ .

The DPT  $F_{\psi,s,q}$  works as follows: it reads  $T_{\sigma_A}$  as input and, on every branch  $\alpha \in \text{ACT}^\omega$  of the tree, simulates  $E_{\psi,s}$  starting from  $q$ , and accepts that branch if both  $E_{\psi,s}$  is accepting and  $\mu(s, \alpha)$  is consistent with  $\sigma_A$ . This can be done by a tree-automaton because a)  $E_{\psi,s}$  is deterministic and thus it can be run on all paths of the tree (i.e., this step would not be possible if  $E_{\psi,s}$  were a non-deterministic automaton); and b) checking if  $\mu(s, \alpha)$  is consistent with  $\sigma_A$  is simply a matter of checking that  $\alpha_0(a) = \sigma_A(a)(s)$  and  $\alpha_{i+1}(a) = \sigma_A(a)(\mu(s, \alpha_{\leq i}))$ , for every  $i \in \mathbb{N}$  and  $a \in A$ .

*Final states of DFW  $M_{\langle\langle A\rangle\rangle\psi}^s$ .* Finally, define the state  $q$  of  $M_{\langle\langle A\rangle\rangle\psi}^s$  to be a final state iff the DPT  $F_{\psi,s,q}$  is non-empty (a decidable condition).

This completes the construction of  $M_{\langle\langle A\rangle\rangle\varphi}^s$ , and the proof of decidability.

### 3.1 Complexity

In this section we analyse the complexity of our decision procedure for a fixed number of agents. We then establish the lower bound by a reduction from a problem known to be 2EXPTIME-hard.

First, we calculate  $\|\varphi\|$ , the number of states of  $M_\varphi^s$ , for each case:

1. Atomic:  $\|p\| = O(1)$  for  $p \in AP$ .
2. Negation:  $\|\neg\varphi\| = \|\varphi\|$ .
3. Conjunction:  $\|\varphi \wedge \varphi'\| = \|\varphi\| \times \|\varphi'\|$ .
4. Epistemic:  $\|\mathbb{K}_a\varphi\| = (2 \times |S| \times \|\varphi\|)^{|S|}$ , and the same for  $\|\mathbb{C}_A\varphi\|$  and  $\|\mathbb{D}_A\varphi\|$ .
5. Strategic:  $\|\langle\langle A\rangle\rangle\psi\| = 2^{O(|lin(\psi)|)} \times \|\tilde{\psi}\||AP(lin(\psi))| \times O(2^{|S|^2})$  where  $lin(\psi)$  is the linearisation of  $\psi$ ,  $\tilde{\psi}$  is the largest history subformula of  $\psi$ , and  $AP(\cdot)$  is the set of atomic predicates occurring in its argument.

The last case requires some explanation. The DPT accepting the set of all joint-strategies  $\sigma_A$  has size  $O(2^{|S|^2})$ , and has two colours. The DPW  $D_\psi$  has double-exponentially many states and single-exponentially many colours (in the size of  $lin(\psi)$ ) [35, 29]. The DPW  $E_{\psi,s}$  has  $2^{O(|lin(\psi)|)} \times \|\tilde{\psi}\||AP(lin(\psi))|$  many states. The DPT  $F_{\psi,s,q}$  has  $2^{O(|lin(\psi)|)} \times \|\tilde{\psi}\||AP(\psi)| \times O(2^{|S|^2})$  many states and  $2^{O(|lin(\psi)|)}$  many colours. This gives the stated value of  $\|\langle\langle A\rangle\rangle\psi\|$ .

Second, we calculate the time for constructing the DFW  $M_\varphi^s$ . In the first four cases this cost is polynomial in the size of the DFW, i.e.,  $\|\varphi\|$ . For the strategy case we incur a cost to calculate the final states, i.e., solving the emptiness of the DPT  $F_{\psi,s,q}$ . The cost of solving the emptiness of a DPT with  $n$  states and  $m$  colours is at most  $n^{O(m)}$  [16]. Thus, the time for constructing  $M_\varphi^s$  is at most  $n^{O(m)}$  where  $n = 2^{O(|lin(\psi)|)} \times \|\tilde{\psi}\||AP(lin(\psi))| \times O(2^{|S|^2})$  and  $m = 2^{O(|lin(\psi)|)}$ .

Finally, let  $z = |S| + |\varphi|$ . The time for constructing  $M_\varphi^s$  of each step of the procedure can be bounded above by  $2^{2^{O(z)}} \times |\Lambda|^{2^{O(z)}}$  where  $|\Lambda|$  is the size of the largest DFW representing atoms of the generalised PA-iCGS (a maximum exists since  $AP$  is finite). Since there are at most  $z$  such steps, the time for constructing, and thus the size, of the resulting DFW is  $2^{2^{O(z^2)}}$ . Testing if  $\epsilon$  is accepted by this automaton has no additional cost. Thus, our algorithm runs in 2EXPTIME.

**Lower-Bound.** To prove the lower bound in Theorem 1 we reduce from a known 2EXPTIME-hard problem, i.e., model checking a CGS with  $Ag = \{a, b\}$  against a formula of the form  $\langle\langle \{a\} \rangle\rangle\psi$  for an LTL formula  $\psi$  [27, 28]. One can translate a two-player CGS  $S$  into a polynomially larger CGS  $S'$  with public actions such that  $S \models \varphi$  iff  $S' \models \varphi$ . Indeed, the agents, actions, and atoms are the same,  $S' = S \times (\text{ACT} \cup \{\epsilon\})$ ,  $S'_0 = S_0 \times \{\epsilon\}$ ,  $\delta'((s, d), d') = (\delta(s, d'), d')$ , and  $\lambda'(p) = \{(s, d) : s \in \lambda(p)\}$  (note that because  $S$  has two-players,  $|\text{ACT}| = |\text{Act}_1| \times |\text{Act}_2|$ , and thus  $|S'|$  is polynomial in the size of  $S$ ).

## 3.2 Subjective Semantics

In this section we show that our result still holds if we use subjective semantics instead of objective semantics.

Our definition of semantics of  $\langle\langle A\rangle\rangle\psi$  is called “objective” (see Remark 1). An alternative definition is called “subjective”: replace “for all  $\pi \in \text{out}(h, \sigma_A)$ ” by “for all  $\pi \in \bigcup_{a \in A, h' \equiv_a h} \text{out}(S, h', \sigma_A)$ ” in the semantics  $(S, h) \models \langle\langle A\rangle\rangle\psi$ . Subjective semantics expresses, intuitively, that the agents  $A$  know that a given strategy will guarantee a certain outcome. We remark that in this case  $\mathbb{K}_a$  is definable in terms of  $\langle\langle A\rangle\rangle$ , i.e.,  $\langle\langle a\rangle\rangle\varphi \mathbf{U} \varphi$ .

The decidability proof of Theorem 1 is for objective semantics. To deal with subjective semantics proceed as follows. For  $u \in \text{ACT}^*$  let  $T_u^s \subseteq S_0$  be the set of  $t$  such that there exists  $a \in A$  with  $t \equiv_a s$ . The DFW  $M_{\langle\langle A\rangle\rangle\psi}^s$  simulates  $E_{\psi,t}$  for all  $t \in T_u^s$ . After reading  $u$ , each automaton  $E_{\psi,t}$  for  $t \in T_u^s$  is in some state, say  $q_t$ . The DFW is required to accept  $u$  iff there exists a joint strategy  $\sigma_A$  such that for every  $\alpha \in \text{ACT}^*$  and  $t \in T_u^s$ , if  $\mu(t, u \cdot \alpha) \in \text{out}(S, \mu(t, u), \sigma_A)$  then  $\alpha$  is accepted by  $E_{\psi,t}$  starting from  $q_t$ . In order to decide the right-hand side we build the DPT  $F_{\psi,t,q_t}$  for  $t \in T_u^s$  (as we did above for  $s$ ). Then we check if the intersection of the automata  $F_{\psi,t,q_t}$  (for  $t \in T_u^s$ ) is non-empty.

## 4 COMPARISON WITH BROADCAST ENVIRONMENTS

The work most closely related to ours is [33] in which the authors show that one can decide if a given formula of knowledge and linear-time, which we denote  $\text{LK}$ , is realisable (by a tuple of uniform strategies) assuming that the environment is a “broadcast environment”. In this section we denote the logic from [33] by  $\text{LK}$ .

The relationship with [33] is twofold: i) the realisability problem for  $\text{LK}$  can be reduced to model checking  $\text{ATL}_\mathbb{K}^*$  specifications, and ii) our notion of “having only public actions” (Definition 3) is orthogonal to “broadcast environments”. We now supply the justifications for i) and ii).

i)  $\text{LK}$  realisability can be reduced to model-checking  $\text{ATL}_\mathbb{K}^*$ . We show how to reduce the realisability problem for  $\text{LK}$  to the model checking problem for  $\text{ATL}_\mathbb{K}^*$ . To do so, we first recall the syntax and semantics of  $\text{LK}$  from [33]. The syntax is defined as the set of formulas  $\psi$  generated by the following grammar:

$$\psi ::= p \mid \neg\psi \mid \psi \wedge \psi \mid X\psi \mid \psi \mathbf{U} \psi \mid \mathbb{K}_a\psi$$

where  $p \in AP$ ,  $a \in Ag$ , and  $A \subseteq Ag$  is non-empty.

The semantics  $\models_{\text{LK}}$  is defined over  $(S, \pi, m)$  where  $S$  is an iCGS,  $\pi \in \text{comp}(S)$  and  $n \in \mathbb{N}$ . We denote the satisfaction relation  $\models_{\text{LK}}$  to distinguish it from  $\models$ . The Boolean and temporal operators are as usual:

$$\begin{aligned} (S, \pi, m) \models_{\text{LK}} p &\quad \text{iff } \pi_m \in \lambda(p) \\ (S, \pi, m) \models_{\text{LK}} \neg\psi &\quad \text{iff } (S, \pi, m) \not\models_{\text{LK}} \psi \\ (S, \pi, m) \models_{\text{LK}} \psi_1 \wedge \psi_2 &\quad \text{iff } (S, \pi, m) \models_{\text{LK}} \psi_i \text{ for } i \in \{1, 2\} \\ (S, \pi, m) \models_{\text{LK}} X\psi &\quad \text{iff } (S, \pi, m+1) \models_{\text{LK}} \psi \\ (S, \pi, m) \models_{\text{LK}} \psi_1 \mathbf{U} \psi_2 &\quad \text{iff for some } j \geq m, (S, \pi, j) \models_{\text{LK}} \psi_2, \\ &\quad \text{and for all } k \text{ with } m \leq k < j, \\ &\quad (S, \pi, k) \models_{\text{LK}} \psi_1 \end{aligned}$$

The epistemic operator  $\mathbb{K}_a$  is follows:

$$(S, \pi, m) \models_{\text{LK}} \mathbb{K}_a\psi \text{ iff } (S, \pi', m') \models_{\text{LK}} \psi \text{ for all } \pi' \in \text{comp}(S) \text{ such that } \pi_{\leq m} \equiv_a \pi'_{\leq m'} \text{ (in particular, } m = m').$$

An  $\text{LK}$ -formula  $\psi$  is *realisable* if there exists a uniform strategy  $\sigma_A$  such that for all  $s_0 \in S_0$  and all  $\pi \in \text{out}(S, s_0)$ , we have that  $(S, \pi, 0) \models \psi$ .

We now present the reduction. Let  $\psi$  be a formula of  $\text{LK}$ . Define  $\hat{\psi}$ , a path formula of  $\text{CTL}_\mathbb{K}^*$ , by recursively replacing  $\mathbb{K}_a\psi$  by  $\mathbb{K}_a A \hat{\psi}$ . We claim that for all iCGS  $S$ ,  $\psi$  is realisable in  $S$  iff  $S \models \langle\langle Ag \rangle\rangle\hat{\psi}$ . To see this it is sufficient to establish the following inductive hypothesis:  $(S, \pi, m) \models_{\text{LK}} \psi$  iff  $(S, \pi, m) \models \hat{\psi}$ . To prove the inductive hypothesis use that the following are equivalent to  $(S, \pi, m) \models_{\text{LK}} \mathbb{K}_a\psi$ :

1.  $(S, \pi', m) \models_{LK} \psi$  for all  $\pi' \in \text{comp}(S)$  such that  $\pi'_{\leq m} \equiv_a \pi_{\leq m}$  (by the definition of  $\models_{LK}$ );
2.  $(S, \pi', m) \models \hat{\psi}$  for all  $\pi' \in \text{comp}(S)$  such that  $\pi'_{\leq m} \equiv_a \pi_{\leq m}$  (by the inductive hypothesis);
3.  $(S, \pi, m) \models \mathbb{K}_a A \hat{\psi}$  (by definition of  $\models$ ).

*ii) Broadcast environments and PA-iCGS are incomparable.* We briefly describe how [33] models “broadcast environment”. That work defines an interpreted systems (see [8] for background on these) in which each agent’s local state consists of a private part (that only depends on its local actions) and a shared part (that depends on the joint actions, but that is the same for all agents). In our terminology a broadcast environment is an iCGS with  $Ag = \{e, 1, 2, \dots, n\}$  whose state set is of the form  $S = L_e \times \prod_{i \leq n} L_i$  (for some finite sets  $L_a$  for  $a \in Ag$ ), and whose transition maps state  $(l_e, l_1, \dots, l_n)$  and joint-action  $J \in ACT$  to the state  $(\tau_e(l_e, J), \tau_1(l_1, J(1)), \dots, \tau_n(l_n, J(n)))$  where  $\tau_e : L_e \times ACT \rightarrow L_e$  and  $\tau_i : L_i \times Act_i \rightarrow L_i$  are functions (similar to the evolution functions in interpreted systems), except that  $L_i$  for  $i \neq e$  does not depend on joint-actions, only on local actions). Finally, define  $(l_e, l_1, \dots, l_n) \sim_i (l'_e, l'_1, \dots, l'_n)$  iff  $l_i = l'_i$  and  $F(l_e) = F(l'_e)$  where  $F : L_e \rightarrow O$  is a function mapping environment states to some fixed set  $O$  of observations (note that  $F$  and  $O$  are independent of  $i$ , and thus all agents have the same observation of the environment).

Now, the set of PA-iCGS is incomparable (wrt. subset) with these iCGS. On the one hand, setting  $L_e = ACT = O$  and  $F$  to be the identity function, results in an iCGS having only public actions. On the other hand, we allow  $L_i$  to depend on the joint-actions (not just the local actions).

## 5. CONCLUSIONS

In this paper we put forward a class of CGS with imperfect information, namely the iCGS only having public actions, which admit a decidable model checking problem, even in the presence of perfect recall. This is in contrast with the fact that even realisability of safety properties on arbitrary iCGS is undecidable [6]. Specifically, we considered a rich formal language to express complex strategic and epistemic properties of agents in MAS. This is the extension  $ATL_{K,C,D}^*$  of the alternating-time temporal logic  $ATL^*$ , with operators for individual, common, and distributed knowledge. We provided these languages with a semantics in terms of iCGS, according to both the objective and subjective interpretation of ATL modalities. Most importantly, we identified a subclass of iCGS – those having only public actions, or PA-iCGS – for which we were able to prove that the model-checking problem is decidable. The interest of these results lies in the fact that PA-iCGS capture many important MAS scenarios, including certain games of imperfect information, epistemic puzzles, blackboard systems, face to face communication, etc. Indeed, all scenarios mentioned in previous work on broadcast environments [23, 33] can be captured by PA-iCGS.

A number of extensions of  $ATL^*$  have been proposed in order to express classic solutions concepts (like Nash Equilibria) [12, 13, 24, 25]. The decidability of model checking PA-iCGS against epistemic extensions of these strategy logics is currently unexplored.

Notwithstanding their generality, there are many features of MAS that are not naturally expressed within PA-iCGS or broadcast environments. We discuss some of them:

*Asynchronous recall.* Social media like Twitter make use of public actions, but are more naturally modeled as asynchronous MAS (rather than synchronous systems, as we do). *Bounded-recall.* Games like Bridge and Stratego are interesting to play in part because humans have to remember some of the history of a play, a feature that might be modeled by bounded recall (rather than perfect recall). However, restricting agents to finite-memory strategies also results in undecidability on arbitrary iCGS [36]. On the other hand, our proof implies that if a formula  $\langle\langle A \rangle\rangle \psi$  is true then there are finite-memory strategies witnessing this fact, and if a formula  $\mathbb{K}_a \varphi$  is true then there is a finite-state machine that accepts exactly the histories making  $\mathbb{K}_a \varphi$  true. This suggests that our results can be used to model agents of bounded-recall.

*Probabilities.* Several scenarios, such as card games and security protocols, involve probability either at the level of the iCGS or at the level of strategies.

In future work we plan to investigate the points raised above, as well as to develop optimal model checking algorithms for fragments of  $ATL_{K,C,D}^*$  and to implement them in an extension of the MCMAS tool for MAS verification [22]. **Acknowledgements.** F. Belardinelli acknowledges the support of the ANR JCJC Project SVeDaS (ANR-16-CE40-0021). S. Rubin is a Marie Curie fellow of the Istituto Nazionale di Alta Matematica. The authors thank Benjamin Aminof for fruitful discussions.

## REFERENCES

- [1] R. Alur, T. Henzinger, and O. Kupferman. Alternating-Time Temporal Logic. *Journal of the ACM*, 49(5):672–713, 2002.
- [2] D. Berwanger, A. B. Mathew, and M. van den Bogaard. Hierarchical Information Patterns and Distributed Strategy Synthesis. In *ATVA ’15*, LNCS 9364, pages 378–393. Springer, 2015.
- [3] N. Bulling and W. Jamroga. Comparing variants of strategic ability: how uncertainty and memory influence general properties of games. *Journal of Autonomous agents and multi-agent systems*, 28(3):474–518, 2014.
- [4] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.
- [5] C. Dima, C. Enea, and D. Guelev. Model-checking an alternating-time temporal logic with knowledge, imperfect information, perfect recall and communicating coalitions. In *GandALF 2010*, volume 25 of *EPTCS*, pages 103–117, 2010.
- [6] C. Dima and F. L. Tiplea. Model-checking ATL under imperfect information and perfect recall semantics is undecidable. *CoRR*, abs/1102.4225, 2011.
- [7] D. Easley and J. Kleinberg. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*. Cambridge University Press, 2010.
- [8] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning about Knowledge*. MIT Press, 1995.
- [9] B. Finkbeiner and S. Schewe. Uniform Distributed Synthesis. In *LICS’05*, pages 321–330. IEEE Computer Society, 2005.

- [10] V. Goranko and W. Jamroga. Comparing semantics of logics for multi-agent systems. *Synthese*, 139(2):241–280, 2004.
- [11] D. P. Guelev, C. Dima, and C. Enea. An alternating-time temporal logic with knowledge, perfect recall and past: axiomatisation and model-checking. *Journal of Applied Non-Classical Logics*, 21(1):93–131, 2011.
- [12] J. Gutierrez, P. Harrenstein, and M. Wooldridge. Reasoning about equilibria in game-like concurrent systems. In *KR’14*. AAAI, 2014.
- [13] J. Gutierrez, P. Harrenstein, and M. Wooldridge. Expressiveness and complexity results for strategic reasoning. In *CONCUR’15*, volume 42 of *LIPICS*, 2015.
- [14] W. Hoek and M. Wooldridge. Cooperation, knowledge, and time: Alternating-time temporal epistemic logic and its applications. *Studia Logica*, 75(1):125–157, 2003.
- [15] W. Jamroga and W. van der Hoek. Agents that know how to play. *Fundamenta Informaticae*, 62:1–35, 2004.
- [16] M. Jurdzinski. Small Progress Measures for Solving Parity Games. In *STACS’00*, LNCS 1770, pages 290–301. Springer, 2000.
- [17] P. Kazmierczak, T. Ågotnes, and W. Jamroga. Multi-agency is coordination and (limited) communication. In *PRIMA’14*, LNCS 8861, pages 91–106. Springer, 2014.
- [18] O. Kupferman and M. Vardi. Synthesizing Distributed Systems. In *LICS’01*, pages 389–398. IEEE Computer Society, 2001.
- [19] O. Kupferman, M. Vardi, and P. Wolper. An Automata Theoretic Approach to Branching-Time Model Checking. *Journal of the ACM*, 47(2):312–360, 2000.
- [20] O. Kupferman, M. Vardi, and P. Wolper. Module Checking. *Information and Computation*, 164(2):322–344, 2001.
- [21] F. Laroussinie, N. Markey, and A. Sangnier. ATLsc with partial observation. In *GandALF 2015*, volume 193 of *EPTCS*, pages 43–57, 2015.
- [22] A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: A model checker for the verification of multi-agent systems. *Software Tools for Technology Transfer*, 19(1):9–30, 2017.
- [23] A. R. Lomuscio, R. van der Meyden, and M. Ryan. Knowledge in multiagent systems: Initial configurations and broadcast. *ACM Trans. Comput. Logic*, 1(2):247–284, Oct. 2000.
- [24] A. Lopes, F. Laroussinie, and N. Markey. ATL with Strategy Contexts: Expressiveness and Model Checking. In *FSTTCS’10*, LIPICS 8, pages 120–132. Leibniz-Zentrum fuer Informatik, 2010.
- [25] F. Mogavero, A. Murano, G. Perelli, and M. Vardi. Reasoning About Strategies: On the Model-Checking Problem. *TOCL*, 15(4):34:1–42, 2014.
- [26] H. P. Nii. Blackboard systems, part one: The blackboard model of problem solving and the evolution of blackboard architectures. *AI Magazine*, 7(2):38–53, 1986.
- [27] A. Pnueli and R. Rosner. Distributed Reactive Systems Are Hard to Synthesize. In *FOCS’90*, pages 746–757. IEEE Computer Society, 1990.
- [28] R. Rosner. *Modular Synthesis of Reactive Systems*. PhD thesis, Weizmann Institute of Science, Rehovot, Israel, 1992.
- [29] S. Safra. *Complexity of Automata on Infinite Objects*. PhD thesis, Weizmann Institute of Science, Rehovot, Israel, 1989.
- [30] S. Schewe and B. Finkbeiner. Distributed synthesis for alternating-time logics. In K. S. Namjoshi, T. Yoneda, T. Higashino, and Y. Okamura, editors, *ATVA’07*, pages 268–283. Springer, 2007.
- [31] P. Schobbens. Alternating-Time Logic with Imperfect Recall. *ENTCS*, 85(2):82–93, 2004.
- [32] R. van der Meyden and H. Shilov. Model checking knowledge and time in systems with perfect recall. In *FST&TCS’99*, LNCS 1738, pages 432–445. Springer, 1999.
- [33] R. van der Meyden and T. Wilke. Synthesis of distributed systems from knowledge-based specifications. In *CONCUR’05*, LNCS 3653, pages 562–576. Springer, 2005.
- [34] H. van Ditmarsch, W. van der Hoek, and B. Kooi. *Dynamic Epistemic Logic*, volume 337 of *Synthese Library*. Springer, 2007.
- [35] M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.
- [36] S. Vester. Alternating-time temporal logic with finite-memory strategies. In *GandALF’13*, volume 119 of *EPTCS*, pages 194–207, 2013.

## **4.12 Publication**

The rest of this page is intentionally left blank

# Strategy Logic with Imperfect Information

Raphaël Berthon\*, Bastien Maubert†, Aniello Murano†, Sasha Rubin† and Moshe Y. Vardi‡

\*École Normale Supérieure de Rennes, Rennes, France

†Università degli Studi di Napoli Federico II, Naples, Italy

‡Rice University, Houston, Texas, USA

**Abstract**—We introduce an extension of Strategy logic for the imperfect-information setting, called  $\text{SL}_{\text{ii}}$ , and study its model-checking problem. As this logic naturally captures multi-player games with imperfect information, the problem turns out to be undecidable. We introduce a syntactical class of “hierarchical instances” for which, intuitively, as one goes down the syntactic tree of the formula, strategy quantifications are concerned with finer observations of the model. We prove that model-checking  $\text{SL}_{\text{ii}}$  restricted to hierarchical instances is decidable. This result, because it allows for complex patterns of existential and universal quantification on strategies, greatly generalises previous ones, such as decidability of multi-player games with imperfect information and hierarchical observations, and decidability of distributed synthesis for hierarchical systems.

