Getting the point

Obtaining and understanding fixpoints in model checking

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr.ir. F.P.T. Baaijens, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op woensdag 17 juni 2015 om 16:00 uur

door

Sjoerd Cranen

geboren te Wijchen

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

voorzitter:	prof.dr. E.H.L. Aarts
1^{e} promotor:	prof.dr.ir. J.F. Groote
2^{e} promotor:	prof.dr. J.J. Lukkien
copromotor:	dr. S.P. Luttik
leden:	prof.dr. E. Grädel (RWTH Aachen)
	prof. C.P. Stirling (The University of Edinburgh)
	prof.dr. J.C. van de Pol (Universiteit Twente)
	dr.ir. M.A. Reniers

voor mijn grootouders

Typeset with LaTeX (TeXLive 2014)
Printed by Ipskamp Drukkers, Amsterdam

IPA Dissertation Series 2015-09
ISBN: 978-90-386-3820-1
A catalogue record is available from the Eindhoven University of Technology Library

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics). The author was employed at the Eindhoven University of Technology and supported by the SenterNovem (which became Agentschap NL, which became Rijksdienst voor Ondernemend Nederland) Hightech Automotive Systems (HTAS) programme, project “Verified, Economical and Robust Integrated Functionality for In-vehicle Embedded Development” (VERIFIED), and the NWO project “Verification Of Complex Hierarchical Systems” (VOCHS).

Preface

When I received my master's degree at the university of Eindhoven, a little over six years ago, I was not quite sure what to do next. I had enjoyed my studies, but I eventually decided that I should see what computer science is like outside the university campus. This decision resulted in a job at Imtech ICT Technical Systems (now part of Axians), where I stayed for a year. Still I wondered whether I would have enjoyed pursuing a career in academia instead. I am grateful to professors Jan Friso Groote and Johan Lukkien for giving me the opportunity to answer that question.

Most of the work in this thesis has been carried out within the HTAS 'VERIFIED' project. When I started, I was given the assignment to 'verify the FlexRay protocol', which at first sight seemed a straightforward enough goal. The most relevant document turned out to be a colour-coded 300 page description of the protocol, which I printed on the first day, promptly causing the IT desk to ask me whether I knew there was the option to print in black and white. I realized I was going to need some help understanding what FlexRay was about. I am grateful to Abhijit Deb, Jan Staschulat, Bart Vermeulen, and the others at NXP for having me over to ask questions and discuss my progress.

One person I would like to thank in particular is Reinder Bril, who arranged the first meeting with NXP Eindhoven. Also after that, he has always shown an interest in my progress, and we even ended up writing a paper together. The scheduling problem we looked at still intrigues me; I still keep a document somewhere on my harddrive with some graphs and numbers that seem to suggest the solution cannot be all that complicated, yet I have not been able to find it.

As a student, a course assignment resulted in a paper, thanks to MohammadReza Mousavi and Michel Reniers. When I returned to the university after a year of absence, working with Michel therefore felt familiar, and I would like to thank him for co-authoring my first publication as a PhD student.

Over the past five years I have been working in the Formal System Analysis group at the university of Eindhoven. During that time, I have been given a lot of freedom to pursue my own interests. Especially in the beginning, this was a mixed blessing, as I am easily distracted by interesting problems—of which there are many! I would therefore like to thank Tim Willemse and Bas Luttik not only for working with me and co-authoring the papers on which this thesis is based, but also for taking so much time out of their already very busy schedule to help me organize myself.

Thanks also to Erich Grädel, Colin Stirling and Jaco van de Pol for accepting the

invitation to join the thesis committee. I am aware that this thesis is not exactly a light read, and I appreciate your efforts to go through it all in detail.

I also want to thank the other people I have worked with. One is Jeroen Keiren, who I should not only thank for working with me on topics of research, but also for taking on the project of restructuring and automating the mCRL2 build. It has been an interesting project. Thanks also to Maciej Gazda. We may have to agree to disagree on certain subjects (like progressive rock), but on the whole I have enjoyed our conversations over the past few years.

Sometimes, explaining your problem to someone else is the best way to understand that problem. I am therefore grateful to Hans Zantema for always being available to discuss a problem (as long as I can explain the rules of the game!), and also to Erik de Vink. Even if our discussions have not always led to an immediate solution, they have always helped me understand my own question better.

Tineke van den Bosch–Zaat I want to thank for doing the things that I did not realise I did not have to worry about, and for being so patient with my time registration.

Frank Stappers deserves a special mention. Although we have not done any academic research together, trying to find a way to commercialize formal methods in our own start-up company has taught me a lot.

I would like to thank some other PhD students for our meetings, drinks and games together. In no particular order, I would like to thank Maarten Manders for the long IPA nights, Neda Noroozi for interesting discussions about absolutely anything, Ulyana Tikhonova, Ana-Maria Sutii, Dana Zhang and Yaping Luo for their hands-on research on gossip protocols, Yanja Dajsuren for the cooperation that time I did Cars in Context, Bogdan 'mr. chairman' Vasilescu for his neverending appetite, Sarmen Keshishzadeh for being the most sensible person in my office, Mahmoud Talebi for the frog and the monkey, Fei Yang for the oranges, Önder Babur for appreciating philosophy, Sebastiaan Joosten for his magic trick at the PhD meeting, and Josh Mengerink for being louder than me and Maarten together (IPA needs you!).

Finally, I want to thank my friends and family for their support. Bram, Emma, Maarten, Marianne, Mark, Paul and Thomas, thanks for staying in touch, it means a lot to me. Mum, dad, Phil and Bren, thank you for always being there for me and Rosa. And Rosa, thank you so much for sharing your life with me.

Contents

Glossary

In this thesis you will find many definitions and notational conventions. Some of these are used across multiple chapters, while others are only used within the chapter they are defined. To aid the reader, they are listed here per category. Due to readability considerations, some symbols may have a different meaning depending on the context they are used in.

General (2.1)

$\mathbb{B}$	The set of Booleans $\{\mathbb{t}, \mathbb{f}\}$
$\mathbb{t}$	Boolean 'true', member of $\mathbb{B}$
$\mathbb{f}$	Boolean 'false', member of $\mathbb{B}$
$\mathbb{N}$	The set of natural numbers

First-order logic (2.1)

$\mathcal{V}$	Set of variables
x, y, z, x', y', z'	Variables from $\mathcal{V}$
Σ	Signature, typically $\langle \mathcal{R}, \mathcal{F}, \mathsf{ar} \rangle$
$\mathcal{R}$	Set of relation symbols
$\mathcal{F}$	Set of function symbols
ar	Arity function
$\mathfrak{A}, \mathfrak{B}, \mathfrak{C}$	Structures, typically $\mathfrak{A} = \langle \Sigma, A, \mathcal{I} \rangle$
A, B, C	Domains of discourse of $\mathfrak{A}, \mathfrak{B}, \mathfrak{C}$
$\mathcal{I}$	Interpretation function
θ, η, θ'	Environments assigning values to variables
$\theta \equiv_V \eta$	θ and η agree on the variables in V
$\theta[x \mapsto a]$	The environment θ, updated to map x to a
t, t_1, t'	Terms, built from function symbols and variables
$t^{\mathfrak{A},\theta}$	Interpretation of a term
$\varphi, \psi, \chi, \varphi'$	First-order formulas
$\varphi \sqsubset \psi$, $\varphi \sqsubseteq \psi$	φ is a (strict) subformula of ψ
$\mathsf{sf}(\varphi)$	The set of subformulas of φ
$\mathfrak{A}, \theta \models \varphi$, $\mathfrak{A}, \theta \not\models \varphi$	Formula φ holds / does not hold in $\mathfrak{A}$ and θ