To establish the decidability result, we introduce and study  $\text{QCTL}_{\text{ii}}^*$ , an extension of QCTL (itself an extension of CTL with second-order quantification over atomic propositions) by parameterising its quantifiers with observations. The simple syntax of  $\text{QCTL}_{\text{ii}}^*$  allows us to provide a conceptually neat reduction of  $\text{SL}_{\text{ii}}$  to  $\text{QCTL}_{\text{ii}}^*$  that separates concerns, allowing one to forget about strategies and players and focus solely on second-order quantification. While the model-checking problem of  $\text{QCTL}_{\text{ii}}^*$  is, in general, undecidable, we identify a syntactic fragment of hierarchical formulas and prove, using an automata-theoretic approach, that it is decidable. The decidability result for  $\text{SL}_{\text{ii}}$  follows since the reduction maps hierarchical instances of  $\text{SL}_{\text{ii}}$  to hierarchical formulas of  $\text{QCTL}_{\text{ii}}^*$ .

## I. INTRODUCTION

Logics for strategic reasoning are a powerful tool for expressing correctness properties of multi-player graph-games, which in turn are natural models for reactive systems and discrete event systems. In particular,  $\text{ATL}^*$  and its related logics were introduced to capture the realisability/synthesis problem for open systems with multiple components. These logics were designed as extensions of branching-time logics such as  $\text{CTL}^*$  that allow one to write alternating properties directly in the syntax, i.e., statements of the form “there exist strategies, one for each player in  $A$ , such for all strategies of the remaining players, the resulting play satisfies  $\varphi$ ”. Strategy logic ( $\text{SL}$ ) [1] generalises these by treating strategies as first-order objects  $x$  that can be quantified  $\langle\langle x \rangle\rangle$  (read “there exists a strategy  $x$ ”) and bound to players  $(a, x)$  (read “player  $a$  uses strategy  $x$ ”). This syntax has flexibility very similar to first-order logic, and thus allows one to directly express many solution concepts from game-theory, e.g., the  $\text{SL}$  formula  $\langle\langle x_1 \rangle\rangle \langle\langle x_2 \rangle\rangle (a_1, x_1)(a_2, x_2) \wedge_{i=1,2} [\langle\langle y_i \rangle\rangle (a_i, y_i) goal_i] \rightarrow goal$  expresses the existence of a Nash equilibrium in a two-player game (with individual Boolean objectives).

An essential property of realistic multi-player games is that players only have a limited view of the state of the system. This is captured by introducing partial-observability into the models, i.e., equivalence-relations  $o$  (called *observations*) over the state space that specify indistinguishable states. In the formal-methods literature it is typical to associate observations to players. In this paper, instead, we associate observations to strategies. Concretely, we introduce an extension  $\text{SL}_{\text{ii}}$  of  $\text{SL}$  that annotates the strategy quantifier  $\langle\langle x \rangle\rangle$  by an observation  $o$ , written  $\langle\langle x \rangle\rangle^o$ . Thus, both the model and the formulas mention observations  $o$ . This novelty allows one to express, in the logic, that a player’s observation changes over time.

Our logic  $\text{SL}_{\text{ii}}$  is extremely powerful: it is an extension of  $\text{SL}$  (which is in the perfect-information setting), and of the imperfect-information strategic logics  $\text{ATL}_{\text{i},\text{R}}^*$  [2] and  $\text{ATL}_{\text{sci}}^*$  [3]. A canonical specification in multi-player game of partial observation is that the players, say  $a_1, \dots, a_n$ , can form a coalition and beat the environment player, say  $b$ . This can be expressed in  $\text{SL}_{\text{ii}}$  as  $\Phi_{\text{SYNTH}} := \langle\langle x_1 \rangle\rangle^{o_1} \dots \langle\langle x_n \rangle\rangle^{o_n} \llbracket y \rrbracket^o (a_1, x_1) \dots (a_n, x_n)(b, y)\varphi$ , where  $\varphi$  is quantifier- and binding-free. Also,  $\text{SL}_{\text{ii}}$  can express more complicated specifications by alternating quantifiers, binding the same strategy to different agents and rebinding (these are inherited from  $\text{SL}$ ), as well as changing observations.

The complexity of  $\text{SL}_{\text{ii}}$  is also visible from an algorithmic point of view. Its satisfiability problem is undecidable (this is already true of  $\text{SL}$ ), and its model-checking problem is undecidable (this is already true of  $\text{ATL}_{\text{i},\text{R}}^*$ , in fact, even for the single formula  $\langle\langle \{a, b\} \rangle\rangle \text{F}p$  in systems with three players [4]). In fact, similar undecidability occurs in foundational work in multi-player games of partial observation, and in distributed synthesis [5], [6]. Since then, the formal-methods community has spent much effort finding restrictions and variations that ensure decidability [7]–[13]. The common thread in these approaches is that the players’ observations (or what they can deduce from their observations) are hierarchical.

Motivated by the problem of finding decidable extensions of strategy logic in the imperfect-information setting, we introduce a syntactical class of “hierarchical instances” of  $\text{SL}_{\text{ii}}$ , i.e., formula/model pairs, and prove that the model-checking problem on this class of instances is decidable. Intuitively, an instance of  $\text{SL}_{\text{ii}}$  is hierarchical if, as one goes down the syntactic tree of the formula, the observations annotating strategy quantifications can only become finer. Although the class of hierarchical instances refers not only to the syntax of the logic but also to the model, the class is syntactical in the

sense that it depends only on the structure of the formula and the observations in the model. Moreover, it is easy to check (i.e., in linear time) if an instance is hierarchical or not.

The class of hierarchical instances generalises some existing approaches and supplies new classes of systems and properties that can be model-checked. For instance, suppose that there is a total order  $\preceq$  among the players such that  $a \preceq b$  implies player  $b$ 's observation is finer than player  $a$ 's observation — such games are said to yield “hierarchical observation” in [13]. In such games it is known that synthesis against  $\text{CTL}^*$  specifications is decidable (in fact, this holds for  $\omega$ -regular specifications [8], [13]). This corresponds to hierarchical instances of  $\text{SL}_{ii}$  in which the observations form a total order in the model and the formula is of the form  $\Phi_{\text{SYNTH}}$  (mentioned above). On the other hand, in hierarchical instances of  $\text{SL}_{ii}$ , the ordering on observations can be a pre partial-order (i.e., not just total), and one can arbitrarily alternate quantifiers in the formulas. For instance, hierarchical instances allow one to decide if a game that yields hierarchical information has a Nash equilibrium. For example, assuming observations  $p_1, p_2, o_1, o_2$  with  $p_i$  finer than  $o_2$  (for  $i = 1, 2$ ), and  $o_2$  finer than  $o_1$ , the formula  $\langle\langle x_1 \rangle\rangle^{o_1} \langle\langle x_2 \rangle\rangle^{o_2} (a_1, x_1)(a_2, x_2) \wedge_{i=1,2} [\langle\langle y_i \rangle\rangle^{p_i} (a_i, y_i) goal_i] \rightarrow goal_i$  expresses that there exists a strategy profile  $(x_1, x_2)$  of uniform strategies, such that neither player has an incentive to deviate using a strategy that observes at least as much as the observations that both players started with. Observe that this formula is in fact *equivalent* to the existence of a Nash equilibrium, i.e., to the same formula in which  $p_i = o_i$ . This shows we can decide the existence of Nash equilibria in games that yield hierarchical observation.

In order to reason about  $\text{SL}_{ii}$  we introduce an intermediate logic  $\text{QCTL}_{ii}^*$ , an extension to the imperfect-information setting of  $\text{QCTL}^*$  [14], itself an extension of  $\text{CTL}^*$  by second-order quantifiers over atoms. This is a low-level logic that does not mention strategies and into which one can effectively compile instances of  $\text{SL}_{ii}$ . States of the models of the logic  $\text{QCTL}_{ii}^*$  have internal structure, much like the multi-player game structures from [15] and distributed systems [16]. Model-checking  $\text{QCTL}_{ii}^*$  is also undecidable (indeed, we show how to reduce from the MSO-theory of the binary tree extended with the equal-length predicate, known to be undecidable [17]). We introduce the syntactical class  $\text{QCTL}_{i,\subseteq}^*$  of hierarchical formulas as those in which innermost quantifiers observe more than outermost quantifiers, and prove that model-checking is decidable using an extension of the automata-theoretic approach for branching-time logics (our decision to base models of  $\text{QCTL}_{ii}^*$  on local states greatly eases the use of automata). Moreover, the compilation of  $\text{SL}_{ii}$  into  $\text{QCTL}_{ii}^*$  preserves being hierarchical, thus establishing our main contribution, i.e., that model checking the hierarchical instances of  $\text{SL}_{ii}$  is decidable.

**Related work.** Formal methods for reasoning about reactive systems with multiple components have been studied mainly in two theoretical frameworks: a) multi-player graph-games of partial-observation [6], [8], [13] and b) synthesis in distributed architectures [5], [7], [9], [11], [18] (the relationship

between these two frameworks is discussed in [13]). All of these works consider the problem of synthesis, which (for objectives in temporal logics) can be expressed in  $\text{SL}_{ii}$  using the formula  $\Phi_{\text{SYNTH}}$  mentioned above. Limited alternation was studied in [12] that, in the language of  $\text{SL}_{ii}$ , considers the model-checking problem of formulas of the form  $\langle\langle x_1 \rangle\rangle^{o_1} \llbracket x_2 \rrbracket^{o_2} \langle\langle x_3 \rangle\rangle^{o_3} (a_1, x_1)(a_2, x_2)(a_3, x_3)\varphi$ , where  $\varphi$  is an  $\omega$ -regular objective. They prove that this is decidable in case player player 3 has perfect observation and player 2 observes at least as much as player 1.

In contrast to all these works, formulas of  $\text{SL}_{ii}$  can express much more complex specifications by alternating quantifiers, sharing strategies, rebinding, and changing observations.

Recently, [13] generalised the classic result of [8]: it weakens the assumption of hierarchical observation to hierarchical information (which are both static notions), and then, further to dynamic hierarchical information which allows for the hierarchy amongst players' information to change along a play. However, they only consider the synthesis problem.

We are aware of two papers that (like we do) give simultaneous structural constraints on both the formula and the model that result in decidability: in the context of synthesis in distributed architecture with process delays, [18] considers  $\text{CTL}^*$  specifications that constrain external variables by the input variables that may effect them in the architecture; and in the context of asynchronous perfect-recall, [10] considers a syntactical restriction on instances for Quantified  $\mu$ -Calculus with partial observation (in contrast, we consider the case of synchronous perfect recall).

The work closest to ours is [19] which introduces a decidable logic  $\text{CL}$  in which one can encode many distributed synthesis problems. However,  $\text{CL}$  is close in spirit to our  $\text{QCTL}_{i,\subseteq}^*$ , and is more appropriate as a tool than as a high-level specification logic like  $\text{SL}_{ii}$ . Furthermore, by means of a natural translation we derive that  $\text{CL}$  is strictly included in the hierarchical instances of  $\text{SL}_{ii}$  (Section II-E). In particular, we find that hierarchical instances of  $\text{SL}_{ii}$  can express non-observable goals, while  $\text{CL}$  does not. Non-observable goals arise naturally in problems in distributed synthesis [5].

Finally, our logic  $\text{SL}_{ii}$  is the first generalisation of (the syntax and semantics of)  $\text{SL}$  to include strategies with partial observation, and, unlike  $\text{CL}$ , generalises previous logics with partial-observation strategies, i.e.,  $\text{ATL}_{i,R}^*$  [2] and  $\text{ATL}_{sc,i}^*$  [3]. A comparison of  $\text{SL}_{ii}$  to  $\text{SL}$ ,  $\text{CL}$ , and  $\text{ATL}_{sc,i}^*$  is given in Section II-E.

**Plan.** The definition of  $\text{SL}_{ii}$  and of hierarchical instances, and the discussion about Nash equilibria, are in Section II. The definition of  $\text{QCTL}_{ii}^*$ , and its hierarchical fragment  $\text{QCTL}_{i,\subseteq}^*$ , are in Section III. The proof that model checking  $\text{QCTL}_{i,\subseteq}^*$  is decidable, including the required automata preliminaries, are in Section IV. The translation of  $\text{SL}_{ii}$  into  $\text{QCTL}_{ii}^*$ , and the fact that this preserves hierarchy, are in Section V. Missing details are in the Appendix.

## II. SL WITH IMPERFECT INFORMATION

In this section we introduce  $\text{SL}_{\text{ii}}$ , an extension of  $\text{SL}$  [1] to the imperfect-information setting with synchronous perfect-recall. Our logic presents two original features: first, observations are not bound to players (as is done in extensions of  $\text{ATL}$  by imperfect information [20] or logics for reasoning about knowledge [21]), and second, we have syntactic observations in the language, which need to be interpreted.

We follow the presentation of  $\text{SL}$  in [1], except that we make some changes that simplify their presentation but do not change their semantics, and some that allow us to capture imperfect information. We introduce the syntax and semantics of  $\text{SL}_{\text{ii}}$ , carefully detailing these changes. We first fix some notation used throughout the paper.

### A. Notation

Let  $\Sigma$  be an alphabet. A *finite* (resp. *infinite*) *word* over  $\Sigma$  is an element of  $\Sigma^*$  (resp.  $\Sigma^\omega$ ). Words are written  $w = w_0w_1w_2\dots$ , i.e., indexing begins with 0. The *length* of a finite word  $w = w_0w_1\dots w_n$  is  $|w| := n + 1$ , and  $\text{last}(w) := w_n$  is its last letter. Given a finite (resp. infinite) word  $w$  and  $0 \leq i \leq |w|$  (resp.  $i \in \mathbb{N}$ ), we let  $w_i$  be the letter at position  $i$  in  $w$ ,  $w_{\leq i}$  is the prefix of  $w$  that ends at position  $i$  and  $w_{\geq i}$  is the suffix of  $w$  that starts at position  $i$ . We write  $w \preccurlyeq w'$  if  $w$  is a prefix of  $w'$ , and  $w^\preccurlyeq$  is the set of finite prefixes of word  $w$ . Finally, the domain of a mapping  $f$  is written  $\text{dom}(f)$ , and for  $n \in \mathbb{N}$  we let  $[n] := \{i \in \mathbb{N} : 1 \leq i \leq n\}$ . The literature sometimes refers to “imperfect information” and sometimes to “partial observation”; we will use the terms interchangeably.

### B. Syntax

The syntax of  $\text{SL}_{\text{ii}}$  is similar to that of strategy logic  $\text{SL}$  in [1]: the only difference is that we annotate strategy quantifiers  $\langle\!\langle x \rangle\!\rangle$  by *observation symbols*  $o$ . For the rest of the paper, for convenience we fix a number of parameters for our logics and models: AP is a finite set of *atomic propositions*, Ag is a finite set of *players*, Var is a finite set of *variables* and Obs is a finite set of *observation symbols*. When we consider model-checking problems, these data are implicitly part of the input.

**Definition 1** ( $\text{SL}_{\text{ii}}$  Syntax). *The syntax of  $\text{SL}_{\text{ii}}$  is defined by the following grammar:*

$$\varphi := p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U}\varphi \mid \langle\!\langle x \rangle\!\rangle^o \varphi \mid (a, x)\varphi$$

where  $p \in \text{AP}$ ,  $x \in \text{Var}$ ,  $o \in \text{Obs}$  and  $a \in \text{Ag}$ .

We use abbreviations  $\top := p \vee \neg p$ ,  $\perp := \neg\top$ ,  $\varphi \rightarrow \varphi' := \neg\varphi \vee \varphi'$ ,  $\varphi \leftrightarrow \varphi' := \varphi \rightarrow \varphi' \wedge \varphi' \rightarrow \varphi$  for boolean connectives,  $\mathbf{F}\varphi := \top \mathbf{U}\varphi$ ,  $\mathbf{G}\varphi := \neg\mathbf{F}\neg\varphi$  for temporal operators, and finally  $\llbracket x \rrbracket^o \varphi := \neg\langle\!\langle x \rangle\!\rangle^o \neg\varphi$ .

The notion of free variables and sentences are defined as for  $\text{SL}$ . We recall these for completeness (formal definitions are in the appendix). A variable  $x$  appears *free* in a formula  $\varphi$  if it appears out of the scope of a strategy quantifier, and a player  $a$  appears free in  $\varphi$  if a temporal operator (either  $\mathbf{X}$  or  $\mathbf{U}$ ) appears in  $\varphi$  out of the scope of any binding for player  $a$ . We let  $\text{free}(\varphi)$  be the set of variables and players that appear free in  $\varphi$ . If  $\text{free}(\varphi)$  is empty,  $\varphi$  is a *sentence*.

### C. Semantics

The models of  $\text{SL}_{\text{ii}}$  are like those of  $\text{SL}$ , i.e., concurrent game structures, but extended by a finite set of observations Obs and, for each  $o \in \text{Obs}$ , by an equivalence-relation  $\mathcal{O}(o)$  over positions that represents what a player using a strategy with that observation can see. That is,  $\mathcal{O}(o)$ -equivalent positions are indistinguishable to a player using a strategy associated with observation  $o$ .

**Definition 2** ( $\text{CGS}_{\text{ii}}$ ). *A concurrent game structure with imperfect information (or  $\text{CGS}_{\text{ii}}$  for short)  $\mathcal{G} = (\text{Ac}, V, E, \ell, v_\iota, \mathcal{O})$  where*

- $\text{Ac}$  is a finite non-empty set of actions,
- $V$  is a finite non-empty set of positions,
- $E : V \times \text{Ac}^{\text{Ag}} \rightarrow V$  is a transition function,
- $\ell : V \rightarrow 2^{\text{AP}}$  is a labelling function,
- $v_\iota \in V$  is an initial position,
- $\mathcal{O} : \text{Obs} \rightarrow V \times V$  is an observation interpretation mapping observations to equivalence relations on positions.

We may write  $\sim_o$  for  $\mathcal{O}(o)$ , and  $v \in \mathcal{G}$  for  $v \in V$ .

The notions of joint actions, plays, strategies and assignments are similar to those for  $\text{SL}$ . We will recall these for completeness. Since the main difference between the semantics of  $\text{SL}$  and  $\text{SL}_{\text{ii}}$  is in the strategy-quantification case where we require strategies to be consistent with observations, we define synchronous perfect recall and  $o$ -strategies before defining the semantics of  $\text{SL}_{\text{ii}}$ . Finally, we mention that we make some (inconsequential) simplifications to the semantics of  $\text{SL}$  as proposed in [1]: i) we dispense with partial assignments and only consider complete assignments, ii) our semantics are w.r.t. a finite play instead of a position (in the Appendix we prove that these simplifications do not change the expressive power of  $\text{SL}$ ).

**Joint actions.** In a position  $v \in V$ , each player  $a$  chooses an action  $c_a \in \text{Ac}$ , and the game proceeds to position  $E(v, c)$ , where  $c \in \text{Ac}^{\text{Ag}}$  stands for the *joint action*  $(c_a)_{a \in \text{Ag}}$ . Given a joint action  $c = (c_a)_{a \in \text{Ag}}$  and  $a \in \text{Ag}$ , we let  $c_a$  denote  $c_a$ . For each position  $v \in V$ ,  $\ell(v)$  is the set of atomic propositions that hold in  $v$ .

**Plays and strategies.** A *finite* (resp. *infinite*) *play* is a finite (resp. infinite) word  $\rho = v_0 \dots v_n$  (resp.  $\pi = v_0 v_1 \dots$ ) such that  $v_0 = v_\iota$  and for all  $i$  with  $0 \leq i < |\rho| - 1$  (resp.  $i \geq 0$ ), there exists a joint action  $c$  such that  $E(v_i, c) = v_{i+1}$ . We let Plays be the set of finite plays. A *strategy* is a function  $\sigma : \text{Plays} \rightarrow \text{Ac}$ , and the set of all strategies is denoted  $\text{Str}$ .

**Assignments.** An *assignment* is a function  $\chi : \text{Ag} \cup \text{Var} \rightarrow \text{Str}$ , assigning a strategy to each player and variable. For an assignment  $\chi$ , player  $a$  and strategy  $\sigma$ ,  $\chi[a \mapsto \sigma]$  is the assignment that maps  $a$  to  $\sigma$  and is equal to  $\chi$  on the rest of its domain, and  $\chi[x \mapsto \sigma]$  is defined similarly, where  $x$  is a variable.

**Outcomes.** For an assignment  $\chi$  and a finite play  $\rho$ , we let  $\text{out}(\chi, \rho)$  be the only infinite play that starts with  $\rho$  and is then extended by letting players follow the strategies assigned by  $\chi$ .

Formally,  $\text{out}(\chi, \rho) := \rho \cdot v_1 v_2 \dots$  where, for all  $i \geq 0$ ,  $v_{i+1} = E(v_i, c)$ , where  $v_0 = \text{last}(\rho)$  and  $c = (\chi(a)(\rho \cdot v_1 \dots v_i))_{a \in \text{Ag}}$ .

**Synchronous perfect recall.** In this work we consider players with *synchronous perfect recall*, meaning that each player remembers the whole history of a play, a classic assumption in games with imperfect information and logics of knowledge and time. Each observation relation is thus extended to finite plays as follows:  $\rho \sim_o \rho'$  if  $|\rho| = |\rho'|$  and  $\rho_i \sim_o \rho'_i$  for every  $i \in \{0, \dots, |\rho| - 1\}$ . For  $o \in \text{Obs}$ , an  $o$ -strategy is a strategy  $\sigma : V^+ \rightarrow \text{Ac}$  such that  $\sigma(\rho) = \sigma(\rho')$  whenever  $\rho \sim_o \rho'$ . The latter constraint captures the essence of imperfect information, which is that players can base their strategic choices only on the information available to them. For  $o \in \text{Obs}$  we let  $\text{Str}_o$  be the set of all  $o$ -strategies.

**Definition 3** ( $\text{SL}_{ii}$  Semantics). *The semantics  $\mathcal{G}, \chi, \rho \models \varphi$  is defined inductively, where  $\varphi$  is an  $\text{SL}_{ii}$ -formula,  $\mathcal{G} = (\text{Ac}, V, E, \ell, v_\iota, \mathcal{O})$  is a CGS<sub>ii</sub>,  $\rho$  is a finite play, and  $\chi$  is an assignment:*

$$\begin{aligned} \mathcal{G}, \chi, \rho \models p &\quad \text{if } p \in \ell(\text{last}(\rho)) \\ \mathcal{G}, \chi, \rho \models \neg\varphi &\quad \text{if } \mathcal{G}, \chi, \rho \not\models \varphi \\ \mathcal{G}, \chi, \rho \models \varphi \vee \varphi' &\quad \text{if } \mathcal{G}, \chi, \rho \models \varphi \text{ or } \mathcal{G}, \chi, \rho \models \varphi' \\ \mathcal{G}, \chi, \rho \models \langle\langle x \rangle\rangle^o \varphi &\quad \text{if } \exists \sigma \in \text{Str}_o \text{ s.t. } \mathcal{G}, \chi[x \mapsto \sigma], \rho \models \varphi \\ \mathcal{G}, \chi, \rho \models (a, x)\varphi &\quad \text{if } \mathcal{G}, \chi[a \mapsto \chi(x)], \rho \models \varphi \end{aligned}$$

and, writing  $\pi = \text{out}(\chi, \rho)$ :

$$\begin{aligned} \mathcal{G}, \chi, \rho \models \mathbf{X}\varphi &\quad \text{if } \mathcal{G}, \chi, \pi^{\leq |\rho|} \models \varphi \\ \mathcal{G}, \chi, \rho \models \varphi \mathbf{U} \varphi' &\quad \text{if } \exists i \geq 0 \text{ s.t. } \mathcal{G}, \chi, \pi^{\leq |\rho| + i - 1} \models \varphi' \\ &\quad \text{and } \forall j \text{ s.t. } 0 \leq j < i, \\ &\quad \mathcal{G}, \chi, \pi^{\leq |\rho| + j - 1} \models \varphi \end{aligned}$$

To explain the temporal operators, we remind the reader that positions begin at 0 and thus  $\pi_n$  is the  $(n+1)$ -st position of  $\pi$ . The satisfaction of a sentence is independent of the assignment (the easy proof is in the Appendix). For an  $\text{SL}_{ii}$  sentence  $\varphi$  we thus let  $\mathcal{G}, \rho \models \varphi$  if  $\mathcal{G}, \chi, \rho \models \varphi$  for some assignment  $\chi$ , and we write  $\mathcal{G} \models \varphi$  if  $\mathcal{G}, v_\iota \models \varphi$ .

#### D. Model checking and hierarchical instances

**Model Checking.** We now introduce the main decision problem of this paper, i.e., the model-checking problem for  $\text{SL}_{ii}$ . An  $\text{SL}_{ii}$ -instance is a formula/model pair  $(\Phi, \mathcal{G})$  where  $\Phi \in \text{SL}_{ii}$  and  $\mathcal{G}$  is a CGS<sub>ii</sub>. The *model-checking problem* for  $\text{SL}_{ii}$  is the decision problem that, given an  $\text{SL}_{ii}$ -instance  $(\Phi, \mathcal{G})$ , returns ‘yes’ if  $\mathcal{G} \models \Phi$ , and ‘no’ otherwise.

It is well known that deciding the existence of winning strategies in multi-player games with imperfect information is undecidable for reachability objectives [15]. Since this problem is easily reduced to the model-checking problem for  $\text{SL}_{ii}$ , we get the following result.

**Theorem 1.** *The model-checking problem for  $\text{SL}_{ii}$  is undecidable.*

**Hierarchical instances.** We now isolate a sub-problem obtained by restricting attention to *hierarchical instances*. Intuitively, an  $\text{SL}_{ii}$ -instance  $(\Phi, \mathcal{G})$  is hierarchical if, as one goes down a path in the syntactic tree of  $\Phi$ , the observations tied

to quantifications become finer. We now make these notions precise.

**Definition 4** (Hierarchical instances). *An  $\text{SL}_{ii}$ -instance  $(\Phi, \mathcal{G})$  is hierarchical if for all subformulas  $\varphi_1, \varphi_2$  of  $\Phi$  of the form  $\varphi_2 = \langle\langle x \rangle\rangle^{o_2} \varphi'_2$  and  $\varphi_1 = \langle\langle y \rangle\rangle^{o_1} \varphi'_1$  where  $\varphi_1$  is a subformula of  $\varphi'_2$ , it holds that  $\mathcal{O}(o_1) \subseteq \mathcal{O}(o_2)$ .*

If  $\mathcal{O}(o_1) \subseteq \mathcal{O}(o_2)$  we say that  $o_1$  is *finer* than  $o_2$  in  $\mathcal{G}$ , and that  $o_2$  is *coarser* than  $o_1$  in  $\mathcal{G}$ . Intuitively, this means that a player with observation  $o_1$  observes game  $\mathcal{G}$  no worse than, i.e., is not less informed, a player with observation  $o_2$ .

**Example 1** (Security levels). *We illustrate hierarchical instances in a “security levels” scenario, e.g., higher levels have access to more data (i.e., can observe more). Assume that the CGS<sub>ii</sub>  $\mathcal{G}$  has  $\mathcal{O}(o_3) \subseteq \mathcal{O}(o_2) \subseteq \mathcal{O}(o_1)$  (i.e., level 3 has the highest security clearance, while level 1 has the lowest). Let  $\varphi = (a_1, x_1)(a_2, x_2)(a_3, x_3)\mathbf{G}\varphi$ . The  $\text{SL}_{ii}$  formula  $\Phi := \langle\langle x_1 \rangle\rangle^{o_1} \llbracket x_2 \rrbracket^{o_2} \langle\langle x_3 \rangle\rangle^{o_3} \varphi$  forms a hierarchical instance when paired with  $\mathcal{G}$ . It expresses that the player  $a_1$  (with lowest clearance) can collude with player  $a_3$  (having the highest clearance) to ensure a safety property  $p$ , even in the presence of an adversary  $a_2$  (with intermediate clearance), as long as the strategy used by  $a_3$  can also depend on the strategy used by  $a_2$ .*

*On the other hand, the similar formula  $\langle\langle x_1 \rangle\rangle^{o_1} \langle\langle x_3 \rangle\rangle^{o_3} \llbracket x_2 \rrbracket^{o_2} \varphi$ , which implies that the strategy used by  $a_3$  cannot depend on the adversarial strategy used by  $a_2$ , does not form a hierarchical instance when paired with  $\mathcal{G}$ .*

Here is the main contribution of this paper:

**Theorem 2.** *The model-checking problem for  $\text{SL}_{ii}$  restricted to the class of hierarchical instances is decidable.*

This is proved in Section V by reducing it to the model-checking problem of the hierarchical fragment of a logic called QCTL<sub>ii</sub><sup>\*</sup>, which we introduce, and prove decidable, in Section III. We now give an important corollary of the main theorem.

A Nash equilibrium in a game is a tuple of strategies such that no player has the incentive to deviate. Assuming that goals are written in  $\text{SL}_{ii}$ , say  $\text{goal}_i$  for  $i \in \text{Ag}$ , and  $\text{Ag} = \{a_i : i \in [n]\}$ , the following formula of  $\text{SL}_{ii}$  expresses the existence of a Nash equilibrium:

$$\Phi_{\text{NE}} := \langle\langle x_1 \rangle\rangle^{o_1} \dots \langle\langle x_n \rangle\rangle^{o_n} (a_1, x_1) \dots (a_n, x_n) \Psi_{\text{NE}}$$

where  $\Psi_{\text{NE}} := \bigwedge_{i \in [n]} ((\langle\langle y_i \rangle\rangle^{o_i} (a_i, y_i) \text{goal}_i) \rightarrow \text{goal}_i)$ .

A CGS<sub>ii</sub>  $\mathcal{G}$  is said to yield *hierarchical observation* [13] if the “finer-than” relation is a total ordering, i.e., if for all  $o, o' \in \text{Obs}$ , either  $\mathcal{O}(o) \subseteq \mathcal{O}(o')$  or  $\mathcal{O}(o') \subseteq \mathcal{O}(o)$ .

Note that the instance  $(\Phi_{\text{NE}}, \mathcal{G})$  is *not* hierarchical (unless  $\mathcal{O}(o_i) = \mathcal{O}(o_j)$  for all  $i, j \in \text{Ag}$ ). Nonetheless, we can decide if a game that yields hierarchical observation has a Nash equilibrium:

**Corollary 1.** *The following problem is decidable: given a CGS<sub>ii</sub> that yields hierarchical observation, whether  $\mathcal{G} \models \Phi_{\text{NE}}$ .*

*Proof.* The main idea is to use the fact that in a one-player game of partial-observation (such a game occurs when all but one player have fixed their strategies, as in the definition of Nash equilibrium), the player has a strategy enforcing some goal iff the player has a uniform-strategy enforcing that goal. Here are the details. Let  $\mathcal{G} = (\text{Ac}, V, E, \ell, v_\ell, \mathcal{O})$  be a CGS<sub>ii</sub> that yields hierarchical observation. Suppose the observation set is Obs. To decide if  $\mathcal{G} \models \Phi_{\text{NE}}$  first build a new CGS<sub>ii</sub>  $\mathcal{G}' = (\text{Ac}, V, E, \ell, v_\ell, \mathcal{O}')$  over observations Obs' := Obs  $\cup \{o_p\}$  such that  $\mathcal{O}'(o) = \mathcal{O}(o)$  and  $\mathcal{O}'(o_p) = \{(v, v) : v \in V\}$ , and consider the sentence

$$\Phi' := \langle\langle x_1 \rangle\rangle^{o_1} \dots \langle\langle x_n \rangle\rangle^{o_n} (a_1, x_1) \dots (a_n, x_n) \Psi'$$

where  $\Psi' := \bigwedge_{i \in [n]} [\langle\langle y_i \rangle\rangle^{o_p} (a_i, y_i) \text{goal}_i] \rightarrow \text{goal}_i]$ .

Then  $(\Phi', \mathcal{G}')$  is a hierarchical instance, and by Theorem 2 we can decide  $\mathcal{G}' \models \Phi'$ . We claim that  $\mathcal{G}' \models \Phi'$  iff  $\mathcal{G} \models \Phi_{\text{NE}}$ . To see this, it is enough to establish that:

$$\mathcal{G}', \chi, v_\ell \models \langle\langle y_i \rangle\rangle^{o_p} (a_i, y_i) \text{goal}_i \leftrightarrow \langle\langle y_i \rangle\rangle^{o_i} (a_i, y_i) \text{goal}_i,$$

for every  $i \in [n]$  and every assignment  $\chi$  such that  $\chi(x_i) = \chi(a_i)$  is an  $o_i$ -uniform strategy.

To this end, fix  $i$  and  $\chi$ . The right-to-left implication is immediate (since  $o_p$  is finer than  $o_i$ ). For the converse, let  $\sigma$  be a  $p$ -uniform strategy (i.e., perfect-information) such that  $\mathcal{G}', \chi[y_i \mapsto \sigma, a_i \mapsto \sigma], v_\ell \models \text{goal}_i$ . Let  $\pi := \text{out}(\chi[y_i \mapsto \sigma, a_i \mapsto \sigma], v_\ell)$ . Construct an  $o_i$ -uniform strategy  $\sigma'$  that agrees with  $\sigma$  on prefixes of  $\pi$ . This can be done as follows: if  $\rho \sim_{o_i} \pi_{\leq j}$  for some  $j$  then define  $\sigma'(\rho) = \sigma(\pi_{\leq j})$  (note that this is well-defined since if there is some such  $j$  then it is unique), and otherwise define  $\sigma'(\rho) = a$  for some fixed action  $a \in \text{Ac}$ .  $\square$

#### E. Comparison with other logics

The main difference between SL and ATL-like strategic logics is that in the latter a strategy is always bound to some player, while in the former bindings and quantifications are separated. This separation adds expressive power, e.g., one can bind the same strategy to different players. Extending ATL with imperfect-information is done by giving each player an indistinguishability relation that its strategies must respect [2]. Our extension of SL by imperfect information, instead, assigns each strategy  $x$  an indistinguishability relation  $o$  when it is quantified  $\langle\langle x \rangle\rangle^o$ . Thus  $\langle\langle x \rangle\rangle^o \varphi$  means ‘‘there exists a strategy with observation  $o$  such that  $\varphi$  holds’’. Associating observations in this way, i.e., to strategies rather than players has two consequences. First, it is a clean generalisation of SL in the perfect information setting [1]. Define the *perfect-information fragment of SL<sub>ii</sub>* to be the logic SL<sub>ii</sub> assuming that Obs =  $\{o\}$  and  $\mathcal{O}(o) = \{(v, v) : v \in \mathcal{G}\}$ . The next proposition says that the perfect-information fragment of SL<sub>ii</sub> is a notational variant of SL [1] (the proof is in the Appendix).

**Proposition 1.** *For every SL sentence  $\varphi$  there is an SL<sub>ii</sub> sentence  $\varphi'$  with Obs =  $\{o\}$ , such that for every CGS  $\mathcal{G}$  there is a CGS<sub>ii</sub>  $\mathcal{G}'$  with  $\mathcal{O}(o) = \{(v, v) : v \in \mathcal{G}\}$  such that  $\mathcal{G} \models \varphi$  iff  $\mathcal{G}' \models \varphi'$ .*

Second, SL<sub>ii</sub> subsumes imperfect-information extensions of ATL<sup>\*</sup> that associate observations to players.

**Proposition 2.** *For every ATL<sub>i,R</sub><sup>\*</sup> formula<sup>1</sup>  $\varphi$  there is an SL<sub>ii</sub> formula  $\varphi'$  such that for every CGS<sub>ii</sub>  $\mathcal{G}$  there is a CGS<sub>ii</sub>  $\mathcal{G}'$  such that  $\mathcal{G} \models \varphi$  iff  $\mathcal{G}' \models \varphi'$ .*

We recall that an ATL<sub>i,R</sub><sup>\*</sup> formula  $\langle A \rangle \psi$  reads as ‘‘there are strategies for players in  $A$  such that  $\psi$  holds whatever players in  $\text{Ag} \setminus A$  do’’. Formula  $\varphi'$  is built from  $\varphi$  by replacing each subformula of the form  $\langle A \rangle \psi$ , where  $A = \{a_1, \dots, a_k\} \subset \text{Ag}$  is a coalition of players and  $\text{Ag} \setminus A = \{a_{k+1}, \dots, a_n\}$ , with formula  $\langle\langle x_1 \rangle\rangle^{o_1} \dots \langle\langle x_k \rangle\rangle^{o_k} \llbracket x_{k+1} \rrbracket^{o_p} \dots \llbracket x_n \rrbracket^{o_p} (a_1, x_1) \dots (a_n, x_n) \psi'$ , where  $\psi'$  is the translation of  $\psi$ . Then  $\mathcal{G}'$  is obtained from  $\mathcal{G}$  by interpreting each  $o_i$  as the equivalence relation for player  $i$  in  $\mathcal{G}$ , and interpreting  $o_p$  as the identity relation.

Third, SL<sub>ii</sub> also subsumes the imperfect-information extension of ATL<sup>\*</sup> with strategy context (see [3] for the definition of ATL<sub>sc</sub><sup>\*</sup> with partial observation, which we refer to as ATL<sub>sc,i</sub><sup>\*</sup>).

**Proposition 3.** *For every ATL<sub>sc,i</sub><sup>\*</sup> formula  $\varphi$  there is an SL<sub>ii</sub> formula  $\varphi'$  such that for every CGS<sub>ii</sub>  $\mathcal{G}$  there is a CGS<sub>ii</sub>  $\mathcal{G}'$  such that  $\mathcal{G} \models \varphi$  iff  $\mathcal{G}' \models \varphi'$ .*

The only difference between ATL<sub>sc,i</sub><sup>\*</sup> and ATL<sub>i,R</sub><sup>\*</sup> is the following: in ATL<sub>i,R</sub><sup>\*</sup>, when a subformula of the form  $\langle A \rangle \psi$  is met, we quantify existentially on strategies for players in  $A$ , and then we consider all possible outcomes obtained by letting other players behave however they want. Therefore, if any player in  $\text{Ag} \setminus A$  had previously been assigned a strategy, it is forgotten. In ATL<sub>sc,i</sub><sup>\*</sup> on the other hand, these strategies are stored in a *strategy context*, which is a *partial* assignment  $\chi$ , defined for the subset of players currently bound to a strategy. A strategy context allows one to quantify universally only on strategies of players who are not in  $A$  and who are not already bound to a strategy. It is then easy to adapt the translation presented for Proposition 2 by parameterising it with a strategy context.  $\mathcal{G}'$  is defined as for Proposition 2.

Fourth, there is a natural and simple translation of instances of the model-checking problem of CL [19] into the hierarchical instances of SL<sub>ii</sub>. Moreover, the image of this translation consists of instances of SL<sub>ii</sub> with a very restricted form, i.e., atoms mentioned in the SL<sub>ii</sub>-formula are observable (by all observations of the CGS<sub>ii</sub>), i.e.,  $v \sim_o v'$  implies  $p \in \ell(v) \leftrightarrow p \in \ell(v')$ .

**Proposition 4.** *There is an effective translation that, given a CL-instance  $(\mathcal{S}, \varphi)$  produces a hierarchical SL<sub>ii</sub>-instance  $(\mathcal{G}, \Phi)$  such that*

- 1)  $\mathcal{S} \models \varphi$  iff  $\mathcal{G} \models \Phi$ ,
- 2) *For all atoms  $p$  in  $\Phi$ , and all observations  $o \in \text{Obs}$ , we have that  $v \sim_o v'$  implies  $p \in \ell(v) \leftrightarrow p \in \ell(v')$ .*

To do this, one first translates CL into (hierarchical) QCTL<sub>ii</sub><sup>\*</sup>, the latter is defined in the next section. This step is a simple

<sup>1</sup>See [2] for the definition of ATL<sub>i,R</sub><sup>\*</sup>, where subscript i refers to ‘‘imperfect information’’ and subscript R to ‘‘perfect recall’’. Also, we consider the so-called *objective semantics* for ATL<sub>i,R</sub><sup>\*</sup>.

reflection of the semantics of CL in that of  $\text{QCTL}_{\text{ii}}^*$ . Then one translates  $\text{QCTL}_{\text{ii}}^*$  into  $\text{SL}_{\text{ii}}$  by a simple adaptation of the translation of  $\text{QCTL}^*$  into  $\text{ATL}_{\text{sc}}^*$  [22]. Details are in the Appendix.

### III. QCTL $^*$ WITH IMPERFECT INFORMATION

In this section we introduce an imperfect-information extension of  $\text{QCTL}^*$  [14], [23]–[27]. In order to introduce imperfect information, instead of considering equivalence relations between states as in concurrent game structures, we will enrich Kripke structures by giving internal structure to their states, i.e., we see states as  $n$ -tuples of local states. This way of modelling imperfect information is inspired from Reif’s multiplayer game structures [15] and distributed systems [16], and we find it very suitable to application of automata techniques, as discussed in Section III-C.

The syntax of  $\text{QCTL}_{\text{ii}}^*$  is similar to that of  $\text{QCTL}^*$ , except that we annotate second-order quantifiers by subsets  $\mathbf{o} \subseteq [n]$ . The idea is that quantifiers annotated by  $\mathbf{o}$  can only “observe” the local states indexed by  $i \in \mathbf{o}$ . We define the tree-semantics of  $\text{QCTL}_{\text{ii}}^*$ : this means that we interpret formulas on trees that are the unfoldings of Kripke structures (this will capture the fact that players in  $\text{SL}_{\text{ii}}$  have synchronous perfect recall).

We then define the syntactic class of *hierarchical formulas* and prove, using an automata-theoretic approach, that model checking this class of formulas is decidable.

#### A. QCTL $_{\text{ii}}^*$ Syntax

The syntax of  $\text{QCTL}_{\text{ii}}^*$  is very similar to that of  $\text{QCTL}^*$ : the only difference is that we annotate quantifiers by a set of indices that defines the “observation” of that quantifier.

**Definition 5** ( $\text{QCTL}_{\text{ii}}^*$  Syntax). Fix  $n \in \mathbb{N}$ . The syntax of  $\text{QCTL}_{\text{ii}}^*$  is defined by the following grammar:

$$\begin{aligned}\varphi &:= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{E}\psi \mid \exists^{\mathbf{o}} p. \varphi \\ \psi &:= \varphi \mid \neg\psi \mid \psi \vee \psi \mid \mathbf{X}\psi \mid \psi \mathbf{U}\psi\end{aligned}$$

where  $p \in \text{AP}$  and  $\mathbf{o} \subseteq [n]$ .

Formulas of type  $\varphi$  are called *state formulas*, those of type  $\psi$  are called *path formulas*, and  $\text{QCTL}_{\text{ii}}^*$  consists of all the state formulas defined by the grammar. We use standard abbreviation  $\top := p \vee \neg p$ ,  $\perp := \neg\top$ ,  $\mathbf{F}\psi := \top \mathbf{U}\psi$ ,  $\mathbf{G}\psi := \neg\mathbf{F}\neg\psi$  and  $\mathbf{A}\psi := \neg\mathbf{E}\neg\psi$ . We use  $\exists p. \varphi$  as a shorthand for  $\exists^{[n]} p. \varphi$ . Finally we let  $\forall p. \varphi := \neg\exists p. \neg\varphi$ .

Given a  $\text{QCTL}_{\text{ii}}^*$  formula  $\varphi$ , we define the set of *quantified propositions*  $\text{AP}_{\exists}(\varphi) \subseteq \text{AP}$  as the set of atomic propositions  $p$  such that  $\varphi$  has a subformula of the form  $\exists^{\mathbf{o}} p. \varphi$ . We also define the set of *free propositions*  $\text{AP}_f(\varphi) \subseteq \text{AP}$  as the set of atomic propositions that appear out of the scope of any quantifier of the form  $\exists^{\mathbf{o}} p$ . Observe that  $\text{AP}_{\exists}(\varphi) \cap \text{AP}_f(\varphi)$  may not be empty in general, i.e., a proposition may appear both free and quantified in (different places of) a formula.

#### B. QCTL $_{\text{ii}}^*$ tree-semantics

We define the semantics on structures whose states are tuples of local states.

**Local states.** Let  $\{L_i\}_{i \in [n]}$  denote  $n \in \mathbb{N}$  disjoint finite sets of *local states*. For  $I \subseteq [n]$ , we let  $L_I := \prod_{i \in I} L_i$  if  $I \neq \emptyset$ , and  $L_\emptyset := \{\mathbf{0}\}$  where  $\mathbf{0}$  is a special symbol.

**Concrete observations.** A set  $\mathbf{o} \subseteq [n]$  is called a *concrete observation* (to distinguish it from observations  $o$  in the definitions of  $\text{SL}_{\text{ii}}$ ).

**Compound Kripke structures.** These are like Kripke structures except that the states are elements of  $L_{[n]}$ . A *compound Kripke structure*, or CKS, over AP, is a tuple  $\mathcal{S} = (S, R, s_t, \ell)$  where  $S \subseteq L_{[n]}$  is a set of *states*,  $R \subseteq S \times S$  is a left-total<sup>2</sup> *transition relation*,  $s_t \in S$  is an *initial state*, and  $\ell : S \rightarrow 2^{\text{AP}}$  is a *labelling function*.

A *path* in  $\mathcal{S}$  is an infinite sequence of states  $\lambda = s_0s_1\dots$  such that for all  $i \in \mathbb{N}$ ,  $(s_i, s_{i+1}) \in R$ . For  $s \in S$ , we let  $\text{Paths}(s)$  be the set of all paths that start in  $s$ . A *finite path* is a finite non-empty prefix of a path. We may write  $s \in \mathcal{S}$  for  $s \in S$ . Since we will interpret  $\text{QCTL}_{\text{ii}}^*$  on unfoldings of CKS, we now define infinite trees.

**Trees.** In many works, trees are defined as prefix-closed sets of words with the empty word  $\epsilon$  as root. Here trees represent unfoldings of Kripke structures, and we find it more convenient to see a node  $u$  as a sequence of states and the root as the initial state. Let  $X$  be a finite set (typically a set of states). An  $X$ -tree  $\tau$  is a nonempty set of words  $\tau \subseteq X^+$  such that:

- there exists  $r \in X$ , called the *root* of  $\tau$ , such that each  $u \in \tau$  starts with  $r$  ( $r \preceq u$ );
- if  $u \cdot x \in \tau$  and  $u \neq \epsilon$ , then  $u \in \tau$ , and
- if  $u \in \tau$  then there exists  $x \in X$  such that  $u \cdot x \in \tau$ .

The elements of a tree  $\tau$  are called *nodes*. If  $u \cdot x \in \tau$ , we say that  $u \cdot x$  is a *child* of  $u$ . The *depth* of a node  $u$  is  $|u|$ . An  $X$ -tree  $\tau$  is *complete* if  $u \in \tau, x \in X$  implies  $u \cdot x \in \tau$ . A *path* in  $\tau$  is an infinite sequence of nodes  $\lambda = u_0u_1\dots$  such that for all  $i \in \mathbb{N}$ ,  $u_{i+1}$  is a child of  $u_i$ , and  $\text{Paths}(u)$  is the set of paths that start in node  $u$ . An *AP-labelled X-tree*, or  $(\text{AP}, X)$ -tree for short, is a pair  $t = (\tau, \ell)$ , where  $\tau$  is an  $X$ -tree called the *domain* of  $t$  and  $\ell : \tau \rightarrow 2^{\text{AP}}$  is a *labelling*. For a labelled tree  $t = (\tau, \ell)$  and an atomic proposition  $p \in \text{AP}$ , we define the *p-projection* of  $t$  as the labelled tree  $t \Downarrow_p := (\tau, \ell \Downarrow_p)$ , where for each  $u \in \tau$ ,  $\ell \Downarrow_p(u) := \ell(u) \setminus \{p\}$ . For a set of trees  $\mathcal{L}$ , we let  $\mathcal{L} \Downarrow_p := \{t \Downarrow_p \mid t \in \mathcal{L}\}$ . Finally, two labelled trees  $t = (\tau, \ell)$  and  $t' = (\tau', \ell')$  are *equivalent modulo p*, written  $t \equiv_p t'$ , if  $t \Downarrow_p = t' \Downarrow_p$  (in particular,  $\tau = \tau'$ ).