Automata (2.2)

$\mathfrak{A}$	Kripke structure or LTS
A	The states of $\mathfrak{A}$
a, a'	States in A
a_0	Initial state of $\mathfrak{A}$
$\mathcal{B}$	Büchi automaton, typically $\langle B, L, \to, F, B_0 \rangle$
	Generalized Büchi automaton, typically $\langle B, L, \to, \mathcal{F}, B_0 \rangle$, see also **Temporal logics** (2.4)
B	The states of $\mathcal{B}$
L	The labels of $\mathcal{B}$
F	The accepting states of $\mathcal{B}$
$\mathcal{F}$	The acceptance sets of generalized Büchi automaton $\mathcal{B}$
b, b'	States in B
b_0, B_0	Initial state(s) of $\mathcal{B}$
ℓ	State labelling function
AP	Set of atomic predicates
$\to$	Transition relation
$a \to b$	Shorthand for $(a, b) \in \to$
$a \xrightarrow{l} b$	Shorthand for $(a, l, b) \in \to$
p, q, p', q'	Paths
p_i	The i-th state in p
p^i	The path p minus its first i states
pq	Path concatenation

Lattice operations (2.3)

α, β, γ	Ordinals
$\bigsqcap A$	Infimum of A
$\bigsqcup A$	Supremum of A
$a \sqcap a'$	Meet of a and a' (infimum of $\{a, a'\}$)
$a \sqcup a'$	Join of a and a' (supremum of $\{a, a'\}$)
$\mathbf{lfp}\, F$	The least fixpoint of F
$\mathbf{gfp}\, F$	The greatest fixpoint of F
$\mathbf{lfp}^\alpha F$	The α-th approximation of $\mathbf{lfp}\, F$
$\mathbf{gfp}^\alpha F$	The α-th approximation of $\mathbf{gfp}\, F$

Temporal logics (2.4)

$\mathfrak{A}, a \models \varphi$	φ holds / does not hold in state a of $\mathfrak{A}$
$\mathfrak{A}, a \not\models \varphi$	
$a \models \varphi$	φ holds / does not hold in state a of $\mathfrak{A}$,
$a \not\models \varphi$	if $\mathfrak{A}$ is clear from the context
$\mathcal{B}$	(Generalized) Büchi automaton, see also **Automata**
$\mathcal{B}_f$	LTL automaton for LTL formula f (4.1)
f, g	Formula of CTL* or ECTL* (or any sublogic thereof)
A, E	CTL* path quantifiers

Temporal logics (2.4)

$\mathsf{X}, \mathsf{U}, \mathsf{R}, \mathsf{F}, \mathsf{G}$	CTL* temporal operators
$\mathbf{t}$	Formula that holds in all states
$\mathbf{f}$	Formula that holds in no state
$\mathfrak{D}$	Structure used to interpret first-order part of FO-L$_\mu$ (4.2.1)
$\mathbb{D}$	Domain of discourse of $\mathfrak{D}$
$\mathcal{S}$	Set of sorts
D, D_i	Sorts
$[\![D]\!]$	Semantic set associated with sort D
$\mathcal{X}$	Set of predicate variables
δ, δ'	Data environment (first-order environment)
θ, θ'	Predicate environment
$[\![t]\!]^\delta$	Alternative syntax for $t^{\mathfrak{D},\delta}$
φ, χ, ψ	Formula of FO-L$_\mu$ (4.2.2)
$[\![\varphi]\!]^{\theta\delta}$	Interpretation of a FO-L$_\mu$ formula φ
$[\cdot], \langle\cdot\rangle$	FO-L$_\mu$ modal operators
$\nu X(d_1 : D_1 = t_1, \ldots) \,.\, \varphi,\ \mu X(d_1 : D_1 = t_1, \ldots) \,.\, \varphi$	FO-L$_\mu$ fixpoint operators

Parity games (5.1)

$\Diamond$	Player even
$\Box$	Player odd
i	A player from $\{\Diamond, \Box\}$
$\neg i$	The opponent of i
G, G'	A parity game $\langle V, \to, \Pi, \Omega\rangle$
V, V', V^*, U, T	Sets of vertices
$\mathcal{U}, \mathcal{T}$	Set of sets of vertices
v, v', u, w	Vertices
$V_{/R}$	The partition of V induced by equivalence relation R
$[v]_R$	The equivalence class of v under R
$\Pi(v)$	The player who owns vertex v
$\Omega(v)$	The priority associated with vertex v
$\mathbb{S}^*_{G,i}, \mathbb{S}^*_i$	The set of all strategies for player i on G
$\mathbb{S}_{G,i}, \mathbb{S}_i$	The set of memoryless strategies for player i on G
s, s', s'	Strategies
$s \Vdash p$	Path p is consistent with strategy s
$v \xrightarrow{s} u$	Paths consistent with s may contain the edge (v, u)
$v \overset{U}{\longmapsto} T, v \overset{U}{\longmapsto} \mathcal{U}$	There is a stutter path from v to T (resp. $\mathcal{U}$) through U
$v \overset{U}{\longmapsto}$	There is a path from v that stays in U
$v \overset{s,U}{\longmapsto} T, v \overset{s,U}{\longmapsto} \mathcal{U}$	Strategy s forces play from v via U to T (resp. $\mathcal{U}$)
$v \overset{s,U}{\longmapsto}$	Strategy s forces play from v to stay in U
$v \overset{i,U}{\longmapsto} T, v \overset{i,U}{\longmapsto} \mathcal{U}$	Player i can force play from v via U to T (resp. $\mathcal{U}$)
$v \overset{i,U}{\longmapsto}$	Player i can force play from v to stay in U
$v \equiv u$	v is related to u by an isomorphism (5.3)
$v \leftrightarroweq u$	v is related to u by a bisimulation

Parity games (ctd.) (5.3)

$v \simeq u$	v is related to u by a stuttering bisimulation
$v \leftrightarrows u$	v is related to u by a governed bisimulation
$v \eqsim u$	v is related to u by a governed stuttering bisimulation
$v \sim u$	v and u are won by the same player

Equational fixpoint logic (6.1)

$\mathbf{lfp} X\bar{x} = \varphi$	
$\mathbf{gfp} X\bar{x} = \varphi$	Fixpoint equations
ε	The empty equation system
$\mathcal{E}, \mathcal{E}', \mathcal{F}$	Fixpoint equation systems
$\mathcal{E}^i$	The equation system $\mathcal{E}$ without its first i equations
$[X\bar{t} : \mathcal{E}]$	The EFL fixpoint operator
$X\bar{t}$	Shorthand for $[X\bar{t} : \varepsilon]$
$\mathcal{F} \sqsubset \mathcal{E}$	$\mathcal{F}$ is a subsystem of $\mathcal{E}$
$\mathsf{bnd}(\mathcal{E})$	The variables in the left-hand sides of the equations in $\mathcal{E}$
$\mathsf{bnd}^*(\mathcal{E})$, $\mathsf{bnd}^*(\varphi)$	The variables bound in $\mathcal{E}$ (resp. φ) and its subsystems
$\mathsf{fv}(\varphi)$, $\mathsf{fv}(\varphi)$	The variables occurring freely in $\mathcal{E}$ (resp. φ)
$\sigma_X, \bar{x}_X, \varphi_X$	The fixpoint symbol, variables and right-hand side associated with X
$<_{\mathcal{E}}, \leq_{\mathcal{E}}$	Orderings on $\mathsf{bnd}^*(\mathcal{E})$
$\mathbf{S}^{\mathfrak{A}}(\theta, \mathcal{E})$	The solution of $\mathcal{E}$ in $\mathfrak{A}$ and θ
$\mathbf{T}_{\mathcal{E}}^{\mathfrak{A},\theta}$	The predicate transformer for $\mathcal{E}$ in $\mathfrak{A}$ and θ
$\mathsf{fo}(\varphi)$	The first-order part of φ (in which equation systems in φ are replaced by ε) (6.4)

Proof graphs (7.1)

S, S', S^*	Sets of proof graph nodes
v, v', u, u'	Proof graph nodes, typically $\langle \alpha, X, \bar{a} \rangle$
α, β, γ	Booleans
$\mathcal{S}$	Set containing the nodes of any proof graph
$\mathcal{S}_{\mathcal{Y}}$	Subset of $\mathcal{S}$ containing only nodes with second-order variables from $\mathcal{Y}$
$\mathfrak{A} \restriction S$	The smallest submodel of $\mathfrak{A}$ containing all elements of A occurring in S
$\mathfrak{A}, \theta \models v$	The statement that node v represents holds in $\mathfrak{A}$ and θ
$\mathfrak{A}, \theta \overset{\alpha}{\models} \varphi$	Shorthand for $\mathfrak{A}, \theta \models \varphi \Leftrightarrow \alpha = \mathbb{t}$
$v^\bullet$	The set of successors (postset) of v

Chapter 1

Introduction

Building correctly functioning software, or correctly functioning logical circuits, is difficult. This is not only witnessed by the ever growing list of software bugs that caused great financial or material damage, but also by the amount of effort that programmers and circuit designers (let us call them 'designers') spend to find mistakes in their designs.

To detect mistakes, designers often employ some form of testing. Testing consists of 'trying out' the design—or parts of the design—a number of times, each time providing different inputs to it. For a program that calculates a new value based on a set of input values, this can be done by trying the program with a number of different input values and checking that the results it provides are correct. The design of a logical circuit can be tested in a similar way: by putting different signals on the input pins, and measuring and checking the values of the output pins, one can gain a certain degree of confidence that the design is correct. For control systems that take their input from sensors and send their output to actuators, one can try out the system a number of times by supplying fake sensor information for a period of time, and measuring that during that time the system indeed generates the right actuator signals.

Testing is an empirical technique that can be used in almost any field of science. For example, most consumer products are subject to some form of testing. If you work in an office, the chances are that a prototype of your chair has been subjected to thousands of 'sitting down' movements by a machine, to ensure that the chair will not collapse when you sit down on it, three years after you purchased it. While at first sight it may seem silly to have a machine that can do nothing more than sit on a chair all day, there is a very practical reason to build it: it is the simplest way to establish that the chair meets the demands. Even though we know a lot about materials and mechanics, it is very difficult to guarantee, just by looking at the design, that an office chair with all its adjustable parts can withstand a certain load.

Logical designs, such as computer programs and circuit designs (at gate level, not the physical circuit designs), have a rather special property. Unlike our chair, the quality of software is not dependent on the quality of welds, fluctuations in the density of polymer parts or the amount of time that friction-reducing grease can do its job. A logical design consists of only components that behave in a completely predictable way, and which still behave predictably when combined to form a larger whole.

For example, the line of C code '`x = 1;`' performs a very predictable task: after

this line is executed, the value of x is equal to 1, whatever its value was before. If a computer executes this program, and does not behave in this manner, then it is fair to say that the computer has in fact executed a different program, or at least that it has failed to execute this one. Likewise, in a logical circuit, if the two inputs of a binary and-gate are given a high signal, then the output is also high. If you find an and-gate that does not have this behaviour, then for all practical intents and purposes, it is fair to say that it is in fact not an and-gate, whatever it says on the label. In short: logical designs do not just exhibit behaviour, they *define* it.

The field of *formal methods* tries to exploit this property, by trying to *calculate* whether a design is correct, rather than trying to establish this by testing the design. We can establish that an apple and an orange hit the ground at the same time if we drop them from a church tower at the same moment, without actually performing this experiment. We can do this because scientists, presumably grown tired of being bombarded with fruit, created a *model* that describes the movement of falling apples and oranges. If we repeat this experiment in practice, it may turn out that the two fruits do not hit the ground at precisely the same moment, because the model is not perfect: maybe we did not have an exact measurement on the gravity, perhaps we did not account for air friction, or maybe the moon was just a bit too close to the earth at the time of the experiment.

While the gravity model for our falling fruits may be inaccurate in places, the complete predictability of the components of our logical designs allow us to create perfect behavioural models for these designs. Using formal methods, one can therefore hope to calculate perfect answers to questions we might have about that behaviour.

There is much more to be said about the applicability of formal methods, testing and the combination of the two, but this discussion is not in the scope of this thesis. The work in this thesis is based on the premise that formal methods are useful for detecting errors in logical designs. The rest of this chapter is intended to make the context of the results in this thesis clearer, and to explicitly state the research questions that are addressed in subsequent chapters.

Section 1.1 sketches the context in which this work was done. It explains the origin of some of the concepts that recur in later chapters and attempts to give an intuition to the most basic notions used there. Section 1.2 explains why fixpoints are of interest when checking logical designs for errors, and illustrates how fixpoint logics can be used to encode verification problems. With this context in mind, the research questions that are treated in this thesis are stated in Section 1.3.

1.1 Model checking, equivalence checking, and process algebras

If the behaviour of a design is given as a mathematical structure, and a question about that design is given by a formula of some temporal logic, *model checking* can often be employed to calculate the answer to the question. The term was coined by Clarke and Emerson in 1981 [CE82], and is elucidated 27 years later as follows.

> We used the term *Model Checking* because we wanted to determine if the temporal formula f was true in the Kripke structure M, i.e., whether the structure M was a model for the formula f. Some people believe erroneously that the use of the term "model" refers to the dictionary meaning of this word [...] and indicates that we are dealing with an abstraction of the actual system under study. — *E.M. Clarke* [Cla08]

Nowadays, the term is often used in a less strict sense; M is often not a Kripke structure, and it is commonly accepted that checking the validity of a first-order formula in a given structure is also an instance of the model checking problem. Two aspects, however, are universally agreed on:

– a model checker exhaustively calculates whether a model satisfies some property, and
– this calculation is done mechanically.

Note that we use the word 'model' in the aforementioned dictionary meaning of the word here; we use it to refer to a description of a design. In most cases, it is assumed that a model is a transition system: a directed graph, with labels on the nodes and/or edges that represent observations that can be made about the system.