**Narrowing.** Let  $X$  and  $Y$  be two finite sets, and let  $(x, y) \in X \times Y$ . The  $X$ -narrowing of  $(x, y)$  is  $(x, y) \downarrow_X := x$ . This definition extends naturally to words over  $X \times Y$  (pointwise), and thus to  $X \times Y$ -trees.

For  $J \subseteq I \subseteq [n]$  and  $z = (l_i)_{i \in I} \in L_I$ , we also define  $z \downarrow_J := z \downarrow_{L_J}$ , where  $z$  is seen as a pair  $z = (z_1, z_2) \in L_J \times L_{I \setminus J}$ . This is well defined because having taken sets  $L_i$  to be disjoint, the ordering of local states in  $z$  is indifferent. We also extend this definition to words and trees.

<sup>2</sup>i.e., for all  $s \in S$ , there exists  $s'$  such that  $(s, s') \in R$ .

Observe that when narrowing a tree, nodes with same narrowing are merged. In particular, for every  $L_I$ -tree  $\tau$ ,  $\tau \downarrow \emptyset$  is the only  $L_{\emptyset}$ -tree,  $0^\omega$ .

**Quantification and uniformity.** In  $\text{QCTL}_{ii}^*$  the intuitive meaning of  $\exists^{\mathbf{o}} p. \varphi$  in a tree  $t$  is that there is some equivalent tree  $t'$  modulo  $p$  such that  $t'$  is  $\mathbf{o}$ -uniform in  $p$  and satisfies  $\varphi$ . Intuitively, a tree is  $\mathbf{o}$ -uniform in  $p$  if it is uniformly labelled by  $p$ , i.e., if every two nodes that are indistinguishable when projected onto the local states indexed by  $\mathbf{o} \subseteq [n]$  agree on their labelling of  $p$ .

**Definition 6** ( $\mathbf{o}$ -indistinguishability and  $\mathbf{o}$ -uniformity in  $p$ ). Fix  $\mathbf{o} \subseteq [n]$  and  $I \subseteq [n]$ .

- Two tuples  $x, x' \in L_I$  are  $\mathbf{o}$ -indistinguishable, written  $x \approx_{\mathbf{o}} x'$ , if  $x \downarrow_{I \cap \mathbf{o}} = x' \downarrow_{I \cap \mathbf{o}}$ .
- Two words  $u = u_0 \dots u_i$  and  $u' = u'_0 \dots u'_j$  over alphabet  $L_I$  are  $\mathbf{o}$ -indistinguishable, written  $u \approx_{\mathbf{o}} u'$ , if  $i = j$  and for all  $k \in \{0, \dots, i\}$  we have  $u_k \approx_{\mathbf{o}} u'_k$ .
- A tree  $t$  is  $\mathbf{o}$ -uniform in  $p$  iff for every pair of nodes  $u, u' \in \tau$  such that  $u \approx_{\mathbf{o}} u'$ , we have  $p \in \ell(u)$  iff  $p \in \ell(u')$ .

Finally, we inductively define the satisfaction relation  $\models$  for the semantics on trees, where  $t = (\tau, \ell)$  is a  $2^{\text{AP}}$ -labelled  $L_I$ -tree,  $u$  is a node and  $\lambda$  is a path in  $\tau$ :

$$\begin{aligned}
t, u \models p & \quad \text{if } p \in \ell(u) \\
t, u \models \neg\varphi & \quad \text{if } t, u \not\models \varphi \\
t, u \models \varphi \vee \varphi' & \quad \text{if } t, u \models \varphi \text{ or } t, u \models \varphi' \\
t, u \models \mathbf{E}\psi & \quad \text{if } \exists \lambda \in \text{Paths}(u) \text{ s.t. } t, \lambda \models \psi \\
t, u \models \exists^{\mathbf{o}} p. \varphi & \quad \text{if } \exists t' \equiv_p t \text{ s.t. } t' \text{ is } \mathbf{o}\text{-uniform in } p \text{ and} \\
& \quad t', u \models \varphi. \\
t, \lambda \models \varphi & \quad \text{if } t, \lambda_0 \models \varphi \\
t, \lambda \models \neg\psi & \quad \text{if } t, \lambda \not\models \psi \\
t, \lambda \models \psi \vee \psi' & \quad \text{if } t, \lambda \models \psi \text{ or } t, \lambda \models \psi' \\
t, \lambda \models \mathbf{X}\psi & \quad \text{if } t, \lambda_{\geq 1} \models \psi \\
t, \lambda \models \psi \mathbf{U} \psi' & \quad \text{if } \exists i \geq 0 \text{ s.t. } t, \lambda_{\geq i} \models \psi' \text{ and} \\
& \quad \forall j \text{ s.t. } 0 \leq j < i, t, \lambda_{\geq j} \models \psi
\end{aligned}$$

We write  $t \models \varphi$  for  $t, r \models \varphi$ , where  $r$  is the root of  $t$ .

**Example 2.** Consider the following CTL formula:

$$\text{border}(p) := \mathbf{AF}p \wedge \mathbf{AG}(p \rightarrow \mathbf{AXAG}\neg p).$$

This formula holds in a labelled tree if and only if each path contains exactly one node labelled with  $p$ . Now, consider the following  $\text{QCTL}_{ii}^*$  formula:

$$\text{level}(p) := \exists^{\emptyset} p. \text{border}(p).$$

For a blind quantifier, two nodes of a tree are indistinguishable if and only if they have same depth. Therefore, this formula holds on a tree iff the  $p$ 's label all and only the nodes at some fixed depth. This formula can thus be used to capture the equal level predicate on trees. Actually, just as  $\text{QCTL}^*$  captures MSO, one can prove that  $\text{QCTL}_{ii}^*$  with tree semantics subsumes MSO with equal level [17], [28], [29]. In Theorem 3

we make use of a similar observation to prove that model-checking  $\text{QCTL}_{ii}^*$  is undecidable.

**Model-checking problem for  $\text{QCTL}_{ii}^*$  under tree semantics.** For the model-checking problem, we interpret  $\text{QCTL}_{ii}^*$  on unfoldings of CKSs.

**Tree unfoldings  $t_S(s)$ .** Let  $\mathcal{S} = (S, R, s_i, \ell)$  be a compound Kripke structure over AP, and let  $s \in S$ . The *tree-unfolding of  $\mathcal{S}$  from  $s$*  is the  $(\text{AP}, S)$ -tree  $t_S(s) := (\tau, \ell')$ , where  $\tau$  is the set of all finite paths that start in  $s$ , and for every  $u \in \tau$ ,  $\ell'(u) := \ell(\text{last}(u))$ . Given a CKS  $\mathcal{S}$ , a state  $s \in \mathcal{S}$  and a  $\text{QCTL}_{ii}^*$  formula  $\varphi$ , we write  $\mathcal{S}, s \models \varphi$  if  $t_S(s) \models \varphi$ . Write  $\mathcal{S} \models \varphi$  if  $t_S(s_i) \models \varphi$ .

The *model-checking problem for  $\text{QCTL}_{ii}^*$  under tree-semantics* is the following decision problem: given an instance  $(\varphi, \mathcal{S})$  where  $\mathcal{S}$  is a CKS, and  $\varphi$  is a  $\text{QCTL}_{ii}^*$  formula, return ‘Yes’ if  $\mathcal{S} \models \varphi$  and ‘No’ otherwise.

### C. Discussion of the definition of $\text{QCTL}_{ii}^*$

We now motivate in detail some aspects of  $\text{QCTL}_{ii}^*$ .

**Modelling of imperfect information.** We model imperfect information by means of local states (rather than equivalence relations) since this greatly facilitates the use of automata techniques. More precisely, in our decision procedure of Section IV, we make extensive use of an operation on tree automata called *narrowing*, which was introduced in [30] to deal with imperfect-information in the context of distributed synthesis for temporal specifications. Given an automaton  $\mathcal{A}$  that works on  $X \times Y$ -trees, where  $X$  and  $Y$  are two finite sets, and assuming that we want to model an operation performed on trees while observing only the  $X$  component of each node, this narrowing operation allows one to build from  $\mathcal{A}$  an automaton  $\mathcal{A}'$  that works on  $X$ -trees, such that  $\mathcal{A}'$  accepts an  $X$ -tree if and only if  $\mathcal{A}$  accepts its widening to  $X \times Y$  (see Section IV for details). One can then make this automaton  $\mathcal{A}'$  perform the desired operation, which will by necessity be performed uniformly with regards to the partial observation, since the  $Y$  component is absent from the input trees.

With our definition of compound Kripke structures, their unfoldings are trees over the Cartesian product  $L_{[n]}$ . To model a quantification  $\exists^{\mathbf{o}} p$  with observation  $\mathbf{o} \subseteq [n]$ , we can thus use the narrowing operation to forget about components  $L_i$ , for  $i \in [n] \setminus \mathbf{o}$ . We then use the classic projection of non-deterministic tree automata to perform existential quantification on atomic proposition  $p$ . Since the choice of the  $p$ -labelling is made directly on  $L_{\mathbf{o}}$ -trees, it is necessarily  $\mathbf{o}$ -uniform.

**Choice of the tree semantics.**  $\text{QCTL}^*$  is obtained by adding to  $\text{CTL}^*$  second-order quantification on atomic propositions. Several semantics have been considered. The two most studied ones are the *structure semantics*, in which formulas are evaluated directly on Kripke structures, and the *tree semantics*, in which Kripke structures are first unfolded into infinite trees. Tree semantics thus allows quantifiers to choose the value of a quantified atomic proposition in each *finite path* of the model, while in structure semantics the choice is only made in each state. When  $\text{QCTL}^*$  is used to express existence

of strategies, existential quantification on atomic propositions labels the structure with strategic choices; in this kind of application, structure semantics reflects so-called *positional* or *memoryless* strategies, while tree semantics captures *perfect-recall* or *memoryfull* strategies. Since in this work we are interested in perfect-recall strategies, we only consider the tree semantics.

#### D. Model checking QCTL<sub>ii</sub><sup>\*</sup>

We now prove that the model-checking problem for QCTL<sub>ii</sub><sup>\*</sup> under tree semantics is undecidable. This comes as no surprise since, as we will show, QCTL<sub>ii</sub><sup>\*</sup> can express the existence of winning strategies in imperfect-information games.

**Theorem 3.** *The model-checking problem for QCTL<sub>ii</sub><sup>\*</sup> under tree-semantics is undecidable.*

*Proof.* Let MSO<sub>eq</sub> denote the extension of the logic MSO by a binary predicate symbol eq. Formulas of MSO<sub>eq</sub> are interpreted on trees, and the semantics of eq( $x, y$ ) is that  $x$  and  $y$  have the same depth in the tree. There is a translation of MSO-formulas to QCTL<sup>\*</sup>-formulas that preserves satisfaction [14]. This translation can be extended to map MSO<sub>eq</sub>-formulas to QCTL<sub>ii</sub><sup>\*</sup>-formulas using the formula level( $\cdot$ ) from Example 2 to help capture the equal-length predicate. Our result follows since the MSO<sub>eq</sub>-theory of the binary tree is undecidable [17].  $\square$

## IV. A DECIDABLE FRAGMENT OF QCTL<sub>ii</sub><sup>\*</sup>: HIERARCHY ON OBSERVATIONS

The main result of this section is the identification of an important decidable fragment of QCTL<sub>ii</sub><sup>\*</sup>.

**Definition 7** (Hierarchical formulas). A QCTL<sub>ii</sub><sup>\*</sup> formula  $\varphi$  is hierarchical if for all subformulas  $\varphi_1, \varphi_2$  of the form  $\varphi_1 = \exists^{\mathbf{o}_1} p_1. \varphi'_1$  and  $\varphi_2 = \exists^{\mathbf{o}_2} p_2. \varphi'_2$  where  $\varphi_2$  is a subformula of  $\varphi'_1$ , we have  $\mathbf{o}_1 \subseteq \mathbf{o}_2$ .

In other words, a formula is hierarchical if innermost propositional quantifiers observe at least as much as outermost ones. We let QCTL<sub>i,≤</sub><sup>\*</sup> be the set of hierarchical QCTL<sub>ii</sub><sup>\*</sup> formulas.

**Theorem 4.** *Model checking QCTL<sub>i,≤</sub><sup>\*</sup> under tree semantics is non-elementary decidable.*

Since our decision procedure for the hierarchical fragment of QCTL<sub>ii</sub><sup>\*</sup> is based on an automata-theoretic approach, we recall some definitions and results for alternating tree automata.

#### A. Alternating parity tree automata

We briefly recall the notion of alternating (parity) tree automata. Because it is sufficient for our needs and simplifies definitions, we assume that all input trees are complete trees. For a set  $Z$ ,  $\mathbb{B}^+(Z)$  is the set of formulas built from the elements of  $Z$  as atomic propositions using the connectives  $\vee$  and  $\wedge$ , and with  $\top, \perp \in \mathbb{B}^+(Z)$ . An *alternating tree automaton (ATA)* on  $(AP, X)$ -trees is a structure  $\mathcal{A} = (Q, \delta, q_i, C)$  where  $Q$  is a finite set of states,  $q_i \in Q$  is an initial state,

$\delta : Q \times 2^{AP} \rightarrow \mathbb{B}^+(X \times Q)$  is a transition function, and  $C : Q \rightarrow \mathbb{N}$  is a colouring function. To ease reading we shall write atoms in  $\mathbb{B}^+(X \times Q)$  between brackets, such as  $[x, q]$ . A *nondeterministic tree automaton (NTA)* on  $(AP, X)$ -trees is an ATA  $\mathcal{A} = (Q, \delta, q_i, C)$  such that for every  $q \in Q$  and  $a \in 2^{AP}$ ,  $\delta(q, a)$  is written in disjunctive normal form and for every direction  $x \in X$  each disjunct contains exactly one element of  $\{x\} \times Q$ . Acceptance is defined as usual (see, e.g., [31]), and we let  $\mathcal{L}(\mathcal{A})$  be the set of trees accepted by  $\mathcal{A}$ .

We recall three classic results on tree automata. The first one is that nondeterministic tree automata are closed under projection, and was established by Rabin to deal with second-order monadic quantification:

**Theorem 5** (Projection [32]). *Given an NTA  $\mathcal{N}$  and an atomic proposition  $p \in AP$ , one can build an NTA  $\mathcal{N} \Downarrow_p$  such that  $\mathcal{L}(\mathcal{N} \Downarrow_p) = \mathcal{L}(\mathcal{N}) \Downarrow_p$ .*

Because it will be important to understand the automata construction for our decision procedure in Section IV, we briefly recall that the projected automaton  $\mathcal{N} \Downarrow_p$  is simply automaton  $\mathcal{N}$  with the only difference that when it reads the label of a node, it can choose whether  $p$  is there or not: if  $\delta$  is the transition function of  $\mathcal{N}$ , that of  $\mathcal{N} \Downarrow_p$  is  $\delta'(q, a) = \delta(q, a \cup \{p\}) \vee \delta(q, a \setminus \{p\})$ , for any state  $q$  and label  $a \in 2^{AP}$ . Another way of seeing it is that  $\mathcal{N} \Downarrow_p$  first guesses a  $p$ -labelling for the input tree, and then simulates  $\mathcal{N}$  on this modified input. To prevent  $\mathcal{N} \Downarrow_p$  from guessing different labels for a same node in different executions, it is crucial that  $\mathcal{N}$  be nondeterministic, which is the reason why we need the next classic result: the crucial simulation theorem, due to Muller and Schupp.

**Theorem 6** (Simulation [33]). *Given an ATA  $\mathcal{A}$ , one can build an NTA  $\mathcal{N}$  such that  $\mathcal{L}(\mathcal{N}) = \mathcal{L}(\mathcal{A})$ .*

The third result was established by Kupferman and Vardi to deal with imperfect information aspects in distributed synthesis. To state it we need to define a widening operation on trees which simply expands the directions in a tree.

**Widening** [30]. Let  $X$  and  $Y$  be two finite sets, let  $t$  be an  $X$ -tree with root  $x$ , and let  $y \in Y$ . The  $Y$ -widening of  $t$  with root  $(x, y)$  is the  $X \times Y$ -tree

$$\tau \uparrow_y^Y := \{u \in (x, y) \cdot (X \times Y)^* \mid u \downarrow_X \in \tau\}.$$

For an  $(AP, X)$ -tree  $t = (\tau, \ell)$ , we let  $t \uparrow_y^Y := (\tau \uparrow_y^Y, \ell')$  where  $\ell'(u) := \ell(u \downarrow_X)$ .

Similarly to the narrowing operation, we extend this definition to tuples of local states by letting, for  $J \subseteq I \subseteq [n]$ ,  $\tau$  an  $L_J$ -tree and  $z' \in L_{I \setminus J}$ ,

$$\tau \uparrow_z^I := \tau \uparrow_{z'}^{L_{I \setminus J}},$$

and similarly for a labelled  $L_J$ -tree  $t$ ,

$$t \uparrow_z^I := t \uparrow_{z'}^{L_{I \setminus J}}.$$

Recall that because the sets of local states  $L_i$  are disjoint, the order of local states in a tuple does not matter and we can identify  $L_I$  with  $L_J \times L_{I \setminus J}$ .

The rough idea of the narrowing operation on ATA is that, if one just observes  $X$ , uniform  $p$ -labellings on  $X \times Y$ -trees can be obtained by choosing the labellings directly on  $X$ -trees, and then lifting them to  $X \times Y$ .

**Theorem 7** (Narrowing [30]). *Given an ATA  $\mathcal{A}$  on  $X \times Y$ -trees one can build an ATA  $\mathcal{A} \downarrow_X$  on  $X$ -trees such that for all  $y \in Y$ ,  $t \in \mathcal{L}(\mathcal{A} \downarrow_X)$  iff  $t \uparrow^{X \times Y}_y \in \mathcal{L}(\mathcal{A})$ .*

### B. Translating QCTL<sub>i, $\subseteq$</sub> \* to ATA

In order to prove Theorem 4 we need some more notations and a technical lemma that contains the automata construction.

For every  $\varphi \in \text{QCTL}_{ii}^*$ , we let

$$I_\varphi := \bigcap_{\mathbf{o} \in \text{Obs}(\varphi)} \mathbf{o} \subseteq [n],$$

where  $\text{Obs}(\varphi)$  is the set of concrete observations that occur in  $\varphi$ , with the intersection over the empty set defined as  $[n]$ . We also let  $L_\varphi := L_{I_\varphi}$  (recall that for  $I \subseteq [n]$ ,  $L_I = \prod_{i \in I} L_i$ ).

Our construction, that transforms a QCTL<sub>i, $\subseteq$</sub> \* formula  $\varphi$  and a CKS  $\mathcal{S}$  into an ATA, builds upon the classic construction from [34], which builds hesitant ATA for CTL\* formulas. Since our aim here is to establish decidability and that the hesitant condition is only used to improve complexity, we do not consider it. However we need to develop an original technique to implement quantifiers with imperfect information thanks to automata narrowing and projection.

The classical approach to model checking via tree automata is to build an automaton that accepts all tree models of the input formula, and check whether it accepts the unfolding of the model [34]. We now explain how we adapt this approach.

**Narrowing of non-uniform trees.** Quantification on atomic propositions is classically performed by means of automata projection (see Theorem 5). But in order to obtain a labelling that is uniform with regards to the observation of the quantifier, we need to make use of the narrow operation (see Theorem 7). Intuitively, to check that a formula  $\exists^\mathbf{o} p. \varphi$  holds in a tree  $t$ , we would like to work on its narrowing  $t' := t \downarrow_{\mathbf{o}}$ , guess a labelling for  $p$  on this tree thanks to automata projection, thus obtaining a tree  $t'_p$ , take its widening  $t''_p := t'_p \uparrow^{[n]}$ , obtaining a tree with an  $\mathbf{o}$ -uniform labelling for  $p$ , and then check that  $\varphi$  holds on  $t''_p$ . The problem here is that, while the narrowing  $\tau \downarrow_{\mathbf{o}}$  of an unlabelled tree  $\tau$  is well defined (see Section III-B), that of a labelled tree  $t = (\tau, \ell)$  is undefined: indeed, unless  $t$  is  $\mathbf{o}$ -uniform in every atomic proposition in AP, there is no way to define the labelling of  $\tau \downarrow_{\mathbf{o}}$  without losing information. This implies that we cannot feed (a narrowing of) the unfolding of the model to our automata. Still, we need an input tree to be successively labelled and widened to guess uniform labellings.

**Splitting quantified from free propositions.** To address this problem, we remark that since we are interested in model checking a QCTL<sub>ii</sub>\* formula  $\varphi$  on a CKS  $\mathcal{S}$ , the automaton that we build for  $\varphi$  can depend on  $\mathcal{S}$ . It can thus guess paths in  $\mathcal{S}$ , and evaluate free occurrences of atomic propositions in  $\mathcal{S}$  without reading the input tree. The input tree is thus no longer used to represent the model. However we use it to carry

labellings for quantified propositions  $\text{AP}_\exists(\varphi)$ : we provide the automaton with an input tree whose labelling is initially empty, and the automaton, through successive narrowing and projection operations, decorates it with uniform labellings for quantified atomic propositions.

We remark that this technique allows one to go beyond CL [19]: by separating between quantified atomic propositions (that need to be uniform) and free atomic propositions (that state facts about the model), we manage to remove the restriction present in CL, that requires that all facts about the model are known to every strategy/agent (see the Appendix).

To do this we assume without loss of generality that propositions that are quantified in  $\varphi$  do not appear free in  $\varphi$ , i.e.,  $\text{AP}_\exists(\varphi) \cap \text{AP}_f(\varphi) = \emptyset$ . If necessary, for every  $p \in \text{AP}_\exists(\varphi) \cap \text{AP}_f(\varphi)$ , we take a fresh atomic proposition  $p'$  and replace all quantified occurrences of  $p$  in  $\varphi$  with  $p'$ . We obtain an equivalent formula  $\varphi'$  on  $\text{AP}' := \text{AP} \cup \{p' \mid p \in \text{AP}_\exists(\varphi) \cap \text{AP}_f(\varphi)\}$  such that  $\text{AP}_\exists(\varphi') \cap \text{AP}_f(\varphi') = \emptyset$ . Observe also that given a formula  $\varphi$  such that  $\text{AP}_\exists(\varphi) \cap \text{AP}_f(\varphi) = \emptyset$ , a CKS  $\mathcal{S}$  and a state  $s \in \mathcal{S}$ , the truth value of  $\varphi$  in  $\mathcal{S}, s$  does not depend on the labelling of  $\mathcal{S}$  for atomic propositions in  $\text{AP}_\exists(\varphi)$ , which can thus be forgotten.

As a consequence, henceforth we assume that an instance  $(\varphi, \mathcal{S})$  of the model-checking problem for QCTL<sub>ii</sub>\* is such that  $\text{AP}_\exists(\varphi) \cap \text{AP}_f(\varphi) = \emptyset$ , and  $\mathcal{S}$  is a CKS over  $\text{AP}(\varphi)$ .

**Merging the decorated input tree and the model.** To state the correctness of our construction, we will need to merge the labels for quantified propositions, carried by the input tree, with those for free propositions, carried by CKS  $\mathcal{S}$ . Because, through successive widenings, the input tree (represented by  $t$  in the definition below) will necessarily be a complete tree, its domain will always contain the domain of the unfolding of  $\mathcal{S}$  (represented by  $t'$  below), hence the following definition.

**Definition 8** (Merge). *Let  $t = (\tau, \ell)$  be a complete  $(\text{AP}, X)$ -tree and  $t' = (\tau', \ell')$  an  $(\text{AP}', X)$ -tree, where  $\text{AP} \cap \text{AP}' = \emptyset$ . We define the merge of  $t$  and  $t'$  as the  $(\text{AP} \cup \text{AP}', X)$ -tree  $t \mathbin{\text{M}\kern-1.6ex\text{l}} t' := (\tau \cap \tau' = \tau', \ell'')$ , where  $\ell''(u) = \ell(u) \cup \ell'(u)$ .*

We now describe our automata construction and establish the following lemma, which is our main technical contribution.