As a small example, we will look at a model of a computer science professor. Note that, anno 2014, a computer science professor is neither a computer program nor a logical circuit, and our model may therefore be inaccurate. Any questions we answer about the model may therefore not be entirely accurate either. Our model, which we shall name $\mathfrak{A}$, is the following transition system:

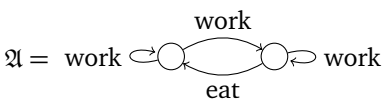


Figure 1.1: A possibly inaccurate model of a computer science professor.

Such a model should be viewed as a description of behaviour: if the professor is in the left state, he will do some work (which we can observe), after which he either is still in the left state, or has moved to the right state; if he is in the right state, he will either do some work and stay in the right state, or he will eat something (this too we can observe) after which he is back in the left state. The observable actions of the professor in this case are 'work' and 'eat'.

A simple question one could ask is 'will this professor always do more work eventually?' The answer is of course 'yes': in the left state, the professor will always do more work, and in the right state, if the professor chooses not to do any work but instead eats something, he will end up in the left state again in which he has no choice but to work again. Note that it is not true of this professor that he will eventually always eat something: he might stay in the left state indefinitely.

Industrial systems can be mapped to a behavioural model in much the same way as the professor. Observable actions of an X-ray machine may be 'radiate', 'show error',

'notice button press', and so on. Properties of interest could be of the form 'will the X-ray machine eventually always radiate the patient when it has noticed the button being pressed?'

Answering questions about the model of our professor is easy enough to do manually; we do not need the help of a computer to calculate the answers. However, for an industrial system, the number of states in its behavioural model is typically too large to compute answers manually. Later, we will take a closer look at one method to compute them mechanically, but first we will look at ways to express properties of interest about a given system. For an overview of developments in model checking, we refer to [Cla08] and the references therein.

1.1.1 Modal logic and equivalences

We said before that the 'work' and 'eat' actions are the only things we can observe about our professor. To see why this is reasonable, suppose that in the left state in Figure 1.1, the professor is content, but in the right state, the professor is hungry.

Because we assumed that the professor's working and eating are the only observable actions, we have in particular assumed that we cannot discern anything else about the professor: we cannot ask him how he feels, and we cannot learn anything from his facial expression (the only information we can use, is the information in Figure 1.1). More specifically, this means that we cannot directly observe that the professor is hungry. At the very best we can deduce, after observing the professor eat, that he must have been hungry.

Given that no direct queries about the internal state of the professor are possible, a natural question is to ask how we can express properties that only reason about the observable behaviour. This question is answered by modal logics. Such logics reason about the observable behaviour from the viewpoint of a state of the system. For instance, although we may not ask 'is the professor hungry?', we are allowed to ask 'can the professor eat something in his current state?' This question is posed only in terms of observable events, and is therefore allowed. Note that we are able to distinguish the states in Figure 1.1, because this question is answered differently for both states. For an overview of modal logics, we refer the interested reader to [BBW06]. Brief descriptions of the syntax and semantics of the modal logics that are used in this thesis are given in Section 2.4.

Modal logics distinguish states of a system based on their observable behaviour. Therefore, two distinct states may satisfy the same formulas of a given modal logic, because their behaviour is the same. This gives rise to the notion of behaviour-based equivalences on transition systems: if the only thing we can observe about a system (the professor) is the actions it performs, and from one state the system can perform exactly the same actions, and in the same order, as from another state, then we cannot distinguish those two states.

We illustrate the concept with an extended model $\mathfrak{A}'$ of our professor. To be able to refer to the states of the model in the text, we annotate each state with a name, but we do not consider these names to be part of the model. The model is a variant of the one in Figure 1.1. The right state in that figure is called r here, and we have

split the left state into two states, called l_1 and l_2. Suppose that in l_1 the professor is not hungry, and in l_2, the professor is hungry, but also under a lot of pressure to finish some work. In l_1, therefore, the professor must first do some work before he is hungry enough to consider eating something, while in l_2, the professor is hungry, but first does some work to release his stress. Let us assume that eating has a calming effect on the professor, so after eating he returns to l_1: calm and not hungry. His behaviour is depicted in Figure 1.2.

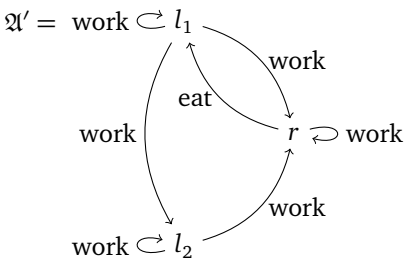


Figure 1.2: A three-state model of a computer science professor.

Now, if we can only observe the professor's working and eating, there is no way for us to tell whether he is in l_1 or in l_2 by observing his future behaviour. We can detect the difference between l_1 and r and between l_2 and r, because in r, the professor has the possibility of eating, which he does not have in l_1 and l_2. But from l_1 and l_2, the professor has the exact same possibilities: either he performs some work and ends up in a state that has the same possibilities as l_1 and l_2 again (namely l_1 or l_2), or he performs some work and ends up in r (which we can identify as being a different state). More formally, the states are divided into two *equivalence classes*, $\{l_1, l_2\}$ and $\{r\}$, and we can only distinguish two states if they are in different equivalence classes. The notion of equivalence that is illustrated here is called *bisimilarity*: in our example, l_1 and l_2 are bisimilar.

Bisimulation was first studied in the context of concurrent programming [Par81] (an almost identical notion was proposed by Milner [Mil80]), where it was used in an automata-theoretic setting to prove properties of concurrent programs. By showing that the initial states of automata induced by two different programs are bisimilar, Park was able to show that these two programs gave rise to the same execution traces.

Apart from bisimulation, one can think of many similar concepts. Simulation, for instance, is the asymmetric counterpart of bisimulation: a state simulates another state if it can display all the behaviour that the other state has. Weaker forms of simulation and bisimulation have also been investigated, in which for instance some actions have been marked as 'unobservable' (this is useful when composing a model from several smaller models). An overview of these relations is given in [Gla93].

Notions of bisimilarity and similarity can be used in the analysis of systems. One approach is to make a much simplified view of your system in the form of a transition system, and checking that the actual system in some sense simulates the simplified one.

This approach is followed by the refinement checker FDR [Gib+14]. Conversely, one can try to extract a smaller system from a larger one, such that the larger one is related via some simulation or bisimulation to the smaller one. Under certain conditions, verifying that properties hold on the smaller system is sufficient to conclude that these properties also hold on the larger system.

1.1.2 Process algebra

Until now, we have only said that a system model is usually some kind of directed graph. Our examples were small enough to be able to simply draw this graph. Obviously, this manner of presenting models is inept when we are talking about systems with millions of states. In such cases, we want a more compact representation.

There are various ways to do this, each with their own advantages and disadvantages. One particular strategy that had a great influence on the model checking tools used in this thesis (we will touch upon this in the next section), is to write down such a graph as a term in a process algebra. A process algebra consists of a collection of *actions*, which form the set underlying the algebra, together with a few operations on those primitives, such as sequential composition, alternative composition, and parallel composition. To make the notation more compact, and to allow the process algebra to express infinite-state systems, a notion of 'data' can be added. A brief overview of the history of process algebras is sketched in [Bae05]. We refer the interested reader to [BPS01] for more information about process algebras.