**Lemma 1** (Translation). *Let  $(\Phi, \mathcal{S})$  be an instance of the model-checking problem for QCTL<sub>i, $\subseteq$</sub> \*. For every subformula  $\varphi$  of  $\Phi$  and state  $s$  of  $\mathcal{S}$ , one can build an ATA  $\mathcal{A}_s^\varphi$  on  $(\text{AP}_\exists(\Phi), L_\varphi)$ -trees such that for every  $(\text{AP}_\exists(\Phi), L_\varphi)$ -tree  $t$  rooted in  $s \downarrow_{I_\varphi}$ ,*

$$t \in \mathcal{L}(\mathcal{A}_s^\varphi) \text{ iff } t \uparrow_y^{[n]} \mathbin{\text{M}\kern-1.6ex\text{l}} t_{\mathcal{S}}(s) \models \varphi, \quad \text{where } y = s \downarrow_{[n] \setminus I_\varphi}.$$

For an  $L_I$ -tree  $t$ , from now on  $t \uparrow_y^{[n]} \mathbin{\text{M}\kern-1.6ex\text{l}} t_{\mathcal{S}}(s)$  stands for  $t \uparrow_y^{[n]} \mathbin{\text{M}\kern-1.6ex\text{l}} t_{\mathcal{S}}(s)$ , where  $y = s \downarrow_{[n] \setminus I}$ : the missing local states in the root of  $t$  are filled with those from  $s$ .

*Proof sketch of Lemma 1.* Let  $(\Phi, \mathcal{S})$  be an instance of the model-checking problem for QCTL<sub>i, $\subseteq$</sub> \*. Let  $\Phi \in \text{QCTL}_{i,\subseteq}^*$ , and let  $\text{AP}_\exists = \text{AP}_\exists(\Phi)$  and  $\text{AP}_f = \text{AP}_f(\Phi)$ , and recall that  $\mathcal{S}$  is labelled over  $\text{AP}_f$ . For each state  $s \in S$  and each subformula

$\varphi$  of  $\Phi$  (note that all subformulas of  $\Phi$  are also hierarchical), we define by induction on  $\varphi$  the ATA  $\mathcal{A}_s^\varphi$  on  $(\text{AP}_\exists, L_\varphi)$ -trees.

$\varphi = p :$

We let  $\mathcal{A}_s^p$  be the ATA over  $L_{[n]}$ -trees with one unique state  $q_\ell$ , with transition function defined as follows:

$$\delta(q_\ell, a) = \begin{cases} \top & \text{if } p \in \text{AP}_f \text{ and } p \in \ell_S(s) \\ & \quad \text{or} \\ & p \in \text{AP}_\exists \text{ and } p \in a \\ \perp & \text{if } p \in \text{AP}_f \text{ and } p \notin \ell_S(s) \\ & \quad \text{or} \\ & p \in \text{AP}_\exists \text{ and } p \notin a \end{cases}$$

$\varphi = \neg\varphi' :$

We obtain  $\mathcal{A}_s^\varphi$  by dualising  $\mathcal{A}_s^{\varphi'}$ , a classic operation.

$\varphi = \varphi_1 \vee \varphi_2 :$

Because  $I_\varphi = I_{\varphi_1} \cap I_{\varphi_2}$ , and each  $\mathcal{A}_s^{\varphi_i}$  works on  $L_{\varphi_i}$ -trees, we first narrow them so that they work on  $L_\varphi$ -trees: for  $i \in \{1, 2\}$ , we let  $\mathcal{A}_i := \mathcal{A}_s^{\varphi_i} \downarrow_{I_\varphi}$ . We then build  $\mathcal{A}_s^\varphi$  by taking the disjoint union of  $\mathcal{A}_1$  and  $\mathcal{A}_2$  and adding a new initial state that nondeterministically chooses which of  $\mathcal{A}_1$  or  $\mathcal{A}_2$  to execute on the input tree, so that  $\mathcal{L}(\mathcal{A}_s^\varphi) = \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$ .

$\varphi = \mathbf{E}\psi :$

The aim is to build an automaton  $\mathcal{A}_s^\varphi$  that works on  $L_\varphi$ -trees and that on input  $t$ , checks for the existence of a path in  $t \uparrow^{[n]} \wedge t_S(s)$  that satisfies  $\psi$ . Observe that a path in  $t \uparrow^{[n]} \wedge t_S(s)$  is a path in  $t_S(s)$ , augmented with the labelling for atomic propositions in  $\text{AP}_\exists$  carried by  $t$ .

To do so,  $\mathcal{A}_s^\varphi$  guesses a path  $\lambda$  in  $(\mathcal{S}, s)$ . It remembers the current state in  $\mathcal{S}$ , which provides the labelling for atomic propositions in  $\text{AP}_f$ , and while it guesses  $\lambda$  it follows its  $L_\varphi$ -narrowing in its input tree  $t$  (which always exists since input to tree automata are complete trees), reading the labels to evaluate propositions in  $\text{AP}_\exists$ .

Let  $\max(\psi) = \{\varphi_1, \dots, \varphi_n\}$  be the set of maximal state sub-formulas of  $\psi$ . In a first step we see these maximal state sub-formulas as atomic propositions. The formula  $\psi$  can thus be seen as an LTL formula, and we can build a nondeterministic parity word automaton  $\mathcal{W}^\psi = (Q^\psi, \Delta^\psi, q_\ell^\psi, C^\psi)$  over alphabet  $2^{\max(\psi)}$  that accepts exactly the models of  $\psi$  [35].<sup>3</sup> We define the ATA  $\mathcal{A}$  that, given as input a  $(\max(\psi), L_\varphi)$ -tree  $t$ , nondeterministically guesses a path  $\lambda$  in  $t \uparrow^{[n]} \wedge t_S(s)$  and simulates  $\mathcal{W}^\psi$  on it, assuming that the labels it reads while following  $\lambda \downarrow_{I_\varphi}$  in its input correctly represent the truth value of formulas in  $\max(\psi)$  along  $\lambda$ . Recall that  $\mathcal{S} = (S, R, s_\ell, \ell_S)$ ; we define  $\mathcal{A} := (Q, \delta, q_\ell, C)$ , where

- $Q = Q^\psi \times S$ ,
- $q_\ell = (q_\ell^\psi, s)$ ,
- $C(q^\psi, s') = C^\psi(q^\psi)$ , and

<sup>3</sup>Note that, as usual for nondeterministic word automata, we take the transition function of type  $\Delta^\psi : Q^\psi \times 2^{\max(\psi)} \rightarrow 2^{Q^\psi}$ .

- for each  $(q^\psi, s') \in Q$  and  $a \in 2^{\max(\psi)}$ ,

$$\delta((q^\psi, s'), a) = \bigvee_{q' \in \Delta^\psi(q^\psi, a)} \bigvee_{s'' \in R(s')} [s'' \downarrow_{I_\varphi}, (q', s'')].$$

The intuition is that  $\mathcal{A}$  reads the current label, chooses nondeterministically which transition to take in  $\mathcal{W}^\psi$ , chooses a next state in  $\mathcal{S}$  and proceeds in the corresponding direction in  $X_\varphi$ . Thus,  $\mathcal{A}$  accepts a  $\max(\varphi)$ -labelled  $L_\varphi$ -tree  $t$  iff there is a path in  $t$  that is the  $L_\varphi$ -narrowing of some path in  $t_S(s)$ , and that satisfies  $\psi$ , where maximal state formulas are considered as atomic propositions.

Now from  $\mathcal{A}$  we build the automaton  $\mathcal{A}_s^\varphi$  over  $L_\varphi$ -trees labelled with “real” atomic propositions in  $\text{AP}_\exists$ . In each node it visits,  $\mathcal{A}_s^\varphi$  guesses what should be its labelling over  $\max(\psi)$ , it simulates  $\mathcal{A}$  accordingly, and checks that the guess it made is correct. If the path being guessed in  $t_S(s)$  is currently in node  $u$  ending with state  $s'$ , and  $\mathcal{A}_s^\varphi$  guesses that  $\varphi_i$  holds in  $u$ , it checks this guess by starting a simulation of automaton  $\mathcal{A}_{s'}^{\varphi_i}$  from node  $v = u \downarrow_{I_\varphi}$  in its input  $t$ .

For each  $s' \in \mathcal{S}$  and each  $\varphi_i \in \max(\psi)$  we first build  $\mathcal{A}_{s'}^{\varphi_i}$ , and we let  $\mathcal{A}_{s'}^i := \mathcal{A}_{s'}^{\varphi_i} = (Q_{s'}^i, \delta_{s'}^i, q_{s'}^i, C_{s'}^i)$ . We also let  $\overline{\mathcal{A}_{s'}^i} = (\overline{Q_{s'}^i}, \overline{\delta_{s'}^i}, \overline{q_{s'}^i}, \overline{C_{s'}^i})$  be its dualisation, and we assume w.l.o.g. that all the state sets are pairwise disjoint. Observe that because each  $\varphi_i$  is a maximal state sub-formula, we have  $I_{\varphi_i} = I_\varphi$ , so that we do not need to narrow down automata  $\mathcal{A}_{s'}^i$  and  $\overline{\mathcal{A}_{s'}^i}$ . We define the ATA

$$\mathcal{A}_s^\varphi = (Q \cup \bigcup_{i, s'} Q_{s'}^i \cup \overline{Q_{s'}^i}, \delta', q_\ell, C'),$$

where the colours of states are left as they were in their original automaton, and  $\delta$  is defined as follows. For states in  $Q_{s'}^i$  (resp.  $\overline{Q_{s'}^i}$ ),  $\delta$  agrees with  $\delta_{s'}^i$  (resp.  $\overline{\delta_{s'}^i}$ ), and for  $(q^\psi, s') \in Q$  and  $a \in 2^{\text{AP}_\exists}$  we let  $\delta'((q^\psi, s'), a)$  be the disjunction over  $a' \in 2^{\max(\psi)}$  of

$$\left( \delta((q^\psi, s'), a') \wedge \bigwedge_{\varphi_i \in a'} \delta_{s'}^i(q_{s'}^i, a) \wedge \bigwedge_{\varphi_i \notin a'} \overline{\delta_{s'}^i}(\overline{q_{s'}^i}, a) \right).$$

$\varphi = \exists^0 p. \varphi' :$

We build automaton  $\mathcal{A}_s^{\varphi'}$  that works on  $L_{\varphi'}$ -trees; because  $\varphi$  is hierarchical, we have that  $\mathbf{o} \subseteq I_{\varphi'}$  and we can narrow down  $\mathcal{A}_s^{\varphi'}$  to work on  $L_\mathbf{o}$ -trees and obtain  $\mathcal{A}_1 := \mathcal{A}_s^{\varphi'} \downarrow_\mathbf{o}$ . By Theorem 6 we can nondeterminise it to get  $\mathcal{A}_2$ , which by Theorem 5 we can project with respect to  $p$ , finally obtaining  $\mathcal{A}_s^\varphi := \mathcal{A}_2 \Downarrow_p$ .

The proof that the construction is correct can be found in the Appendix.  $\square$

### C. Proof of Theorem 4

We can now prove Theorem 4. Let  $\mathcal{S}$  be a CKS,  $s \in \mathcal{S}$ , and  $\varphi \in \text{QCTL}_{\mathbf{i} \subseteq}^*$ . By Lemma 1 one can build an ATA  $\mathcal{A}_s^\varphi$  such that for every labelled  $L_\varphi$ -tree  $t$  rooted in  $s \downarrow_{I_\varphi}$ , it holds that  $t \in \mathcal{L}(\mathcal{A}_s^\varphi)$  iff  $t \uparrow^{[n]} \wedge t_S(s) \models \varphi$ . Let  $\tau$  be the full  $L_\varphi$ -tree rooted in  $s \downarrow_\varphi$ , and let  $t = (\tau, \ell_\emptyset)$ , where  $\ell_\emptyset$  is the empty labelling. Clearly,  $t \uparrow^{[n]} \wedge t_S(s) = t_S(s)$ , and because  $t$  is rooted in  $s \downarrow_\varphi$ , we have  $t \in \mathcal{L}(\mathcal{A}_s^\varphi)$  iff  $t_S(s) \models \varphi$ . It

only remains to build a simple deterministic tree automaton  $\mathcal{A}$  over  $L_\varphi$ -trees such that  $\mathcal{L}(\mathcal{A}) = \{t\}$ , and check for emptiness of the alternating tree automaton  $\mathcal{L}(\mathcal{A} \cap \mathcal{A}_\varphi^{\mathcal{G}})$ . Because nondeterminisation makes the size of the automaton gain one exponential for each nested quantifier on propositions, the procedure is nonelementary, and hardness is inherited from the model-checking problem for QCTL [14].

## V. MODEL-CHECKING HIERARCHICAL INSTANCES OF $\text{SL}_{ii}$

In this section we establish that the model-checking problem for  $\text{SL}_{ii}$  restricted to the class of hierarchical instances is decidable (Theorem 2).

We build upon the proof in [22] that establishes the decidability of the model-checking problem for  $\text{ATL}_{sc}^*$  by reduction to the model-checking problem for QCTL\*. The main difference is that we reduce to the model-checking problem for QCTL\* instead, using quantifiers on atomic propositions parameterised with observations that reflect the ones used in the  $\text{SL}_{ii}$  model-checking instance.

Let  $(\Phi, \mathcal{G})$  be a hierarchical instance of the  $\text{SL}_{ii}$  model-checking problem, where  $\mathcal{G} = (\text{Ac}, V, E, \ell, v_i, \mathcal{O})$ . We will first show how to define a CKS  $\mathcal{S}_\mathcal{G}$  and a bijection  $\rho \mapsto u_\rho$  between the set of finite plays  $\rho$  starting in a given position  $v$  and the set of nodes in  $t_{\mathcal{S}_\mathcal{G}}(s_v)$ .

Then, for every subformula  $\varphi$  of  $\Phi$  and partial function  $f : \text{Ag} \rightarrow \text{Var}$ , we will define a QCTL\* formula  $(\varphi)^f$  (that will also depend on  $\mathcal{G}$ ) such that the following holds:

**Proposition 5.** Suppose that  $\text{free}(\varphi) \cap \text{Ag} \subseteq \text{dom}(f)$ , and  $f(a) = x$  implies  $\chi(a) = \chi(x)$  for all  $a \in \text{dom}(f)$ . Then

$$\mathcal{G}, \chi, \rho \models \varphi \quad \text{if and only if} \quad t_{\mathcal{S}_\mathcal{G}}(s_\rho) \models (\varphi)^f.$$

Applying this to the sentence  $\Phi$ , any assignment  $\chi$ , and the empty function  $\emptyset$ , we get that

$$\mathcal{G}, \chi, v_i \models \Phi \quad \text{if and only if} \quad t_{\mathcal{S}_\mathcal{G}}(s_{v_i}) \models (\Phi)^\emptyset.$$

**Constructing the CKS  $\mathcal{S}_\mathcal{G}$ .** We will define  $\mathcal{S}_\mathcal{G}$  so that (indistinguishable) nodes in its tree-unfolding correspond to (indistinguishable) finite plays in  $\mathcal{G}$ . The CKS will make use of atomic propositions  $\text{AP}_v := \{p_v \mid v \in V\}$  (that we assume to be disjoint from AP). The idea is that  $p_v$  allows the QCTL\* formula  $(\Phi)^\emptyset$  to refer to the current position  $v$  in  $\mathcal{G}$ . Later we will see that  $(\Phi)^\emptyset$  will also make use of atomic propositions  $\text{AP}_c := \{p_c^x \mid c \in \text{Ac} \text{ and } x \in \text{Var}\}$  that we assume, again, are disjoint from  $\text{AP} \cup \text{AP}_v$ . This allows the formula to use  $p_c^x$  to refer to the actions  $c$  advised by strategies  $x$ .

Suppose  $\text{Obs} = \{o_1, \dots, o_n\}$ . For  $i \in [n]$ , define the local states  $L_i := \{[v]_{o_i} \mid v \in V\}$  where  $[v]_o$  is the equivalence class of  $v$  for relation  $\sim_o$ . Since we need to know the actual position of the CGS<sub>ii</sub> to define the dynamics, we also let  $L_{n+1} := V$ .

Define the CKS  $\mathcal{S}_\mathcal{G} := (S, R, s_\iota, \ell')$  where

- $S := \{s_v \mid v \in V\}$ ,
- $R := \{(s_v, s_{v'}) \mid \exists c \in \text{Ac}^{\text{Ag}} \text{ s.t. } E(v, c) = v'\} \subseteq S^2$ ,
- $s_\iota := s_{v_\iota}$ ,
- $\ell'(s_v) := \ell(v) \cup \{p_v\} \subseteq \text{AP} \cup \text{AP}_v$ ,

and  $s_v := ([v]_{o_1}, \dots, [v]_{o_n}, v) \in \prod_{i \in [n+1]} L_i$ .

We now show how to connect finite plays in  $\mathcal{G}$  with nodes in the tree unfolding of  $\mathcal{S}_\mathcal{G}$ . For every finite play  $\rho = v_0 \dots v_k$ , define the node  $u_\rho := s_{v_0} \dots s_{v_k}$  in  $t_{\mathcal{S}_\mathcal{G}}(s_{v_0})$  (which exists, by definition of  $\mathcal{S}_\mathcal{G}$  and of tree unfoldings). Note that the mapping  $\rho \mapsto u_\rho$  defines a bijection between the set of finite plays and the set of nodes in  $t_{\mathcal{S}_\mathcal{G}}(s_\iota)$ .

**Constructing the  $\text{QCTL}_{i,\subseteq}^*$  formulas  $(\varphi)^f$ .** We now describe how to transform an  $\text{SL}_{ii}$  formula  $\varphi$  and a partial function  $f : \text{Ag} \rightarrow \text{Var}$  into a  $\text{QCTL}_{ii}^*$  formula  $(\varphi)^f$  (that will also depend on  $\mathcal{G}$ ). Suppose that  $\text{Ac} = \{c_1, \dots, c_l\}$ , and define  $(\varphi)^f$  by induction on  $\varphi$ . We begin with the simple cases:  $(p)^f := p$ ;  $(\neg\varphi)^f := \neg(\varphi)^f$ ; and  $(\varphi_1 \vee \varphi_2)^f := (\varphi_1)^f \vee (\varphi_2)^f$ .

We continue with the second-order quantifier case:

$$(\langle\langle x\rangle\rangle^o \varphi)^f := \exists \tilde{o} p_{c_1}^x \dots \exists \tilde{o} p_{c_l}^x \cdot \varphi_{\text{str}}(x) \wedge (\varphi)^f$$

where  $\tilde{o}_i := \{j \mid \mathcal{O}(o_i) \subseteq \mathcal{O}(o_j)\}$ , and

$$\varphi_{\text{str}}(x) := \mathbf{AG} \bigvee_{c \in \text{Ac}} (p_c^x \wedge \bigwedge_{c' \neq c} \neg p_{c'}^x).$$

We describe this formula in words. For each possible action  $c \in \text{Ac}$ , an existential quantification on the atomic proposition  $p_c^x$  “chooses” for each finite play  $\rho = v_0 \dots v_k$  of  $\mathcal{G}$  (or, equivalently, for each node  $u_\rho$  of the tree  $t_{\mathcal{S}_\mathcal{G}}(s_{v_0})$ ) whether strategy  $x$  plays action  $c$  in  $\rho$  or not. Formula  $\varphi_{\text{str}}(x)$  ensures that in each finite play, exactly one action is chosen for strategy  $x$ , and thus that atomic propositions  $p_c^x$  indeed characterise a strategy, call it  $\sigma_x$ .<sup>4</sup>

Moreover, a quantifier with concrete observation  $\tilde{o}_i$  receives information corresponding to observation  $o_i$  (observe that for all  $i \in [n]$ ,  $i \in \tilde{o}_i$ ) as well as information corresponding to coarser observations. Note that including all coarser observations does not increase the information accessible to the quantifier: indeed, one can show that two nodes are  $\{i\}$ -indistinguishable if and only if they are  $\tilde{o}_i$ -indistinguishable. However, this definition of  $\tilde{o}_i$  allows us to obtain hierarchical formulas. Since quantification on propositions  $p_c^x$  is done uniformly with regards to concrete observation  $\tilde{o}_i$ , it follows that  $\sigma_x$  is an  $o_i$ -strategy.

Here are the remaining cases:

$$\begin{aligned} ((a, x)\varphi)^f &:= (\varphi)^{f[a \mapsto x]} \\ (\mathbf{X}\varphi_1)^f &:= \mathbf{A}(\psi_{\text{out}}(f) \rightarrow \mathbf{X}(\varphi_1)^f) \\ (\varphi_1 \mathbf{U} \varphi_2)^f &:= \mathbf{A}(\psi_{\text{out}}(f) \rightarrow (\varphi_1)^f \mathbf{U} (\varphi_2)^f) \end{aligned}$$

where

$$\psi_{\text{out}}(f) := \mathbf{G} \left( \bigwedge_{v \in V} \bigwedge_{c \in \text{Ac}^{\text{Ag}}} \left( p_v \wedge \bigwedge_{a \in \text{Ag}} p_{ca}^{f(a)} \rightarrow \mathbf{X} p_{E(v,c)} \right) \right).$$

The formula  $\psi_{\text{out}}(f)$  is used to select the unique path assuming that every player, say  $a$ , follows the strategy  $\sigma_{f(a)}$ .

This completes the justification of Proposition 5.

<sup>4</sup>More precisely, if  $\varphi_{\text{str}}(x)$  holds in node  $u_\rho$ , it ensures that propositions from  $\text{AP}_c$  define a partial strategy, defined on all nodes of the subtree rooted in  $u_\rho$ . This is enough because  $\text{SL}_{ii}$  can only talk about the future: when evaluating a formula in a finite play  $\rho$ , the definition of strategies on plays that do not start with  $\rho$  is irrelevant.

**Preserving hierarchy.** To complete the proof we show that  $(\Phi)^\emptyset$  is a hierarchical QCTL<sub>ii</sub><sup>\*</sup> formula. This simply follows from the fact that  $\Phi$  is hierarchical in  $\mathcal{G}$  and that for every two observations  $o_i$  and  $o_j$  in Obs such that  $\mathcal{O}(o_i) \subseteq \mathcal{O}(o_j)$ , by definition of  $\widetilde{o}_k$  we have that  $\widetilde{o}_i \subseteq \widetilde{o}_j$ .

This completes the proof of Theorem 2.

## VI. OUTLOOK

We introduced  $\text{SL}_{ii}$ , a logic for reasoning about strategic behaviour in multi-player games with imperfect information. The syntax mentions observations, and thus allows one to write formulas that talk about dynamic observations. We isolated the class of hierarchical formula/model pairs  $\Phi, \mathcal{G}$  and proved that one can decide if  $\mathcal{G} \models \Phi$ . The proof reduces (hierarchical) instances to (hierarchical) formulas of QCTL<sub>ii</sub><sup>\*</sup>, a low-level logic that we introduced, and that serves as a natural bridge between  $\text{SL}_{ii}$  (that talks about players and strategies) and automata constructions. We believe that QCTL<sub>ii</sub><sup>\*</sup> is of independent interest and deserves study in its own right.

Since one can alternate quantifiers in  $\text{SL}_{ii}$ , our decidability result goes beyond synthesis. As we showed, we can use it to decide if a game that yields hierarchical observation has a Nash equilibrium. A crude but easy analysis of our main decision procedure shows it is non-elementary.

This naturally leads to a number of avenues for future work: define and study the expressive power and computational complexity of fragments of  $\text{SL}_{ii}$  [36]; adapt the notion of hierarchical instances to allow for situations in which hierarchies can change infinitely often along a play [13]; and extend the logic to include epistemic operators for individual and common knowledge, as is done in [37], which are important for reasoning about distributed systems [21].

## REFERENCES

- [1] F. Mogavero, A. Murano, G. Perelli, and M. Y. Vardi, “Reasoning about strategies: On the model-checking problem,” *TOCL*, vol. 15, no. 4, pp. 34:1–34:47, 2014.
- [2] N. Bulling and W. Jamroga, “Comparing variants of strategic ability: how uncertainty and memory influence general properties of games,” *AAMAS’14*, vol. 28, no. 3, pp. 474–518, 2014.
- [3] F. Laroussinie, N. Markey, and A. Sangnier, “ATLsc with partial observation,” in *GandALF’15*, 2015, pp. 43–57.
- [4] C. Dima and F. L. Tiplea, “Model-checking ATL under imperfect information and perfect recall semantics is undecidable,” *CoRR*, vol. abs/1102.4225, 2011.
- [5] A. Pnueli and R. Rosner, “Distributed reactive systems are hard to synthesize,” in *FOCS’90*, 1990, pp. 746–757.
- [6] G. L. Peterson and J. H. Reif, “Multiple-person alternation,” in *SFCS’79*, 1979, pp. 348–363.
- [7] O. Kupfermann and M. Y. Vardi, “Synthesizing distributed systems,” in *LICS’01*, 2001, pp. 389–398.
- [8] G. Peterson, J. Reif, and S. Azhar, “Decision algorithms for multiplayer noncooperative games of incomplete information,” *CAMWA*, vol. 43, no. 1, pp. 179–206, 2002.
- [9] B. Finkbeiner and S. Schewe, “Uniform distributed synthesis,” in *LICS’05*, 2005, pp. 321–330.
- [10] S. Pinchinat and S. Riedweg, “A decidable class of problems for control under partial observation,” *IPL*, vol. 95, no. 4, pp. 454–460, 2005.
- [11] S. Schewe and B. Finkbeiner, “Distributed synthesis for alternating-time logics,” in *ATVA’07*, 2007, pp. 268–283.
- [12] K. Chatterjee and L. Doyen, “Games with a weak adversary,” in *ICALP’14*, 2014, pp. 110–121.
- [13] D. Berwanger, A. B. Mathew, and M. Van den Bogaard, “Hierarchical information patterns and distributed strategy synthesis,” in *ATVA’15*, 2015, pp. 378–393.
- [14] F. Laroussinie and N. Markey, “Quantified CTL: expressiveness and complexity,” *LMCS*, vol. 10, no. 4, 2014.
- [15] G. Peterson, J. Reif, and S. Azhar, “Lower bounds for multiplayer noncooperative games of incomplete information,” *CAMWA*, vol. 41, no. 7, pp. 957–992, 2001.
- [16] J. Y. Halpern and M. Y. Vardi, “The complexity of reasoning about knowledge and time. i. lower bounds,” *JCSS*, vol. 38, no. 1, pp. 195–237, 1989.
- [17] H. Läuchli and C. Savioz, “Monadic second order definable relations on the binary tree,” *JSL*, vol. 52, no. 01, pp. 219–226, 1987.
- [18] P. Gastin, N. Sznajder, and M. Zeitoun, “Distributed synthesis for well-connected architectures,” *FMSD*, vol. 34, no. 3, pp. 215–237, 2009.
- [19] B. Finkbeiner and S. Schewe, “Coordination logic,” in *CSL’10*, 2010, pp. 305–319.
- [20] R. Alur, T. A. Henzinger, and O. Kupferman, “Alternating-time temporal logic,” *JACM*, vol. 49, no. 5, pp. 672–713, 2002.
- [21] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi, *Reasoning about knowledge*. MIT press Cambridge, 1995, vol. 4.
- [22] F. Laroussinie and N. Markey, “Augmenting ATL with strategy contexts,” *IC*, vol. 245, pp. 98–123, 2015.
- [23] A. Sistla, “Theoretical Issues in the Design and Cerification of Distributed Systems.” Ph.D. dissertation, Harvard University, Cambridge, MA, USA, 1983.
- [24] E. A. Emerson and A. P. Sistla, “Deciding branching time logic,” in *STOC’84*, 1984, pp. 14–24.
- [25] O. Kupferman, “Augmenting branching temporal logics with existential quantification over atomic propositions,” in *CAV’95*, 1995, pp. 325–338.
- [26] O. Kupferman, P. Madhusudan, P. S. Thiagarajan, and M. Y. Vardi, “Open systems in reactive environments: Control and synthesis,” in *CONCUR’00*, 2000, pp. 92–107.
- [27] T. French, “Decidability of quantified propositional branching time logics,” in *AJCAI’01*, 2001, pp. 165–176.
- [28] C. C. Elgot and M. O. Rabin, “Decidability and undecidability of extensions of second (first) order theory of (generalized) successor,” *JSL*, vol. 31, no. 2, pp. 169–181, 1966.
- [29] W. Thomas, “Infinite trees and automaton-definable relations over omega-words,” *TCS*, vol. 103, no. 1, pp. 143–159, 1992.
- [30] O. Kupferman and M. Y. Vardi, “Church’s problem revisited,” *BSL*, pp. 245–263, 1999.
- [31] D. E. Muller and P. E. Schupp, “Alternating automata on infinite trees,” *TCS*, vol. 54, pp. 267–276, 1987.
- [32] M. O. Rabin, “Decidability of second-order theories and automata on infinite trees,” *TAMS*, vol. 141, pp. 1–35, 1969.
- [33] D. E. Muller and P. E. Schupp, “Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra,” *TCS*, vol. 141, no. 1&2, pp. 69–107, 1995.
- [34] O. Kupferman, M. Y. Vardi, and P. Wolper, “An automata-theoretic approach to branching-time model checking,” *JACM*, vol. 47, no. 2, pp. 312–360, 2000.
- [35] M. Y. Vardi and P. Wolper, “Reasoning about infinite computations,” *IC*, vol. 115, no. 1, pp. 1–37, 1994.
- [36] F. Mogavero, A. Murano, G. Perelli, and M. Y. Vardi, “What makes ATL\* decidable? a decidable fragment of Strategy Logic,” in *CONCUR’12*, 2012, pp. 193–208.
- [37] P. Cermák, A. Lomuscio, F. Mogavero, and A. Murano, “MCMAS-SLK: A model checker for the verification of strategy logic specifications,” in *CAV’14*, 2014, pp. 525–532.
- [38] W. Zielonka, “Infinite games on finitely coloured graphs with applications to automata on infinite trees,” *TCS*, vol. 200, no. 1-2, pp. 135–183, 1998.

## APPENDIX

### A. Strategy logic under imperfect information

We establish two facts about  $\text{SL}_{\text{ii}}$  mentioned in the body: 1) the statement  $\mathcal{G}, \chi, \rho \models \varphi$  is independent of  $\chi$  if  $\varphi$  is a sentence; 2) Proposition 1 which says that the perfect-information fragment of  $\text{SL}_{\text{ii}}$  is a notational variant of  $\text{SL}$  given in [1].

#### Truth of sentences are independent of the assignment.

We begin with a formal definition of free players and variables in  $\text{SL}_{\text{ii}}$ .

**Definition 9** (Free players and variables). *The set  $\text{free}(\varphi)$  of free players and free variables of an  $\text{SL}_{\text{ii}}$  formula  $\varphi$  is defined as follows:*

- $\text{free}(p) := \emptyset$ , where  $p \in \text{AP}$ .
- $\text{free}(\neg\varphi) := \text{free}(\varphi)$ .
- $\text{free}(\varphi_1 \vee \varphi_2) := \text{free}(\varphi_1) \cup \text{free}(\varphi_2)$ .
- $\text{free}(\mathbf{X}\varphi) := \text{Ag} \cup \text{free}(\varphi)$ .
- $\text{free}(\varphi_1 \mathbf{U} \varphi_2) := \text{Ag} \cup \text{free}(\varphi_1) \cup \text{free}(\varphi_2)$ .
- $\text{free}(\langle x \rangle^o \varphi) := \text{free}(\varphi) \setminus \{x\}$ .
- $\text{free}((a, x)\varphi) := \text{free}(\varphi)$ , if  $a \notin \text{free}(\varphi)$ .
- $\text{free}((a, x)\varphi) := (\text{free}(\varphi) \setminus \{a\}) \cup \{x\}$ , if  $a \in \text{free}(\varphi)$ .

**Lemma 2.** *For all CGS<sub>ii</sub>  $\mathcal{G}$ , finite plays  $\rho$ , assignments  $\chi, \chi'$ , and  $\text{SL}_{\text{ii}}$  sentences  $\Phi$ ,*

$$\mathcal{G}, \chi, \rho \models \Phi \quad \text{iff} \quad \mathcal{G}, \chi', \rho \models \Phi.$$

*Proof.* Since a sentence has no free variables or players, it is enough to prove the following for formulas  $\varphi$ : if  $\chi$  and  $\chi'$  agree on the free variables and players of  $\varphi$ , then  $\mathcal{G}, \chi, \rho \models \varphi$  iff  $\mathcal{G}, \chi', \rho \models \varphi$ . The proof is by induction on  $\varphi$ .

The case of an atom is immediate since the semantics do not depend on the assignment.

The case of negation follows immediately from the inductive hypothesis using the fact that  $\text{free}(\neg\varphi) = \text{free}(\varphi)$ . The case of disjunction is similar.

For the quantifier case  $\langle x \rangle^o \varphi$  use the fact that  $\chi[x \mapsto \sigma]$  and  $\chi'[x \mapsto \sigma]$  agree on the free placeholders of  $\varphi$  (since we assumed that  $\chi, \chi'$  agree on the free placeholders of  $\langle x \rangle^o \varphi$ ).

For the binding case  $(a, x)\varphi$  use the fact that  $\chi[a \mapsto \chi(x)]$  and  $\chi'[a \mapsto \chi'(x)]$  agree on the free placeholders of  $\varphi$ .

For the next-operator case  $\mathbf{X}\varphi$  use the fact that  $\text{out}(\chi, \rho) = \text{out}(\chi', \rho)$  (since  $\text{out}$  only depends on the strategies assigned to players), and  $\chi, \chi'$  agree on  $\text{free}(\varphi)$  since  $\text{free}(\varphi) \subseteq \text{free}(\mathbf{X}\varphi)$ . The case of  $\mathbf{U}$  is similar.  $\square$

1) *Comparison of the perfect-information fragment of  $\text{SL}_{\text{ii}}$  with the definition of  $\text{SL}$  from [1]:* We recall the definitions of the syntax and semantics of  $\text{SL}$ .

**Definition 10** ( $\text{SL}$  syntax [1]). *The syntax of  $\text{SL}$  is defined by the following grammar:*

$$\varphi := p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \varphi \mid \langle x \rangle \varphi \mid (a, x)\varphi$$

where  $p \in \text{AP}$ ,  $x \in \text{Var}$ , and  $a \in \text{Ag}$ .

An *assignment* is a partial function  $\chi : \text{Ag} \cup \text{Var} \rightarrow \text{Str}$ , assigning to each player and variable in its domain a strategy.

For an assignment  $\chi$ , a player  $a$  and a strategy  $\sigma$ ,  $\chi[a \mapsto \sigma]$  is the assignment of domain  $\text{dom}(\chi) \cup \{a\}$  that maps  $a$  to  $\sigma$  and is equal to  $\chi$  on the rest of its domain, and  $\chi[x \mapsto \sigma]$  is defined similarly, where  $x$  is a variable. An assignment  $\chi$  is *player-complete* for  $\mathcal{G}$ , or simply *player-complete* when  $\mathcal{G}$  is clear from the context, if it assigns a strategy to each player in  $\mathcal{G}$ , i.e.,  $\text{Ag} \subseteq \text{dom}(\chi)$ . For a player-complete assignment  $\chi$  and a position  $v$ , let  $\text{out}(\chi, v)$  be the infinite play obtained when the game starts in  $v$  and all players follow the strategies assigned by  $\chi$ :  $\text{out}(\chi, v) := v_0 v_1 \dots$  with  $v_0 = v$  and for each  $i \geq 0$ ,  $v_{i+1} = E(v_i, c)$ , where  $c = (\chi(a)(v_0 \dots v_i))_{a \in \text{Ag}}$ .

In addition, given a formula  $\varphi \in \text{SL}$ , an assignment is *variable-complete* for  $\varphi$ , or simply *variable-complete* when  $\varphi$  is clear from the context, if  $\text{free}(\varphi) \cap \text{Var} \subseteq \text{dom}(\chi)$ .

Given an assignment  $\chi$  and an initial play  $\rho$ , define the  $\rho$ -translation of  $\chi$  as the assignment  $\chi^\rho$  such that  $\text{dom}(\chi^\rho) = \text{dom}(\chi)$ , and for every  $l \in \text{dom}(\chi^\rho)$ ,  $\chi^\rho(l) = \chi(l)^\rho$ . We also define the *global translation* of a complete assignment  $\chi$  together with a position  $v$  as follows: for every  $i \in \mathbb{N}$ , the  $i$ -*global translation* of  $(\chi, v)$  is defined as  $(\chi, v)^i := (\chi^{\pi \leq i}, \pi_i)$ , where  $\pi = \text{out}(\chi, v)$ .

Models of  $\text{SL}$  formulas are concurrent games structures (CGS), i.e., tuples  $\mathcal{G} = (\text{Ac}, V, E, \ell, v_\ell)$  where the entries are as for CGS<sub>ii</sub> (but without  $\text{Obs}$  and  $\mathcal{O}$ ).

**Definition 11** ( $\text{SL}$  Semantics [1]). *Let  $\varphi$  be an  $\text{SL}$ -formula. The semantics of  $\varphi$  in a CGS  $\mathcal{G}$  at position  $v \in \mathcal{G}$  with a variable-complete assignment  $\chi$  is defined inductively as follows:*

$$\begin{aligned} \mathcal{G}, \chi, v \models p &\quad \text{if } p \in \ell(v) \\ \mathcal{G}, \chi, v \models \neg\varphi &\quad \text{if } \mathcal{G}, \chi, v \not\models \varphi \\ \mathcal{G}, \chi, v \models \varphi \vee \varphi' &\quad \text{if } \mathcal{G}, \chi, v \models \varphi \text{ or } \mathcal{G}, \chi, v \models \varphi' \\ \mathcal{G}, \chi, v \models \langle x \rangle \varphi &\quad \text{if } \exists \sigma \in \text{Str} \text{ s.t. } \mathcal{G}, \chi[x \mapsto \sigma], v \models \varphi \\ \mathcal{G}, \chi, v \models (a, x)\varphi &\quad \text{if } \mathcal{G}, \chi[a \mapsto \chi(x)], v \models \varphi \end{aligned}$$

If, in addition,  $\chi$  is player-complete, then

$$\begin{aligned} \mathcal{G}, \chi, v \models \mathbf{X}\varphi &\quad \text{if } \mathcal{G}, (\chi, v)^1 \models \varphi \\ \mathcal{G}, \chi, v \models \varphi \mathbf{U} \varphi' &\quad \text{if } \exists i \geq 0 \text{ s.t. } \mathcal{G}, (\chi, v)^i \models \varphi' \text{ and} \\ &\quad \forall j \text{ s.t. } 0 \leq j < i, \mathcal{G}, (\chi, v)^j \models \varphi \end{aligned}$$

For a sentence  $\varphi \in \text{SL}$  define  $\mathcal{G}, v \models \varphi$  if  $\mathcal{G}, \emptyset, v \models \varphi$  (where  $\emptyset$  is the empty assignment), and write  $\mathcal{G} \models \varphi$  if  $\mathcal{G}, v_\ell \models \varphi$ . This completes the recap of the syntax and semantics of  $\text{SL}$ .

**Proof of proposition 1.** We prove this in two steps. First, note that in the definition of  $\text{SL}$ , one can restrict to complete assignments when defining  $\models$ . Indeed, an easy induction shows that for all partial assignments  $\chi, \chi'$  such that  $\chi'$  extends  $\chi$  (i.e.,  $\text{dom}(\chi) \subseteq \text{dom}(\chi')$  and  $\chi(l) = \chi'(l)$  for all  $l \in \text{dom}(\chi)$ ), we have that  $\mathcal{G}, \chi, v \models \varphi$  iff  $\mathcal{G}, \chi', v \models \varphi$ . Thus, from now on we assume that all assignments of  $\text{SL}$  are complete. In addition, in [1] the authors define  $\mathcal{G}, v \models \varphi$ , for  $\varphi$  a sentence, as  $\mathcal{G}, \emptyset, v \models \varphi$  where  $\emptyset$  is the empty assignment. This is equivalent to stating that  $\mathcal{G}, \chi, v \models \varphi$  for all complete assignments  $\chi$ .

To prove proposition 1, let  $\varphi$  be an  $\text{SL}$  formula. Fix an observation symbol  $o$  and let  $\text{Obs} := \{o\}$ . Define the  $\text{SL}_{\text{ii}}$

formula  $\varphi'$  by annotating every strategy quantifier in  $\varphi$  by  $o$ , i.e., by replacing  $\langle\langle x\rangle\rangle$  by  $\langle\langle x\rangle\rangle^o$ . Let  $\mathcal{G}$  be a CGS. Define the CGS<sub>ii</sub>  $\mathcal{G}'$  to be  $(\mathcal{G}, \text{Obs}, \mathcal{O})$  where  $\mathcal{O}(o) = \{(v, v) : v \in V\}$  is the identity observation. We now prove that for every complete assignment  $\chi$ , and  $v \in \mathcal{G}$ ,  $\mathcal{G}, \chi, v \models \varphi$  iff  $\mathcal{G}', \chi, v \models \varphi'$ , which establishes the proposition when  $\varphi$  is a sentence. This follows by induction on  $\varphi$  using the following inductive hypothesis: For all  $n \in \mathbb{N}$ ,

$$\mathcal{G}, (\chi, v)^n \models \varphi \text{ iff } \mathcal{G}', \chi, \pi^{\leq n} \models \varphi,$$

where  $\pi = \text{out}(\chi, v)$ . The inductive step is immediate from the definitions of the semantics of  $\text{SL}$  and  $\text{SL}_{\text{ii}}$ .

### B. Translation of CL into SL<sub>ii</sub>

In this section we prove Proposition 4 by giving two canonical translations: first from CL-instances into QCTL<sub>i,≤</sub><sup>\*</sup>-instances, and then translating QCTL<sub>ii</sub><sup>\*</sup>-instances into hierarchical SL<sub>ii</sub>-instances.

We briefly recall the syntax and semantics of CL, and refer to [19] for further details. For definitions about trees, see Section III-B.

**Notation for trees.** Note that our definition for trees slightly differs from the one in [19], where the root is the empty word. Here we adopt this convention to keep closer to notations in [19]. Thus,  $(Y, X)$ -trees in CL are of the form  $(\tau, l)$  where  $\tau \subseteq X^*$  and  $l : \tau \rightarrow 2^Y$ .

For two disjoint sets  $X$  and  $Y$ , we identify  $2^X \times 2^Y$  with  $2^{X \cup Y}$ . Let  $X$  and  $Y$  be two sets with  $Z = X \cup Y$ , and let  $M$  and  $N$  be two disjoint sets. Given an  $M$ -labelled  $2^Z$ -tree  $t = (\tau, \ell_M)$  and an  $N$ -labelled  $2^Z$ -tree  $t' = (\tau', \ell_N)$  with same domain  $\tau = \tau'$ , we define  $t \uplus t' := (\tau, \ell')$ , where for every  $u \in \tau$ ,  $\ell'(u) = \ell_M(u) \cup \ell_N(u)$ . Now, given a complete  $M$ -labelled  $2^X$ -tree  $t = ((2^X)^*, \ell_M)$  and a complete  $N$ -labelled  $2^Y$ -tree  $t' = ((2^Y)^*, \ell_N)$ , we define  $t \oplus t' := t \upharpoonright^{2^Z \setminus X} \uplus t' \upharpoonright^{2^Z \setminus Y}$ .

**CL Syntax.** Let  $\mathcal{C}$  be a set of *coordination variables*, and let  $\mathcal{S}$  be a set of *strategy variables* disjoint from  $\mathcal{C}$ . The syntax of CL is given by the following grammar:

$$\varphi ::= x \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U}\varphi \mid \exists C \exists s. \varphi$$

where  $x \in \mathcal{C} \cup \mathcal{S}$ ,  $C \subseteq \mathcal{C}$  and  $s \in \mathcal{S}$ , and with the restriction that each coordination variable appears in at most once *subtree quantifier*  $\exists C \exists s.$ , and similarly for strategy variables.

The notion of free and bound (coordination or strategy) variables is as usual. The set of free coordination variables in  $\varphi$  is noted  $\mathcal{F}_\varphi$ . A bound coordination variable  $c$  is *visible* to a strategy variable  $s$  if  $s$  is in the scope of the quantifier that introduces  $c$ , and  $\text{Scope}_\varphi(s)$  is the union of the set of bound coordination variables visible to  $s$  and the set of free coordination variables (note that this union is disjoint). We will see, in the semantics, that the meaning of a bound strategy variable  $s$  is a strategy  $f_s : (2^{\text{Scope}_\varphi(s)})^* \rightarrow 2^{\{s\}}$ . Free strategy variables are called *atomic propositions*, and we denote the set of atomic propositions in  $\varphi$  by  $\text{AP}_\varphi$ .

**CL Semantics.** A CL formula  $\varphi$  is evaluated on a complete  $\text{AP}_\varphi$ -labelled  $2^{\mathcal{F}_\varphi}$ -tree  $t$ . An  $(\text{AP}_\varphi, 2^{\mathcal{F}_\varphi})$ -tree  $t = (\tau, \ell)$  satisfies a CL formula  $\varphi$  if for every path  $\lambda$  that starts in the

root we have  $t, \lambda, 0 \models \varphi$ , where the satisfaction of a formula at position  $i \geq 0$  of a path  $\lambda$  is defined inductively as follows:

$$\begin{aligned} t, \lambda, i \models p & \quad \text{if } p \in \ell(\lambda_i) \\ t, \lambda, i \models \neg\varphi' & \quad \text{if } t, \lambda, i \not\models \varphi' \\ t, \lambda, i \models \varphi_1 \vee \varphi_2 & \quad \text{if } t, \lambda, i \models \varphi_1 \text{ or } t, \lambda, i \models \varphi_2 \\ t, \lambda, i \models \mathbf{X}\varphi' & \quad \text{if } t, \lambda, i + 1 \models \varphi' \\ t, \lambda, i \models \varphi_1 \mathbf{U}\varphi_2 & \quad \text{if } \exists j \geq i \text{ s.t. } t, \lambda, j \models \varphi_2 \text{ and} \\ & \quad \forall k \text{ s.t. } i \leq k < j, t, \lambda, k \models \varphi_1 \\ t, \lambda, i \models \exists C \exists s. \varphi' & \quad \text{if } \exists f : (2^{\text{Scope}_\varphi(s)})^* \rightarrow 2^{\{s\}} \text{ s.t.} \\ & \quad t_{\lambda_i} \oplus ((2^{\text{Scope}_\varphi(s)})^*, f) \models \varphi', \end{aligned}$$

where  $t_{\lambda_i}$  is the subtree of  $t$  rooted in  $\lambda_i$ .

First, observe that in the last inductive case,  $t_{\lambda_i}$  being a  $2^{\mathcal{F}_\varphi}$ -tree,  $t_{\lambda_i} \oplus ((2^{\text{Scope}_\varphi(s)})^*, f)$  is a  $2^{\mathcal{F}_\varphi \cup \text{Scope}_\varphi(s)}$ -tree. By definition,  $\text{Scope}_\varphi(s) = \mathcal{F}_\varphi \cup C = \mathcal{F}_{\varphi'}$ . It follows that  $\mathcal{F}_\varphi \cup \text{Scope}_\varphi(s) = \text{Scope}_\varphi(s) = \mathcal{F}_{\varphi'}$ , hence  $\varphi'$  is indeed evaluated on a  $\mathcal{F}_{\varphi'}$ -tree.

**Remark 1.** Note that all strategies observe the value of all atomic propositions. Formally, for every CL-formula  $\varphi$  of the form  $\varphi = \exists C_1 \exists s_1. \dots, \exists C_i \exists s_i. \varphi'$  evaluated on a  $2^{\mathcal{F}_\varphi}$ -tree  $t = (\tau, \ell)$ ,  $\varphi'$  is evaluated on a  $2^{\mathcal{F}_\varphi \cup C_1 \cup \dots \cup C_i}$ -tree  $t' = (\tau', \ell')$  such that for every  $p \in \text{AP}_\varphi$ , for every pair of nodes  $u, u' \in t'$  such that  $u \downarrow_{2^{\mathcal{F}_\varphi}} = u' \downarrow_{2^{\mathcal{F}_\varphi}}$ , it holds that  $p \in \ell'(u)$  iff  $p \in \ell'(u')$ .