In the remainder of this section, we illustrate the use of process algebra as a specification language. We first illustrate the use of process algebra without data, and then show how data can be used to create a more compact description.

Let us take another look at the second model of the professor. Our set of actions is $\{\text{work}, \text{eat}\}$. Let us denote sequential composition of actions by $\cdot$, and alternative composition of (nondeterministic choice between) processes by $+$. The states l_1, l_2 and r are then represented by the unique solution to the equations for Professor, StressedProfessor and HungryProfessor, respectively, in the following system of process equations:

$$\begin{aligned}
\text{Professor} &= \text{work} \cdot \text{Professor} \\
&\quad + \text{work} \cdot \text{HungryProfessor} \\
&\quad + \text{work} \cdot \text{StressedProfessor}; \\
\text{StressedProfessor} &= \text{work} \cdot \text{StressedProfessor} \\
&\quad + \text{work} \cdot \text{HungryProfessor}; \\
\text{HungryProfessor} &= \text{work} \cdot \text{HungryProfessor} \\
&\quad + \text{eat} \cdot \text{Professor};
\end{aligned}$$

This specification says, for instance, that the Professor process (which represents state l_1) can choose to do either of three things: it can choose to perform a 'work' action and then behave like the Professor process again, it can perform a 'work' action and behave like the HungryProfessor process (representing r) or it can perform a 'work' action and behave like the StressedProfessor process (representing l_2). For the precise

semantics of the above (which is in fact a term of the mCRL2 process algebra), see [GM14].

To conclude our section about process algebra, we have a brief look at a notion of data for process algebra, which will make the expression above more compact.

The idea is to parameterize process names with a number of first-order variables, representing some element of a first-order structure $\mathfrak{A}$. The right hand side of the process equation is then a process algebra term in which the functions and relations of $\mathfrak{A}$ may be used. With the introduction of data also comes a generalisation of the '+' operator, denoted by the $\sum$ sign, and a conditional operator $f \to P_1 \diamond P_2$, where f is a first-order logic formula over $\mathfrak{A}$ (in which free first-order variables may occur). The conditional process $f \to P_1 \diamond P_2$ is equal to P_1 if f holds in $\mathfrak{A}$, and is equal to P_2 otherwise. The notation $f \to P_1$ is shorthand for $f \to P_1 \diamond \delta$, where δ, the deadlock process, is the identity element for the '+' and $\sum$ operators.

Suppose we are working in the context of a first-order structure that contains the elements $\{\text{Content}, \text{Stressed}, \text{Hungry}\}$, in which $\approx$ is the identity relation and $>$ is a binary relation such that $\text{Hungry} > \text{Stressed} > \text{Content}$. The model of the professor can then be expressed by the process term in Figure 1.3.

$$\begin{aligned}\text{Professor}(m) = \sum_{m'} m' > m \vee m' \approx m \to \text{work} \cdot \text{Professor}(m') \\ + m \approx \text{Hungry} \to \text{eat} \cdot \text{Professor}(\text{Content})\end{aligned}$$

Figure 1.3: The computer science professor as a process with data.

The process 'Professor(Hungry)' from Figure 1.3 specifies the same process as 'HungryProfessor' earlier; this can be seen by substituting 'Hungry' for m and expanding $\sum_{m'} m' > m \to \text{work} \cdot \text{Professor}(m')$ to

$$\begin{aligned}&\text{Content} > \text{Hungry} \vee \text{Content} \approx \text{Hungry} \to \text{work} \cdot \text{Professor}(\text{Content}) \diamond \delta \\ &+ \text{Stressed} > \text{Hungry} \vee \text{Content} \approx \text{Hungry} \to \text{work} \cdot \text{Professor}(\text{Stressed}) \diamond \delta \\ &+ \text{Hungry} > \text{Hungry} \vee \text{Hungry} \approx \text{Hungry} \to \text{work} \cdot \text{Professor}(\text{Hungry}) \diamond \delta,\end{aligned}$$

which is bisimilar to $\delta + \delta + \text{work} \cdot \text{Professor}(\text{Hungry})$, which is again bisimilar to $\text{work} \cdot \text{Professor}(\text{Hungry})$. Similar expansions can be done for the other values of m, yielding processes bisimilar to those given before.

Aside from being a convient abbreviation mechanism, the quantification provided by the summation operator brings something new to the table. Where regular process algebra only has alternative composition ('+') and hence can only specify finite branching structures, the process $X(m) = \sum_{m'} a \cdot X(m')$ is infinitely branching if $\mathfrak{A}$ is infinite.

Finally, we note that this data extension can also be done in a many-sorted setting. In this case, the condition of the conditional process operator is an expression of many sorted first-order logic, and all variables are assigned a sort. This approach is taken

by the ACP-based process algebra mCRL2 [GM14] that is used in Chapter 3 to create a system model.

1.2 Fixpoints

There are many ways to solve the model checking problem, but for this thesis, one method is of particular importance: encoding the problem in a formula of some fixpoint logic, and then solving that formula.

One of the most expressive modal logics is the (modal) μ-calculus, also written L_μ, introduced by Kozen in 1983 [Koz83]. Emerson and Lei first gave a model checking algorithm for L_μ in [EL86], praising the logic for its ability to capture fairness constraints, and showing that a number of other temporal logics (CTL, FCTL, PDL and PDL-Δ) could be translated into L_μ. They stressed that these translations are *succinct*, in the sense that the size of the resulting L_μ formula is linear in the size of the original formula. Hence their model checking algorithm provided a fairly efficient way to model check formulas from all these logics.

The expressive power of L_μ lies not in its modal operators (only the standard box and diamond modality from modal logic are present, where CTL* in addition has operators like F and U, and PDL-Δ has the Kleene star and Δ operator). It is the least fixpoint operator μ and greatest fixpoint operator ν that, combined with these simple box and diamond modalities, make it possible to express any linear and branching time property.

Example. Returning to the model of our professor, we might want to know whether there is a possibility that at some point in the future, the professor is able to eat something. This can be expressed by the following L_μ formula.

$$\mu X \,.\, \langle \text{eat} \rangle \mathbf{t} \vee \langle \cdot \rangle X$$

To see how this formula encodes the property of interest, we informally explain its semantics. Every L_μ formula represents the set of states of the model under scrutiny (in our case, the model of the professor) in which the formula holds. The formula $\mathbf{t}$ ('true') is defined to hold in every state. The diamond modality $\langle \text{eat} \rangle \varphi$ is true for those states that have an outgoing edge labelled 'eat' to a state in which φ holds. In particular, $\langle \text{eat} \rangle \mathbf{t}$ therefore represents all states that have an outgoing 'eat' transition.

The formula $\langle \cdot \rangle X$ says that there is an outgoing transition to a state in which X holds (the '·' can be seen as a wildcard; in our case $\langle \cdot \rangle X$ can be read as an abbreviation of $\langle \text{eat} \rangle X \vee \langle \text{work} \rangle X$, as 'eat' and 'work' are the only actions in our system). However, X is a variable, which has the potential to represent any set of states.

In this formula, the X is bound by a *least fixpoint operator*, μ, which defines the set of states that X represents. It requires that X is the smallest set of states such that $\langle \text{eat} \rangle \mathbf{t} \vee \langle \cdot \rangle X$ holds for exactly the states in X. Let us try to find this set of states, given the model of the professor of Figure 1.2.

We are looking for a set of states that contains all states in which $\langle \text{eat} \rangle \mathbf{t} \vee \langle \cdot \rangle X$ holds. In particular, this set must always include the set of states in which $\langle \text{eat} \rangle \mathbf{t}$ holds. This is

the singleton set $\{r\}$, because r is the only state from which an 'eat' action is possible. We therefore conclude that X represents a superset of $\{r\}$. By the same reasoning, X must also include the states in which $\langle\cdot\rangle X$ holds. Because we know that X holds in r, X must therefore also hold in any state that can reach r in one step. This means that X must also hold in l_1 and l_2, so the set represented by X is a superset of $\{l_1, l_2, r\}$. These are all the states in the system, and so we conclude that our property holds for all states of the professor, or in other words, that $\mathfrak{A}$ is a model for the formula $\mu X \,.\, \langle\text{eat}\rangle\mathbf{t} \vee \langle\cdot\rangle X$. Note that if $X = \{l_1, l_2, r\}$, then indeed every state in this set has either an outgoing transition labelled 'eat' or an outgoing transition to a state in which X holds. ■

This example shows how the least fixpoint operator in L_μ can be employed to express a reachability property: we used it to characterize those states in $\mathfrak{A}$ from which in a finite number of steps a state can be reached from which an 'eat' action is possible. Dually, the *greatest fixpoint operator* ν can be used to express invariants. Combinations of these two fixpoint operators allow for even more complex properties (involving, for instance, fairness. We will not go into detail here, for an introduction to L_μ see, *e.g.*, [CGP99]).

1.2.1 Boolean equation systems

The L_μ model checking problem can be encoded into a formula of an equational, propositional (non-modal) fixpoint logic called Boolean equation systems (BES) [And92; Mad97]. This logic corresponds to L_μ without the modal operators, which are translated to ordinary conjunctions and disjunctions in the translation procedure. Roughly speaking, the idea is to introduce an equation for every state of the system, of which the right hand side corresponds to the L_μ formula. Because each equation is associated with a unique state, the modalities in the L_μ formula can be resolved: a formula $\langle\cdot\rangle f$ can be translated to a disjunction of statements, one for each successor of the current state, that are true if and only if f holds in the corresponding successor state.

The result of the encoding is a system of Boolean fixpoint equations that equate a set of Boolean variables with propositional formulas in which those variables again occur. The solutions of the variables in the system then correspond to the solution to the model checking problem. The details of this translation are given in Maders thesis [*ibid.*], we only illustrate the idea with an example.

Example. To translate the formula $\varphi = \mu X \,.\, \langle\text{eat}\rangle\mathbf{t} \vee \langle\cdot\rangle X$ from the previous example to a Boolean equation system, we introduce an equation $\mu X_s = \varphi_s$ for each state $s \in \{l_1, l_2, r\}$. Here, φ_s is the formula $\langle\text{eat}\rangle\mathbf{t} \vee \langle\cdot\rangle X$, in which every $\langle\text{a}\rangle\psi$ modality is resolved for state s: it is replaced by the formula $\bigvee_{s \xrightarrow{\text{a}} t} \psi_t$. The result is the following BES:

$$\begin{aligned}
\mu X_{l_1} &= X_{l_1} \vee X_{l_2} \vee X_r \\
\mu X_{l_2} &= X_{l_2} \vee X_r \\
\mu X_r &= \mathbf{t} \vee X_{l_1} \vee X_r
\end{aligned}$$

Note that in the equations for l_1 and l_2, the $\langle \text{eat} \rangle \mathbf{t}$ subformula gives rise to a disjunction $\bigvee_{l_i \xrightarrow{\text{eat}}_t} \mathbf{t}_t$ that has a disjunct for each successor state t that is reachable via an 'eat' action. Because l_1 and l_2 do not have such successors, this disjunction disappears in the first two equations. In the equation for X_r, we see this disjunction appear as the constant $\mathbf{t}$. The other disjuncts (in all equations) correspond to the target states of the outgoing transitions in each state.

The solution to this system is an assignment of Booleans b_{l_1}, b_{l_2} and b_r to the variables X_{l_1}, X_{l_2} and X_r such that b_{l_1}, b_{l_2} and b_r are the smallest (w.r.t. the ordering $\mathbf{f} < \mathbf{t}$) values that satisfy the equations. The solutions for X_{l_1}, X_{l_2} and X_r then indicate whether φ holds in l_1, l_2 and r, respectively. These solutions can be found easily by noticing that $X_r = \mathbf{t}$ because $\mathbf{t}$ appears in the disjunction, and then substituting $\mathbf{t}$ for X_r in the first two equations. ■

1.2.2 Parameterized Boolean equation systems

Based on Mader's encoding of the model checking problem, a method was developed to model check processes described by a process algebra with data [Mat98; GM99]. This method uses Boolean equation systems, extended with a notion of data, called *parameterized Boolean equation systems* (PBES). A similar first-order extension of Boolean equation systems called *predicate equation systems* (PES) was introduced by Lin [Lin96] and was also used by Zhang and Cleaveland [ZC05]. Both formalisms achieve the same thing: the model checking problem can be encoded compactly in a system of first-order fixpoint equations, and moreover, this encoding is no longer limited to finite state spaces.

Example. Let our professor be described by the process algebra term in Figure 1.3. Like in the previous example, we will check the formula $\mu X \,.\, \langle \text{eat} \rangle \mathbf{t} \vee \langle \cdot \rangle X$ holds on this process. The following PBES encodes this problem:

$$\begin{aligned} \mu X(m) = {} & (m \approx \text{Hungry} \wedge \mathbf{t}) \vee \\ & (\exists_{m'} \; (m' > m \vee m' \approx m) \wedge X(m')) \vee \\ & (m \approx \text{Hungry} \wedge X(\text{Content})) \end{aligned}$$

This single equation PBES expresses that X is the smallest unary relation (using the correspondence between unary relations and sets: smallest w.r.t. the subset relation) that is equivalent to the expression on the right hand side. To see how the model checking problem is encoded in this PBES, note the following.

A formula of the form $\mu X \,.\, \varphi$ gives rise to an equation for X, where X is equated to φ in which the modal operators are resolved.

Every state of the process that Figure 1.3 represents is identified by an element of the model $\mathfrak{A}$ over which the process algebra term was formulated (to recall, the elements in that model are $\{\text{Content}, \text{Stressed}, \text{Hungry}\}$). In the modal formula, X is a predicate on states. Therefore, in the PBES, X is a unary relation on states.

To resolve the modality in $\langle \text{eat} \rangle \mathbf{t}$ for state m, we have to establish whether the 'eat' action is enabled from that state. From Figure 1.3, it is readily seen that this is the case

when $m \approx$ Hungry. In the target state of that transition (Content), **t** must hold. Note that the **t** in the L_μ formula is a predicate on states that always holds. The translation therefore translates the statement '**t** holds for the state Content' to the Boolean constant **t** (the confusion here is caused by our overloaded use of the symbol **t**, as a constant predicate on states in L_μ, and as a Boolean constant in the BES). This explains the first disjunct: $m \approx \text{Hungry} \wedge \mathbf{t}$.