Thus, in CL one cannot directly capture strategic problems on concurrent game structures where atomic propositions are not observable to all players. This contrasts with the definition of ATL with imperfect information in [20, Section 7.1] where imperfect information of agents is modelled by defining which atomic propositions they can observe.

We now describe a natural translation of CL-instances to SL<sub>ii</sub>-instances. This translation has two consequences:

- 1) It reduces the model-checking problem of CL to that of the hierarchical fragment of SL<sub>ii</sub>.
- 2) It shows that CL only produces instances in which all atoms are uniform with regard to all observations, i.e., instances  $(\Phi, \mathcal{G})$  such that for every  $p \in \text{AP}$  and  $o \in \text{Obs}$ ,  $v \sim_o v'$  implies  $p \in \ell(v) \leftrightarrow p \in \ell(v')$ .

The input to the model-checking problem for CL consists of a CL formula  $\varphi$  and a finite representation of a  $(\text{AP}_\varphi, 2^{\mathcal{F}_\varphi})$ -tree  $t$ . The standard assumption is to assume  $t$  is a regular tree, i.e., is the unfolding of a finite structure. Precisely, a *finite representation* of a  $(\text{AP}_\varphi, 2^{\mathcal{F}_\varphi})$ -tree  $t = (\tau, \ell)$  is a structure  $\mathcal{S} = (S, R, s_\ell, \ell)$  such that

- $S = 2^{\mathcal{F}_\varphi}$ ,
- $s_\ell \in S$ ,
- $R = S \times S$ ,
- $\ell : S \rightarrow 2^{\text{AP}_\varphi}$ ,

and  $t = t_{\mathcal{S}}(s_\ell)$  is the unfolding of  $\mathcal{S}$ .

Thus, an *instance* of the model-checking problem for CL is a pair  $(\varphi, \mathcal{S})$  where  $\varphi$  is a CL formula (over variables

$\mathcal{S} \cup \mathcal{C}$ ), and  $\mathcal{S} = (S, R, s_\ell, \ell)$  is a finite representation of a  $(\text{AP}_\varphi, 2^{\mathcal{F}_\varphi})$ -tree. The *model-checking problem for CL* is the following decision problem: given an instance  $(\varphi, \mathcal{S})$ , return ‘Yes’ if  $t_{\mathcal{S}}(s_\ell) \models \varphi$  and ‘No’ otherwise.

We will present the translation in two steps: first from CL-instances into  $\text{QCTL}_{i,\subseteq}^*$ -instances, and then from  $\text{QCTL}_{ii}^*$ -instances to  $\text{SL}_{ii}$ -instances such that  $\text{QCTL}_{i,\subseteq}^*$ -instances translate to hierarchical  $\text{SL}_{ii}$ -instances. We remind the reader that we use tree-semantics for  $\text{QCTL}_{i,\subseteq}^*$  (indeed, we only defined tree-semantics, not structure-semantics, for  $\text{QCTL}_{i,\subseteq}^*$ ).

1) *Translating CL to  $\text{QCTL}_{i,\subseteq}^*$ :* Let  $(\varphi, \mathcal{S})$  be an instance of the model-checking problem for CL, where  $\mathcal{S} = (S, R, s_\ell, \ell)$ . We will construct a  $\text{QCTL}_{i,\subseteq}^*$ -instance  $(\bar{\varphi}, \bar{\mathcal{S}})$  such that  $\mathcal{S} \models \varphi$  iff  $\bar{\mathcal{S}} \models \bar{\varphi}$ . Let  $\overline{\text{AP}}$  be the set of all strategy variables occurring in  $\varphi$ , let  $\mathcal{C}(\varphi)$  be the set of coordination variables that appear in  $\varphi$ , and assume, w.l.o.g., that  $\mathcal{C}(\varphi) = [n]$  for some  $n \in \mathbb{N}$ . Let  $\text{hidden}(\varphi) := \mathcal{C}(\varphi) \setminus \mathcal{F}_\varphi$ .

First, we define the CKS  $\bar{\mathcal{S}}$  over  $\overline{\text{AP}}$ : the idea is to add in the structure  $\mathcal{S}$  the local states corresponding to coordination variables that are not seen by all the strategies.

Formally,  $\bar{\mathcal{S}} := (\bar{S}, \bar{R}, \bar{s}_\ell, \bar{\ell})$  where

- $\bar{S} = \prod_{c \in \mathcal{C}(\varphi)} L_c$  where  $L_c = \{c_0, c_1\}$  (for all  $c \in \mathcal{F}_\varphi$ ),
- $\bar{R} = \bar{S} \times \bar{S}$ ,
- $\bar{s}_\ell \in \bar{S}$  is any state  $s$  such that  $s \downarrow_{\mathcal{F}_\varphi} = s_\ell$ , and
- for every  $s \in \bar{S}$ ,  $\bar{\ell}(s) = \ell(s \downarrow_{\mathcal{F}_\varphi})$ .

Second, we define concrete observations corresponding to strategy variables in  $\varphi$ . As explained in [19], and as reflected in the semantics of CL, the intuition is that a strategy variable  $s$  in formula  $\varphi$  observes coordination variables  $\text{Scope}_\varphi(s)$ . Therefore, we simply define, for each strategy variable  $s$  in  $\varphi$ , the concrete observation  $\mathbf{o}_s := \text{Scope}_\varphi(s)$ .

Finally, we define the  $\text{QCTL}_{ii}^*$  formula  $\bar{\varphi}$ . This is done by induction on  $\varphi$  as follows (recall that we take for atomic propositions in  $\text{QCTL}_{ii}^*$  the set of all strategy variables in  $\varphi$ ):

$$\begin{aligned} \bar{x} &:= x \\ \bar{\neg\varphi} &:= \neg\bar{\varphi} \\ \bar{\varphi_1 \vee \varphi_2} &:= \bar{\varphi_1} \vee \bar{\varphi_2} \\ \bar{\mathbf{X}\varphi} &:= \mathbf{X}\bar{\varphi} \\ \bar{\varphi_1 \mathbf{U} \varphi_2} &:= \bar{\varphi_1} \mathbf{U} \bar{\varphi_2} \\ \bar{\exists C \exists s. \varphi} &:= \exists^{\mathbf{o}_s} s. \mathbf{A}\bar{\varphi} \end{aligned}$$

In the last case, note that  $C \subseteq \mathbf{o}_s = \text{Scope}_\varphi(s)$ .

Note that  $\bar{\varphi}$  is a hierarchical  $\text{QCTL}_{ii}^*$ -formula. Also, one can easily check that the following holds:

**Lemma 3.**  $t_{\mathcal{S}}(s_\ell) \models \varphi$  iff  $t_{\bar{\mathcal{S}}}(\bar{s}_\ell) \models \mathbf{A}\bar{\varphi}$ .

Importantly, we notice that  $\mathbf{A}\bar{\varphi} \in \text{QCTL}_{i,\subseteq}^*$ , and that:

**Lemma 4.** For every  $x \in \text{AP}_\varphi$  and every  $s$  quantified in  $\varphi$ ,  $t_{\bar{\mathcal{S}}}(\bar{s}_\ell)$  is  $\mathbf{o}_s$ -uniform in  $x$ .

2) *Translation from  $\text{QCTL}_{ii}^*$  to  $\text{SL}_{ii}$ :* We now present a translation of  $\text{QCTL}_{ii}^*$ -instances to  $\text{SL}_{ii}$ -instances. It is a simple adaption of the the reduction from the model-checking

problem for  $\text{QCTL}^*$  to the model-checking problem for  $\text{ATL}_{\text{sc}}^*$  presented in [22].

Let  $(\mathcal{S} = (S, R, s_\ell, \ell), \varphi)$  be an instance of the model-checking problem for  $\text{QCTL}_{ii}^*$ , where the set of states is  $S \subseteq \prod_{i \in [n]} L_i$ . We assume, without loss of generality, that every atomic proposition is quantified at most once, and that if it appears quantified it does not appear free. Also, let  $\text{AP}_\exists(\varphi) = \{p_1, \dots, p_k\}$  be the set of atomic propositions quantified in  $\varphi$ , and for  $i \in [k]$ , let  $\mathbf{o}_i$  be the concrete observation associated to the quantifier on  $p_i$ .

We build the CGS<sub>ii</sub>  $\mathcal{G}_\mathcal{S} := (\text{Ac}, V, E, \ell', v_\ell, \mathcal{O})$  over agents  $\text{Ag} := \{a_0, a_1, \dots, a_k\}$ , observations  $\text{Obs} := \{o_0, o_1, \dots, o_k\}$  and atomic propositions  $\text{AP} := \text{AP}(\varphi) \cup \{p_S\}$ , where  $p_S$  is a fresh atomic proposition. Intuitively, agent  $a_0$  is in charge of choosing transitions in  $\mathcal{S}$ , while agent  $a_i$  for  $i \geq 1$  is in charge of choosing the valuation for  $p_i \in \text{AP}_\exists(\varphi)$ .

To this aim, we let

$$V := \begin{cases} \{v_s \mid s \in S\} \cup \\ \{v_{s,i} \mid s \in S \text{ and } i \in [k]\} \cup \\ \{v_{p_i} \mid 0 \leq i \leq k\} \cup \\ \{v_\perp\} \end{cases}$$

and

$$\text{Ac} := \{c^s \mid s \in S\} \cup \{c^i \mid 0 \leq i \leq k\}.$$

In positions of the form  $v_s$  with  $s \in S$ , transitions are determined by the action chosen by agent  $a_0$ . First, she can choose to simulate a transition in  $\mathcal{S}$ : for every joint action  $\mathbf{c} \in \text{Ac}^{\text{Ag}}$  such that  $\mathbf{c}_0 = c^{s'}$ ,

$$E(v_s, \mathbf{c}) := \begin{cases} v_{s'} & \text{if } R(s, s') \\ v_\perp & \text{otherwise.} \end{cases}$$

She can also choose to move to a position in which agent  $a_i$  will choose the valuation for  $p_i$  in the current node: for every joint action  $\mathbf{c} \in \text{Ac}^{\text{Ag}}$  such that  $\mathbf{c}_0 = c^i$ ,

$$E(v_s, \mathbf{c}) := \begin{cases} v_{s,i} & \text{if } i \neq 0 \\ v_\perp & \text{otherwise.} \end{cases}$$

Next, in a position of the form  $v_{s,i}$ , agent  $a_i$  determines the transition, which codes the labelling of  $p_i$  in the current node: choosing  $c^i$  means that  $p_i$  holds in the current node, choosing any other action codes that  $p_i$  does not hold. Formally, for every joint action  $\mathbf{c} \in \text{Ac}^{\text{Ag}}$ ,

$$E(v_{s,i}, \mathbf{c}) := \begin{cases} v_{p_i} & \text{if } \mathbf{c}_i = c^i \\ v_\perp & \text{otherwise.} \end{cases}$$

Positions of the form  $v_{s,i}$  and  $v_\perp$  are sink positions.

The labelling function  $\ell'$  is defined as follows:

$$\ell'(v) := \begin{cases} \ell(s) \cup \{p_S\} & \text{if } v \in \{v_s \mid s \in S\} \\ \emptyset & \text{if } v \in \{v_{s,i} \mid s \in S, i \in [k]\} \\ \{p_i\} & \text{if } v = v_{p_i} \text{ with } i \in [k] \end{cases}$$

Finally we let  $v_\iota := v_{s_\iota}$  and we define the observation interpretation as follows:

$$\mathcal{O}(o_0) := \{(v, v) \mid v \in V\},$$

meaning that agent  $a_0$  has perfect information, and for  $i \in [k]$ ,  $\mathcal{O}(o_i)$  is the smallest reflexive relation such that

$$\mathcal{O}(o_i) \supseteq \bigcup_{s, s' \in S} \{(v_s, v_{s'}), (v_{s,i}, v_{s',i}) \mid s \approx_{\mathbf{o}_i} s'\}.$$

We explain the latter definition. First, observe that for every finite play  $\rho$  in  $\mathcal{G}_S$  that stays in  $V_S = \{v_s \mid s \in S\}$ , writing  $\rho = v_{s_0} \dots v_{s_n}$ , one can associate a finite path  $\lambda_\rho = s_0 \dots s_n$  in  $S$ . This mapping actually defines a bijection between the set of finite paths in  $S$  that start in  $s_\iota$  and the set of finite plays in  $\mathcal{G}_S$  that remain in  $V_S$ .

Now, according to the definition of the transition function, a strategy  $\sigma_i$  for agent  $i$  with  $i \in [k]$  is only relevant on finite plays of the form  $\rho = \rho' \cdot v_{s,i}$ , where  $\rho' \in V_S^*$ , and  $\sigma_i(\rho)$  is meant to determine whether  $p_i$  holds in  $\lambda_{\rho'}$ . If  $\sigma_i$  is  $\mathbf{o}_i$ -uniform, by definition of  $\mathcal{O}(o_i)$ , it determines an  $\mathbf{o}_i$ -uniform labelling for  $p$  in  $t_S(s_\iota)$ . Reciprocally, an  $\mathbf{o}_i$ -uniform labelling for  $p$  in  $t_S(s_\iota)$  induces an  $\mathcal{O}(o_i)$ -strategy for agent  $a_i$ .

It remains to transform  $\varphi$  into an  $\text{SL}_{ii}$ -formula. To simulate the path quantifier of  $\text{QCTL}_{ii}^*$ , a strategy for agent  $a_0$  is enough as she alone chooses the path in  $S$ . However the semantics of  $\text{SL}_{ii}$  is only defined on complete assignments, and we therefore need to make sure that all agents are bound to a strategy before using a temporal operator, and we must do so without breaking the hierarchy. For this we record in the translation what was the observation of the last propositional quantifier translated.

We define the  $\text{SL}_{ii}$  formula  $\overline{\varphi}^{\mathbf{o}_i}$  by induction on  $\varphi$  as follows:

$$\begin{aligned} \overline{p}^{\mathbf{o}_i} &:= \begin{cases} \mathbf{A}^{\mathbf{o}_i} \langle\langle x_0 \rangle\rangle^{o_0}(a_0, x_0)(a_i, x_i) \mathbf{X} \mathbf{X} p & \text{if } p = p_i \\ p & \text{otherwise} \end{cases} \\ \overline{\neg\varphi}^{\mathbf{o}_i} &:= \neg\overline{\varphi}^{\mathbf{o}_i} \\ \overline{\varphi_1 \vee \varphi_2}^{\mathbf{o}_i} &:= \overline{\varphi_1}^{\mathbf{o}_i} \vee \overline{\varphi_2}^{\mathbf{o}_i} \\ \overline{\mathbf{X}\varphi}^{\mathbf{o}_i} &:= \mathbf{X}\overline{\varphi}^{\mathbf{o}_i} \\ \overline{\varphi_1 \mathbf{U} \varphi_2}^{\mathbf{o}_i} &:= \overline{\varphi_1}^{\mathbf{o}_i} \mathbf{U} \overline{\varphi_2}^{\mathbf{o}_i} \\ \overline{\mathbf{E}\psi}^{\mathbf{o}_i} &:= \mathbf{A}^{\mathbf{o}_i} \langle\langle x_0 \rangle\rangle^{o_0}(a_0, x_0)(\mathbf{G} p_S \wedge \overline{\psi}^{\mathbf{o}_i}) \\ \overline{\exists^{\mathbf{o}_j} p_j. \varphi}^{\mathbf{o}_i} &:= \langle\langle x_j \rangle\rangle^{o_j} \overline{\psi}^{\mathbf{o}_j} \end{aligned}$$

where  $\mathbf{A}^{\mathbf{o}_i}$  is a macro for  $\llbracket x_0 \rrbracket^{o_i} \dots \llbracket x_k \rrbracket^{o_i}(a_0, x_0) \dots (a_k, x_k)$ . The cases for path formulas are similar. The universal quantification on paths is just used to obtain a complete assignment, then we redefine the relevant strategies.

We have the following:

**Lemma 5.**  $\mathcal{S} \models \varphi$  iff  $\mathcal{G}_S \models \overline{\varphi}^{[n]}$ .

We observe that if  $\varphi$  is hierarchical, then  $(\overline{\varphi}^{[n]}, \mathcal{G}_S)$  is a hierarchical instance, and:

**Lemma 6.** For every  $p \in \text{AP}_f(\varphi)$  and for every  $i \in [k]$ , if  $t_S(s_\iota)$  is  $\mathbf{o}_i$ -uniform in  $p$  then  $v \sim_{\mathbf{o}_i} v'$  implies that  $p \in \ell(v)$  iff  $p \in \ell(v')$ .

### C. Proof of Theorem 3

**Theorem 3.** The model-checking problem for  $\text{QCTL}_{ii}^*$  under tree-semantics is undecidable.

*Proof.* Let  $\text{MSO}_{\text{eq}}$  denote the extension of the logic  $\text{MSO}$  (without unary predicates) by a binary predicate symbol  $\text{eq}$ .  $\text{MSO}_{\text{eq}}$  is interpreted on the full binary tree, and the semantics of  $\text{eq}(x, y)$  is that  $x$  and  $y$  have the same depth in the tree. We show how to effectively translate  $\text{MSO}_{\text{eq}}$  into  $\text{QCTL}_{ii}^*$ ; then our result follows since the  $\text{MSO}_{\text{eq}}$ -theory of the binary tree is undecidable [17].

The translation of  $\text{MSO}$  to  $\text{QCTL}^*$  from [14] can be extended to one from  $\text{MSO}_{\text{eq}}$  to  $\text{QCTL}_{ii}^*$  by adding rules for the equal level predicate. Indeed, for  $\varphi(x, x_1, \dots, x_i, X_1, \dots, X_j) \in \text{MSO}$ , we inductively define the  $\text{QCTL}_{ii}^*$  formula  $\widehat{\varphi}$  as follows:

$$\begin{aligned} \widehat{x = x_k} &:= p_{x_k} & \widehat{x_k = x_l} &:= \mathbf{EF}(p_{x_k} \wedge p_{x_l}) \\ \widehat{x \in X_k} &:= p_{x_k} & \widehat{x_k \in X_l} &:= \mathbf{EF}(p_{x_k} \wedge p_{X_l}) \\ \widehat{\neg\varphi'} &:= \neg\widehat{\varphi'} & \widehat{\varphi_1 \vee \varphi_2} &:= \widehat{\varphi_1} \vee \widehat{\varphi_2} \\ \widehat{\exists x_k. \varphi'} &:= \exists p_{x_k}. \text{uniq}(p_{x_k}) \wedge \widehat{\varphi'} & \\ \widehat{\exists X_k. \varphi'} &:= \exists p_{X_k}. \widehat{\varphi'} & \\ \widehat{S(x, x_k)} &:= \mathbf{EX} p_{x_k} & \widehat{S(x_k, x)} &:= \perp \\ \widehat{S(x_k, x_l)} &:= \mathbf{EF}(p_{x_k} \wedge \mathbf{EX} p_{x_l}) \end{aligned}$$

where  $\text{uniq}(p) := \mathbf{EF} p \wedge \forall q. (\mathbf{EF}(p \wedge q) \rightarrow \mathbf{AG}(p \rightarrow q))$  holds in a tree iff it has exactly one node labelled with  $p$ . To understand the  $x = x_k$  case, we will interpret  $x$  as the root. To understand the  $S(x_k, x)$  case, observe that  $x$  has no incoming edge since it is interpreted as the root.

The rules for  $\text{eq}$  are as follows:

$$\begin{aligned} \widehat{\text{eq}(x, x_k)} &:= p_{x_k} \\ \widehat{\text{eq}(x_k, x_l)} &:= \exists^\emptyset p. \mathbf{border}(p) \wedge \mathbf{AG}(p_{x_k} \rightarrow p \wedge p_{x_l} \rightarrow p) \end{aligned}$$

To understand the first case, observe that since  $x$  is interpreted as the root,  $x_k$  is on the same level as  $x$  if and only if it is also assigned the root. To understand the second case, recall from Example 2 that the  $\text{QCTL}_{ii}^*$  formula  $\exists^\emptyset p. \mathbf{border}(p)$  places one unique horizontal line of  $p$ 's in the tree, and thus requiring that  $x_k$  and  $x_l$  be both on this line ensures that they are on the same level.

To finish, let  $\mathcal{S}$  be a CKS with two states  $s_0$  and  $s_1$  (local states are irrelevant here), whose transition relation is the complete relation, and with empty labelling function. Clearly,  $t_S(s_0)$  is the full binary tree. It is easy to prove the following by induction:

For every  $\varphi(x, x_1, \dots, x_i, X_1, \dots, X_j) \in \text{MSO}_{\text{eq}}$ ,  $t_S(s) \models \varphi(s, u_1, \dots, u_i, U_1, \dots, U_j)$  if and only if  $t_S(s_0)', s \models \widehat{\varphi}$ , where  $t_S(s_0)'$  is obtained from  $t_S(s_0)$  by changing the labelling for variables  $p_{x_k}$  and  $p_{X_k}$  as follows:  $p_{x_k} \in \ell'(u)$  if  $u = u_k$  and  $p_{X_k} \in \ell'(u)$  if  $u \in U_k$ .

In particular, it follows that

$$t_S(s_0), s_0 \models \varphi(x) \text{ iff } t_S(s_0), s_0 \models \widehat{\varphi}.$$

This completes the proof.

1

#### D. Proof of Lemma 1

Before presenting the proof of Lemma 1 we recall the definition of acceptance by ATA via games between Eve and Adam. Let  $\mathcal{A} = (Q, \delta, q_i, C)$  be an ATA over  $(AP, X)$ -trees, let  $t = (\tau, \ell)$  be such a tree and let  $u_\ell \in \tau$ . We define the parity game  $\mathcal{G}(\mathcal{A}, t, u_\ell) = (V, E, v_\ell, C')$ : the set of positions is  $V = \tau \times Q \times \mathbb{B}^+(X \times Q)$ , the initial position is  $v_\ell = (u_\ell, q_\ell, \delta(q_\ell, u_\ell))$ , and a position  $(u, q, \alpha)$  belongs to Eve if  $\alpha$  is of the form  $\alpha_1 \vee \alpha_2$  or  $[x, q']$ ; otherwise it belongs to Adam. Moves in  $\mathcal{G}(\mathcal{A}, t, u_\ell)$  are defined by the following rules:

$(u, q, \alpha_1 \dagger \alpha_2) \rightarrow (u, q, \alpha_i)$  where  $\dagger \in \{\vee, \wedge\}$  and  $i \in \{1, 2\}$ ,  
 $(u, q, [x, q']) \rightarrow (u \cdot x, q', \delta(q', \ell(u \cdot x)))$

Positions of the form  $(u, q, \top)$  and  $(u, q, \perp)$  are deadlocks, winning for Eve and Adam respectively. The colouring is inherited from the one of the automaton:  $C'(u, q, \alpha) = C(q)$ .

A tree  $t$  is *accepted* in node  $u$  by  $\mathcal{A}$  if Eve has a winning strategy in  $\mathcal{G}(\mathcal{A}, t, u)$ .

We now recall the lemma and the automata construction, and we prove it to be correct.

**Lemma 7** (Translation). *Let  $(\Phi, \mathcal{S})$  be an instance of the model-checking problem for  $\text{QCTL}_{i,\subseteq}^*$ . For every subformula  $\varphi$  of  $\Phi$  and state  $s$  of  $\mathcal{S}$ , one can build an ATA  $\mathcal{A}_s^\varphi$  on  $(\text{AP}_\exists(\Phi), L_\varphi)$ -trees such that for every  $(\text{AP}_\exists(\Phi), L_\varphi)$ -tree  $t$  rooted in  $s \downarrow_{I_\varphi}$ ,*

$$t \in \mathcal{L}(\mathcal{A}_s^\varphi) \text{ iff } t \upharpoonright_y^{[n]} \wedge t_{\mathcal{S}}(s) \models \varphi, \quad \text{where } y = s \downarrow_{[n] \setminus I_\varphi}.$$

For an  $L_I$ -tree  $t$ , from now on  $t \uparrow^{[n]} \mathbin{\text{\texttt{M}}\kern-1.5ex\text{\texttt{M}}} t_S(s)$  stands for  $t \uparrow_y^{[n]} \mathbin{\text{\texttt{M}}\kern-1.5ex\text{\texttt{M}}} t_S(s)$ , where  $y = s \downarrow_{[n] \setminus I}$ : while lifting tree  $t$  to  $[n]$ , the missing local states in the root of  $t$  are filled with those from  $s$ .

*Proof sketch of Lemma 1.* Let  $(\Phi, \mathcal{S})$  be an instance of the model-checking problem for  $\text{QCTL}_{i,\subseteq}^*$ . Let  $\Phi \in \text{QCTL}_{i,\subseteq}^*$ , and let  $\text{AP}_\exists = \text{AP}_\exists(\Phi)$  and  $\text{AP}_f = \text{AP}_f(\Phi)$ , and recall that  $\mathcal{S}$  is labelled over  $\text{AP}_f$ . For each state  $s \in S$  and each subformula  $\varphi$  of  $\Phi$  (note that all subformulas of  $\Phi$  are also hierarchical), we define by induction on  $\varphi$  the ATA  $\mathcal{A}_s^\varphi$  on  $(\text{AP}_\exists, L_\varphi)$ -trees.

$$\varphi = p :$$

We let  $\mathcal{A}_s^p$  be the ATA over  $L_{[n]}$ -trees with one unique state  $q_s$ , with transition function defined as follows:

$$\delta(q_\ell, a) = \begin{cases} \top & \text{if } \begin{array}{l} p \in \text{AP}_f \text{ and } p \in \ell_S(s) \\ \quad \text{or} \\ p \in \text{AP}_\exists \text{ and } p \in a \end{array} \\ \perp & \text{if } \begin{array}{l} p \in \text{AP}_f \text{ and } p \notin \ell_S(s) \\ \quad \text{or} \\ p \in \text{AP}_\exists \text{ and } p \notin a \end{array} \end{cases}$$

$$\varphi = \neg\varphi' :$$

We obtain  $\mathcal{A}_s^\varphi$  by dualising  $\mathcal{A}_s^{\varphi'}$ , a classic operation.

$$\varphi = \varphi_1 \vee \varphi_2 :$$

Because  $I_\varphi = I_{\varphi_1} \cap I_{\varphi_2}$ , and each  $\mathcal{A}_s^{\varphi_i}$  works on  $L_{\varphi_i}$ -trees, we first narrow them so that they work on  $L_\varphi$ -trees: for  $i \in \{1, 2\}$ , we let  $\mathcal{A}_i := \mathcal{A}_s^{\varphi_i} \downarrow_{I_\varphi}$ . We then build  $\mathcal{A}_s^\varphi$  by taking the disjoint union of  $\mathcal{A}_1$  and  $\mathcal{A}_2$  and adding a new initial state that nondeterministically chooses which of  $\mathcal{A}_1$  or  $\mathcal{A}_2$  to execute on the input tree, so that  $\mathcal{L}(\mathcal{A}_s^\varphi) = \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$ .

$$\varphi = \mathbf{E}\psi :$$

The aim is to build an automaton  $\mathcal{A}_s^\varphi$  that works on  $L_\varphi$ -trees and that on input  $t$ , checks for the existence of a path in  $t \uparrow^{[n]} \wedge t_{\mathcal{S}}(s)$  that satisfies  $\psi$ . Observe that a path in  $t \uparrow^{[n]} \wedge t_{\mathcal{S}}(s)$  is a path in  $t_{\mathcal{S}}(s)$ , augmented with the labelling for atomic propositions in  $\text{AP}_{\exists}$  carried by  $t$ .

To do so,  $\mathcal{A}_s^\varphi$  guesses a path  $\lambda$  in  $(\mathcal{S}, s)$ . It remembers the current state in  $\mathcal{S}$ , which provides the labelling for atomic propositions in  $\text{AP}_f$ , and while it guesses  $\lambda$  it follows its  $L_\varphi$ -narrowing in its input tree  $t$  (which always exists since input to tree automata are complete trees), reading the labels to evaluate propositions in  $\text{AP}_\exists$ .

Let  $\max(\psi) = \{\varphi_1, \dots, \varphi_n\}$  be the set of maximal state sub-formulas of  $\psi$ . In a first step we see these maximal state sub-formulas as atomic propositions. The formula  $\psi$  can thus be seen as an LTL formula, and we can build a nondeterministic parity word automaton  $\mathcal{W}^\psi = (Q^\psi, \Delta^\psi, q_0^\psi, C^\psi)$  over alphabet  $2^{\max(\psi)}$  that accepts exactly the models of  $\psi$  [35].<sup>5</sup> We define the ATA  $\mathcal{A}$  that, given as input a  $(\max(\psi), L_\varphi)$ -tree  $t$ , nondeterministically guesses a path  $\lambda$  in  $t \uparrow^{[n]} \mathbin{\text{\texttt{M}}} t_S(s)$  and simulates  $\mathcal{W}^\psi$  on it, assuming that the labels it reads while following  $\lambda \downarrow_{I_\varphi}$  in its input correctly represent the truth value of formulas in  $\max(\psi)$  along  $\lambda$ . Recall that  $S = (S, R, s_i, \ell_S)$ ; we define  $\mathcal{A} := (Q, \delta, q_i, C)$ , where

- $Q = Q^\psi \times S$ ,
  - $q_t = (q_t^\psi, s)$ ,
  - $C(q^\psi, s') = C^\psi(q^\psi)$ , and
  - for each  $(q^\psi, s') \in Q$  and  $a \in 2^{\max(\psi)}$ ,

$$\delta((q^\psi, s'), a) = \bigvee_{q' \in \Delta^\psi(q^\psi, a)} \bigvee_{s'' \in R(s')} [s'' \downarrow_{I_\varphi}, (q', s'')].$$

The intuition is that  $\mathcal{A}$  reads the current label, chooses nondeterministically which transition to take in  $\mathcal{W}^\psi$ , chooses a next state in  $\mathcal{S}$  and proceeds in the corresponding direction in  $X_\varphi$ . Thus,  $\mathcal{A}$  accepts a  $\max(\varphi)$ -labelled  $L_\varphi$ -tree  $t$  iff there is a path in  $t$  that is the  $L_\varphi$ -narrowing of some path in  $t_{\mathcal{S}}(s)$ , and that satisfies  $\psi$ , where maximal state formulas are considered as atomic propositions.

Now from  $\mathcal{A}$  we build the automaton  $\mathcal{A}_s^\varphi$  over  $L_\varphi$ -trees labelled with “real” atomic propositions in  $\text{AP}_\exists$ . In each node it visits,  $\mathcal{A}_s^\varphi$  guesses what should be its labelling over  $\max(\psi)$ , it simulates  $\mathcal{A}$  accordingly, and checks that the guess it made is correct. If the path being guessed in  $t_{\mathcal{S}}(s)$  is currently in node  $u$  ending with state  $s'$ , and  $\mathcal{A}_s^\varphi$  guesses that  $\varphi_i$  holds in

<sup>5</sup>Note that, as usual for nondeterministic word automata, we take the transition function of type  $\Delta^\psi : Q^\psi \times 2^{\max(\psi)} \rightarrow 2^{Q^\psi}$ .

$u$ , it checks this guess by starting a simulation of automaton  $\mathcal{A}_{s'}^{\varphi_i}$  from node  $v = u \downarrow_{I_\varphi}$  in its input  $t$ .

For each  $s' \in \mathcal{S}$  and each  $\varphi_i \in \max(\psi)$  we first build  $\mathcal{A}_{s'}^{\varphi_i}$ , and we let  $\mathcal{A}_{s'}^i := \mathcal{A}_{s'}^{\varphi_i} = (Q_{s'}^i, \delta_{s'}^i, q_{s'}^i, C_{s'}^i)$ . We also let  $\overline{\mathcal{A}_{s'}^i} = (\overline{Q_{s'}^i}, \overline{\delta_{s'}^i}, \overline{q_{s'}^i}, \overline{C_{s'}^i})$  be its dualisation, and we assume w.l.o.g. that all the state sets are pairwise disjoint. Observe that because each  $\varphi_i$  is a maximal state sub-formula, we have  $I_{\varphi_i} = I_\varphi$ , so that we do not need to narrow down automata  $\mathcal{A}_{s'}^i$  and  $\overline{\mathcal{A}_{s'}^i}$ . We define the ATA

$$\mathcal{A}_s^\varphi = (Q \cup \bigcup_{i,s'} Q_{s'}^i \cup \overline{Q_{s'}^i}, \delta', q_*, C'),$$

where the colours of states are left as they were in their original automaton, and  $\delta$  is defined as follows. For states in  $Q_{s'}^i$  (resp.  $\overline{Q_{s'}^i}$ ),  $\delta$  agrees with  $\delta_{s'}^i$  (resp.  $\overline{\delta_{s'}^i}$ ), and for  $(q^\psi, s') \in Q$  and  $a \in 2^{\text{AP}_\exists}$  we let

$$\begin{aligned} \delta'((q^\psi, s'), a) := \bigvee_{a' \in 2^{\max(\psi)}} & \left( \delta((q^\psi, s'), a') \wedge \right. \\ & \bigwedge_{\varphi_i \in a'} \delta_{s'}^i(q_{s'}^i, a) \wedge \\ & \left. \bigwedge_{\varphi_i \notin a'} \overline{\delta_{s'}^i}(\overline{q_{s'}^i}, a) \right). \end{aligned}$$

$\varphi = \exists^0 p \cdot \varphi'$ :

We build automaton  $\mathcal{A}_s^{\varphi'}$  that works on  $L_{\varphi'}$ -trees; because  $\varphi$  is hierarchical, we have that  $\mathbf{o} \subseteq I_{\varphi'}$  and we can narrow down  $\mathcal{A}_s^{\varphi'}$  to work on  $L_{\mathbf{o}}$ -trees and obtain  $\mathcal{A}_1 := \mathcal{A}_s^{\varphi'} \downarrow_{\mathbf{o}}$ . By Theorem 6 we can nondeterminise it to get  $\mathcal{A}_2$ , which by Theorem 5 we can project with respect to  $p$ , finally obtaining  $\mathcal{A}_s^\varphi := \mathcal{A}_2 \Downarrow_p$ .

**Correctness.** We now prove by induction on  $\varphi$  that the construction is correct. In each case, we let  $t = (\tau, \ell)$  be a complete  $(\text{AP}_\exists, L_\varphi)$ -tree rooted in  $s \downarrow_{L_\varphi}$ .

$\varphi = p$ :

First, note that  $I_p = [n]$ , so that  $t$  is rooted in  $s \downarrow_{L_p} = s$ . Let us consider first the case where  $p \in \text{AP}_f$ . By definition of  $\mathcal{A}_s^p$ , we have that  $t \in \mathcal{L}(\mathcal{A}_s^p)$  iff  $p \in \ell_s(s)$ . On the other hand, by definition of the merge operation, of the unfolding, and because  $\text{AP}_\exists$  and  $\text{AP}_f$  are disjoint, we have  $t \uparrow^{[n]} \wedge t_s(s) \models p$  iff  $p \in \ell_s(s)$ , and we are done. Now if  $p \in \text{AP}_\exists$ : by definition of  $\mathcal{A}_s^p$ , we have  $t \in \mathcal{L}(\mathcal{A}_s^p)$  iff  $p \in \ell(s)$ ; also, by definition of the merge, of the unfolding, and because  $\text{AP}_\exists$  and  $\text{AP}_f$  are disjoint, we have that  $t \uparrow^{[n]} \wedge t_s(s) \models p$  iff  $p \in \ell(s)$ , which concludes.

$\varphi = \neg\varphi'$ :

This case is trivial.

$\varphi_1 \vee \varphi_2$ :

We have  $\mathcal{A}_i = \mathcal{A}_{s'}^{\varphi_i} \downarrow_{L_\varphi}$ , so by Theorem 7, for all  $y \in L_{I_{\varphi_i} \setminus I_\varphi}$ , we have  $t \in \mathcal{L}(\mathcal{A}_i)$  iff  $t \uparrow_y^{L_{\varphi_i}} \in \mathcal{L}(\mathcal{A}_{s'}^{\varphi_i})$ , which by

induction hypothesis holds iff  $(t \uparrow_y^{L_{\varphi_i}}) \uparrow^{[n]} \wedge t_s(s) \models \varphi_i$ . Choosing  $y = s \downarrow_{I_{\varphi_i} \setminus I_\varphi}$ , we get that  $t \in \mathcal{L}(\mathcal{A}_i)$  iff  $t \uparrow^{[n]} \wedge t_s(s) \models \varphi_i$ . We conclude by reminding that  $\mathcal{L}(\mathcal{A}_s^\varphi) = \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$ .

$\varphi = \mathbf{E}\psi$ :

Suppose that  $t' = t \uparrow^{[n]} \wedge t_s(s) \models \mathbf{E}\psi$ . There exists a path  $\lambda$  starting at the root  $s$  of  $t'$  such that  $t', \lambda \models \psi$ . Again, let  $\max(\psi)$  be the set of maximal state subformulas of  $\psi$ , and let  $w$  be the infinite word over  $2^{\max(\psi)}$  that agrees with  $\lambda$  on the state formulas in  $\max(\psi)$ . By definition,  $\mathcal{W}^\psi$  has an accepting execution on  $w$ . Now in the acceptance game of  $\mathcal{A}_s^\varphi$  on  $t$ , Eve can guess the path  $\lambda$ , following  $\lambda \downarrow_{X_\varphi}$  in its input  $t$ , and she can also guess the corresponding word  $w$  on  $2^{\max(\psi)}$  and an accepting execution of  $\mathcal{W}^\psi$  on  $w$ . Let  $u' \in t'$  be a node of  $\lambda$ ,  $s'$  its last direction and let  $u = u' \downarrow_{L_\varphi} \in t$ . Assume that in node  $u$  of the input tree, in a state  $(q^\psi, s') \in Q$ , Adam challenges Eve on some  $\varphi_i \in \max(\psi)$  that she assumes to be true in  $u'$ , i.e., Adam chooses the conjunct  $\delta_{s'}^i(q_{s'}^i, a)$ , where  $a$  is the label of  $u$ . Note that in the evaluation game this means that Adam moves to position  $(u, (q^\psi, s'), \delta_{s'}^i(q_{s'}^i, a))$ . We want to show that Eve wins from this position.

Let  $t_u$  (resp.  $t'_{u'}$ ) be the subtree of  $t$  (resp.  $t'$ ) starting in  $u$  (resp.  $u'$ )<sup>6</sup>. It is enough to show that  $t_u$  is accepted by  $\mathcal{A}_{s'}^i = \mathcal{A}_{s'}^{\varphi_i}$ . Observe that  $t_u$  is rooted in the last direction of  $u = u' \downarrow_{L_\varphi}$ , and since the last direction of  $u'$  is  $s'$  we have that  $t_u$  is rooted in  $s' \downarrow_{L_\varphi}$ . Since  $I_\varphi = I_{\varphi_i}$  we have that  $t_u$  is rooted in  $s' \downarrow_{L_{\varphi_i}}$ . We can thus apply the induction hypothesis on  $t_u$  with  $\varphi_i$ , and we get that

$$t_u \in \mathcal{L}(\mathcal{A}_{s'}^{\varphi_i}) \text{ iff } t_u \uparrow^{[n]} \wedge t_s(s') \models \varphi_i. \quad (1)$$

Now, because  $u'$  ends in  $s'$  we also have that

$$t'_{u'} = t_u \uparrow^{[n]} \wedge t_s(s'). \quad (2)$$

Putting (1) and (2) together, we obtain that

$$t_u \in \mathcal{L}(\mathcal{A}_{s'}^i) \text{ iff } t'_{u'} \models \varphi_i. \quad (3)$$

Because we have assumed that Eve guesses  $w$  correctly, we also have that  $t', u' \models \varphi_i$ , i.e.,  $t'_{u'} \models \varphi_i$ . This, together with (3), gives us that  $t_u$  is accepted by  $\mathcal{A}_{s'}^i$ .

Eve thus has a winning strategy from the initial position of the acceptance game of  $\mathcal{A}_{s'}^i$  on  $t_u$ . This initial position is  $(u, q_{s'}^i, \delta_{s'}^i(q_{s'}^i, a))$ . Since  $(u, q_{s'}^i, \delta_{s'}^i(q_{s'}^i, a))$  and  $(u, q, \delta_{s'}^i(q_{s'}^i, a))$  contain the same node  $u$  and transition formula  $\delta_{s'}^i(q_{s'}^i, a)$ , a winning strategy in one of these positions<sup>7</sup> is also a winning strategy in the other, and therefore Eve wins Adam's challenge. With a similar argument, we get that also when Adam challenges Eve on some  $\varphi_i$  assumed not to be true in node  $v$ , Eve wins the challenge. Finally, Eve wins the acceptance game of  $\mathcal{A}_s^\varphi$  on  $t$ , and thus  $t \in \mathcal{L}(\mathcal{A}_s^\varphi)$ .

For the other direction, assume that  $t \in \mathcal{L}(\mathcal{A}_s^\varphi)$ , i.e., Eve wins the evaluation game of  $\mathcal{A}_s^\varphi$  on  $t$ . Again, let  $t' = t \uparrow^{[n]}$

<sup>6</sup>If  $u = w \cdot x$ , a subtree  $t_u$  of  $t = (\tau, \ell)$  is defined as  $t_u := (\tau_u, \ell_u)$  with  $\tau_u = \{x \cdot w' \mid w \cdot x \cdot w' \in \tau\}$ , and  $\ell_u(x \cdot w') = \ell(w \cdot x \cdot w')$ : we remove from each node all directions before last( $u$ ).

<sup>7</sup>Recall that positional strategies are sufficient in parity games [38].

$\wedge t_S(s)$ . A winning strategy for Eve describes a path  $\lambda$  in  $t_S(s)$ , which is also a path in  $t'$ . This winning strategy also defines an infinite word  $w$  over  $2^{\max(\psi)}$  such that  $w$  agrees with  $\lambda$  on the formulas in  $\max(\psi)$ , and it also describes an accepting run of  $\mathcal{W}^\psi$  on  $w$ . Hence  $t', \lambda \models \psi$ , and  $t' \models \varphi$ .

$$\varphi = \exists^0 p. \varphi' :$$

First, observe that because  $\varphi$  is hierarchical, we have that  $I_\varphi = \mathbf{o}$ . Next, by Theorem 5 we have that

$$t \in \mathcal{L}(\mathcal{A}_s^\varphi) \text{ iff there exists } t_p \equiv_p t \text{ s.t. } t_p \in \mathcal{L}(\mathcal{A}_2). \quad (4)$$

By Theorem 6,  $\mathcal{L}(\mathcal{A}_2) = \mathcal{L}(\mathcal{A}_1)$ , and since  $\mathcal{A}_1 = \mathcal{A}_s^{\varphi'} \downarrow_{\mathbf{o}} = \mathcal{A}_s^{\varphi'} \downarrow_{L_\varphi}$  we get by Theorem 7 that

$$t_p \in \mathcal{L}(\mathcal{A}_2) \text{ iff } t_p \uparrow_y^{L_{\varphi'}} \in \mathcal{L}(\mathcal{A}_s^{\varphi'}), \text{ where } y = s \downarrow_{(I_{\varphi'} \setminus I_\varphi)}. \quad (5)$$

Now  $t_p$  and  $t$  have the same root,  $s \downarrow_{L_\varphi}$ . The root of  $t_p \uparrow_y^{L_{\varphi'}}$  is thus  $(s \downarrow_{L_\varphi}, y) = s \downarrow_{L_{\varphi'}}$ , and we can apply the induction hypothesis on  $t_p \uparrow_y^{L_{\varphi'}}$  with  $\varphi'$ :

$$t_p \uparrow_y^{L_{\varphi'}} \in \mathcal{L}(\mathcal{A}_s^{\varphi'}) \text{ iff } (t_p \uparrow_y^{L_{\varphi'}}) \uparrow^{[n]} \wedge t_S(s) \models \varphi'. \quad (6)$$

Now, combining points (4), (5) and (6), together with the fact that  $(t_p \uparrow_y^{L_{\varphi'}}) \uparrow^{[n]} \wedge t_S(s) = t_p \uparrow^{[n]} \wedge t_S(s)$  gives us that

$$t \in \mathcal{L}(\mathcal{A}_s^\varphi) \text{ iff } \exists t_p \equiv_p t \text{ s.t. } t_p \uparrow^{[n]} \wedge t_S(s) \models \varphi'. \quad (7)$$

We now prove equation below which, together with point (7), concludes the proof:

$$\begin{aligned} & \exists t_p \equiv_p t \text{ s.t. } t_p \uparrow^{[n]} \wedge t_S(s) \models \varphi' \\ & \qquad \text{iff} \\ & \qquad t \uparrow^{[n]} \wedge t_S(s) \models \exists^0 p. \varphi' \end{aligned} \quad (8)$$

For the first direction, assume that there exists  $t_p \equiv_p t$  such that  $t_p \uparrow^{[n]} \wedge t_S(s) \models \varphi'$ . First, by definition of the merge, because  $p \in \text{AP}_\exists$  and  $\text{AP}_\exists$  and  $\text{AP}_f$  are disjoint, the  $p$ -labelling of  $t_p \uparrow^{[n]} \wedge t_S(s)$  is determined by the  $p$ -labelling of  $t_p \uparrow^{[n]}$ , which by definition of the widening is  $\mathbf{o}$ -uniform. In addition it is clear that  $t_p \uparrow^{[n]} \wedge t_S(s) \equiv_p t \uparrow^{[n]} \wedge t_S(s)$ , which concludes this direction.

For the other direction, assume that  $t \uparrow^{[n]} \wedge t_S(s) \models \exists^0 p. \varphi'$ : there exists  $t'_p \equiv_p t \uparrow^{[n]} \wedge t_S(s)$  such that  $t'_p$  is  $\mathbf{o}$ -uniform in  $p$  and  $t'_p \models \varphi'$ . Let us write  $t'_p = (\tau', \ell'_p)$  and  $t = (\tau, \ell)$ . We define  $t_p := (\tau, \ell_p)$  where for each  $u \in \tau$ , if there exists  $u' \in \tau'$  such that  $u' \downarrow_{\mathbf{o}} = u$ , we let

$$\ell_p(u) = \begin{cases} \ell(u) \cup \{p\} & \text{if } p \in \ell'_p(u') \\ \ell(u) \setminus \{p\} & \text{otherwise.} \end{cases}$$

This is well defined because  $t'_p$  is  $\mathbf{o}$ -uniform in  $p$ : if two nodes  $u', v'$  project on  $u$ , we have  $u' \approx_{\mathbf{o}} v'$  and thus they agree on  $p$ . In case there is no  $u' \in \tau'$  such that  $u' \downarrow_{L_\varphi} = u$ , we can let  $\ell_p(u) = \ell(u)$  as this node disappears in  $t_p \uparrow^{[n]} \wedge t_S(s)$ . Clearly,  $t_p \equiv_p t$ . Now we write  $t''_p = t_p \uparrow^{[n]} \wedge t_S(s)$  and we prove that  $t''_p = t'_p$  hence  $t''_p \models \varphi'$ , which concludes. It is clear that  $t''_p$  and  $t'_p$  have the same domain. Also, because  $t'_p \equiv_p t \uparrow^{[n]} \wedge t_S(s)$  and  $t''_p = t_p \uparrow^{[n]} \wedge t_S(s)$ , by definition of

the merge both agree with  $t_S(s)$  for all atomic propositions in  $\text{AP}_f$ . Because  $t_p \equiv_p t$ , and again by definition of the merge,  $t''_p$  and  $t'_p$  also agree on all atomic propositions in  $\text{AP}_\exists \setminus \{p\}$ . Finally, by definition of  $t_p$  and because  $t'_p$  is  $\mathbf{o}$ -uniform in  $p$ , we get that  $t''_p$  and  $t'_p$  also agree on  $p$ , and therefore  $t''_p = t'_p$ .  $\square$