The second two disjuncts are from the $\langle\cdot\rangle X$ subformula. This modality is resolved by looking at all transitions that are possible, and asserting that X holds for the target state of one of those transitions. From state m, the summation in Figure 1.3 allows outgoing transitions to m', if $m' > m \vee m' \approx m$. The statement that one of these target states satisfies X is captured by the expression $\exists_{m'}\ (m' > m \vee m' \approx m) \wedge X(m')$. What remains is to take into account the 'eat' transition from Figure 1.3 in the same manner, leading to the final disjunct $m \approx \text{Hungry} \wedge X(\text{Content})$.

Solving this system is again not difficult. The relation that X represents must at least contain the element 'Hungry', due to the first disjunct. The second disjunct then requires that the other two elements also are included, because Hungry $>$ Content $\wedge$ X(Hungry) and Hungry $>$ Stressed $\wedge\, X$(Hungry). ■

The PBES formalism has a few advantages. It is rather compact, and its instances can in many cases be solved by algebraic reasoning. This was emphasized in [GW04], where manual algebraic techniques are proposed based on a notion of equivalence of equation systems, and later also for the PES formalism in [ZC05], in which a proof system is developed as the basis for an on the fly model checker. Using symbolic techniques, the Gauß elimination technique for BESs can be extended to work for PBESs [GW05a]. The automated model checkers mentioned here both suffer from the problem that they are not complete: the richness of first-order logic makes it easy to write down undecidable problems, and in practice it is difficult to write software that can handle sufficiently rich decidable fractions without restricting users too much in their freedom to formulate processes in the way they desire (usually, some syntactic restriction is needed to make the input suitable for such tools).

A PBES over a finite structure can be instantiated to an equivalent finite BES, which can be used to solve the PBES [DPW08]. In some cases, even PBESs over infinite structures can be instantiated to finite BESs. Due to the correspondence between Boolean equation systems and *parity games*, a graph based formalism, parity game solvers can also be used to solve BESs. One advantage of this approach is that it only relies on being able to establish equality between states of the system (to avoid calculating the answer for the same state more than once); it does not need to decide equivalence of first-order formulas.

The instantiation technique is especially useful in combination with algebraic techniques. An instantiated PBES may grow very large, and algebraic techniques can help to simplify the PBES prior to the instantiation, thus reducing the size of the resulting BES. Several such techniques have been researched over the past years. The simplest reductions analyse the PBES to find parameters that are unused; eliminating these parameters may substantially reduce the amount of BES variables needed to instantiate the PBES [OWW09]. More advanced techniques try to find conditions under which

parameters become irrelevant (for instance because they will be assigned a new value before they are used again), and use this information to limit the number of values that the parameters of a fixpoint variable can take when instantiating the PBES [KWW13; KWW14]. Abstraction in the PBES setting can also lead to considerable size reduction, subsuming some of the previously mentioned techniques, and can in some cases yield a PBES that can be expanded to a finite BES, while expansion of the original PBES does not terminate [Cra+ed].

1.3 In this thesis

The research described in this thesis covers a few different topics, all related to model checking via fixpoint logic.

Chapter 3 This chapter concerns a communication protocol called FlexRay, that is designed for use in the automotive industry. As this protocol is intended to be used in many in-car applications, including safety critical ones, we would like to establish that the communication protocol itself is not flawed. For reasons explained in more detail in Chapter 3, the initial phase of the protocol, in which communication between nodes is set up, is the part that exhibits the most interesting behaviour. Once communication is set up correctly, nodes in the network will communicate according to the same, predefined schedule, which reserves time slots on the communication channel for each node. This makes it easy to see that once the communication is set up correctly, nodes do not interfere with each other. We therefore focus on the initial phase of the protocol in our case study, trying to answer the following question.

> Does the FlexRay startup sequence always terminate succesfully?

It turns out that it is not a trivial task to answer this question. Apart from some issues with the nature of the FlexRay specifications—it is not at all clear what level of service the protocol is intended to guarantee, making it difficult to establish what a 'succesful' termination of the startup sequence entails—a number of technical issues pop up. The remaining chapters in this thesis address those issues.

The results from this chapter were originally published in the following paper.

[Cra12a] S. Cranen. Model checking the FlexRay startup phase. In *FMICS*, pages 131–145, 2012.

A more detailed, preliminary version of that paper is available as a technical report.

[Cra12b] S. Cranen. Model checking the FlexRay startup phase. Technical Report 12-01, Technische Universiteit Eindhoven, 2012.

Chapter 4 While executing the case study of Chapter 3, its progress was regularly discussed with project partner NXP, a manufacturer of, amongst others, FlexRay chips. During one of these meetings I wanted to explain what exactly I was going to verify on the model I created. The first-order modal μ-calculus formulas from Chapter 3 appeared on my slides, and I spent a long time pointing at various bits, trying to give

an intuition about what the various symbols in the formulas achieve. Proponents of modal μ-calculus emphasize its high expressivity; in this case, I would have settled for a more readable, less expressive logic like CTL*.

Given that the modal μ-calculus is more expressive than CTL*, a natural question is to ask whether any CTL* formula can be efficiently translated into a μ-calculus formula. Already in 1992, Dam had presented a translation from CTL* to the propositional modal μ-calculus [Dam92], but this translation yields formulas that are much larger than the original: in the worst case, their size grows with a factor that is doubly exponential in the size of the input formula. Two years later, Bhat and Cleaveland presented a translation to an equational variant of the propositional modal μ-calculus [BC96], which only caused an exponential blowup of the formula. The mCRL2 toolset, which we used to check the FlexRay models, uses a (non-equational) first-order μ-calculus. We therefore asked ourselves if this first-order aspect resolves the issue of the blowup in the translation. More specifically, given a first-order extension of the propositional modal μ-calculus called FO-L_μ, we asked:

> Is there a succinct translation from CTL* to FO-L_μ?

By 'succinct' we mean that the translation may only yield formulas that are a constant factor larger than the original formula. Using a translation from CTL* to Büchi automata, and exploiting the expressivity of the data language in mCRL2 to encode this translation, we positively answer this question, giving rise to a second question:

> Can the resulting FO-L_μ formula be checked on a system as efficiently as the original formula?

It turns out that model checking the resulting formula indeed has the same complexity as model checking the original, suggesting that the translation can be an easily implementable way of extending μ-calculus model checkers to support CTL*.

The results from this chapter have been published in the following paper.

[CGR11] S. Cranen, J.F. Groote, and M.A. Reniers. A linear translation from CTL* to the first-order modal μ-calculus. *Theoretical Computer Science*, 412:3129–3139, 2011.

An earlier version appeared as a technical report.

[Cra10] S. Cranen, J.F. Groote, and M.A. Reniers. A linear translation from LTL to the first-order modal μ-calculus. Technical Report 10-09, Technische Universiteit Eindhoven, 2010.

Chapter 5 State space explosion is a well-known problem in the model checking community, and it nearly always manifests itself when model checking industrial applications. The FlexRay case study was no exception; we limited ourselves to a minimal network of three nodes, abstracting away from as much detail as we could, and even then the largest state spaces we encountered took days to check. Checking a network with four nodes was infeasible due to large time and memory consumption (an experiment was started on a computation server with 1TB of main memory, but after a month of computing, the experiment ended unfinished, in a server crash.

When model checking via parameterized Boolean equation systems, most of the time is spent in generating and solving *parity games*. These are graph structures that capture the semantics of the parameterized Boolean equation system; the solution of the game is the answer to the encoded verification problem. The classical state space explosion problem (the system model consists of too many states) translates in this setting to the parity games having too many states.

Classical state space reduction techniques provide a generic way to reduce the number of states in the system model, in such a way that certain classes of properties are preserved in the reduced system. For this to be practical, these classes have to be chosen large enough so as not to lose too much expressivity, but they must be small enough to make a decent reduction possible.

In contrast to the classical setting, the structure that suffers from the state space explosion in our case also captures the semantics of the formula that is checked. One could theorize that a reduction on such a structure therefore could, generally speaking, eliminate more system states, because we can effectively reduce with respect to a single formula, rather than a class of formulas.

Keiren and Willemse already showed in 2011 that a parity game reduced using strong bisimilarity (an equivalence relation that is normally used to reduce state spaces of system models) has the same solution as the original parity game [KW11]. The reduced game is smaller, and can therefore be solved in less time than the original. Furthermore, experiments showed that in most cases it is quicker to first minimize the parity game and then solve it, as opposed to solving the original game directly. In other words: the speedup achieved by solving a smaller game was in general larger than the penalty induced by having to calculate the bisimulation quotient.

Strong bisimulation is the finest relation that is commonly used to reduce state spaces, and it preserves a very large class of properties (namely, all properties that can be expressed in the modal μ-calculus). In the setting of parity games, however, we are only interested in preserving the solution: we only need to preserve a single property. Reducing a parity game using a coarser relation will produce a smaller game, and will therefore, hopefully, speed up solving the parity game.

In joint work with the authors of the aforementioned paper we therefore investigated the next natural candidate for parity game reduction: stuttering equivalence. While it is coarser than strong bisimulation, it is still efficiently computable (in $O(n \cdot (n+m))$ time, where m and n are the number of edges and vertices in the parity game, respectively [GV90]). The question we asked was:

> Can stuttering equivalence be used to reduce parity games?

The answer to this question was 'yes', and moreover, experiments indicated that indeed there was a small performance gain compared to reduction using strong bisimilarity. An obvious next question is to ask whether there are efficiently computable relations that are even coarser than stuttering equivalence. One aspect of parity games that is taken into account by stuttering equivalence, is that vertices are owned by different players, and that the solution of the game depends on which player owns which nodes. Stuttering equivalence therefore never relates two vertices that are owned by different players. Sometimes, however, the structure of the graph makes it irrelevant which

player owns a vertex. Essentially, the solution of the game is not so much determined by which player owns a vertex, but by the vertices towards which a player can move. If from a certain vertex the opponent can force the owner of the vertex to move in a way that is beneficial to the opponent, then that vertex might as well have been owned by the opponent. This idea of forced moves makes that vertices with different state labels (different owners, in this case) sometimes are easily seen to be won by the same player. We therefore ask ourselves the following question.

> Can stuttering equivalence on parity games be weakened to take forced moves into account?

Drawing inspiration from *idempotence identifying bisimilarity* [KW11], a relation defined on Boolean equation systems (which are closely related to parity games), we define a relation called *governed stuttering bisimilarity*. This relation is coarser than stuttering, still preserves the solution, and can be computed with a similar algorithm. Although the worst-case time complexity of calculating the quotient is worse than that for stuttering bisimilarity, in practice it is not much slower to compute than stuttering bisimilarity, while it yields better reductions.

The results from this chapter have been published in the following publications.

[CKW11] S. Cranen, J.J.A. Keiren, and T.A.C. Willemse. Stuttering mostly speeds up solving parity games. In M. Bobaru, K. Havelund, G.J. Holzmann, and R. Joshi, editors, *NFM 2011*, volume 6617 of *LNCS*, pages 207–221, 2011.

[CKW12b] S. Cranen, J.J.A. Keiren, and T.A.C. Willemse. A cure for stuttering parity games. In A. Roychoudhury and M. D'Souza, editors, *ICTAC 2012*, volume 7521 of *LNCS*, pages 198–212, 2012.

A technical report appeared that includes the proofs omitted in the latter paper.

[CKW12a] S. Cranen, J.J.A. Keiren, and T.A.C. Willemse. A cure for stuttering parity games. Technical Report 12-05, Technische Universiteit Eindhoven, 2012.

There is some overlap in this chapter with the PhD thesis of Jeroen Keiren, in collaboration with whom most of this work was done:

[Kei13] J.J.A. Keiren. *Advanced reduction techniques for model checking*. PhD thesis, Technische Universiteit Eindhoven, 2013.

Chapters 6 and 7 Fixpoint logics have the advantage of being a very generic mechanism with which a wide variety of problems can be solved. As more and more techniques are developed to perform analysis on the relatively compact fixpoint formulas to reduce their complexity before solving them in the parity game setting [OWW09; OW10; KWW13; KWW14; Cra+ed], the use of fixpoint logic for model checking purposes is becoming more attractive.

An important downside, however, is that the state of the art solvers that are used to compute the solution to fixpoint formulas, and therefore the solution to the encoded verification problem, provide no more than a true/false verdict. In the FlexRay case study, it has occurred that after two days of computing, the solution to a fixpoint

formula turned out to be false, while we expected it to be true. Apparently there was a mistake in our model, but without any additional information, it is very hard to establish what went wrong. Due to the long computation time, a trial-and-error approach in fixing the error in the model is not very practical either.

To make model checking via fixpoint logic easier to use, we would like to have a model checker that provides us with an explanation. In LTL and CTL* model checking with specialized model checking algorithms, such explanations are usually returned by the algorithm in the form of a counterexample (in case the formula does not hold on the model) or a witness (in case the formula holds). In [Tan02], Tan and Cleaveland suggest a witness extraction procedure that involves calculating a proof object they call a *support set*, which is then used to extract counterexaples and witnesses. Our initial question is whether this approach can be used for first-order fixpoint logics, and whether we can extract sensible diagnostics for *all* μ-calculus model checking problems, rather than only for the CTL* fragment.

> Can counterexamples and witnesses be generated for fixpoint formulas that encode arbitrary μ-calculus model checking problems?

Besides model checking problems, also equivalence checking problems (deciding whether two states are related by some fixpoint-definable relation) can be encoded in fixpoint logic. This versatility of fixpoint logic leads to another natural question.

> Can counterexamples and witnesses be generated for fixpoint formulas that encode arbitrary equivalence checking problems?

We partially answered these questions in the setting of *parameterized Boolean equation systems* (PBES), an equational first-order fixpoint logic that underlies the software tools that were used in the FlexRay case study. The results were published in [CLW13]. Because the resulting techniques seemed applicable to a wider range of logics, in particular the much studied *least fixpoint logic* (LFP), this thesis presents a more general result that is applicable to both PBES and LFP. We do this by first defining a logic we call *equational fixpoint logic* (EFL), of which PBES and LFP are normal forms, in the sense that they are syntactic fragments of EFL, and every EFL formula can be translated to a PBES formula and to an LFP formula. This is done in Chapter 6. We then continue in Chapter 7 with the question of finding counterexamples and witnesses for EFL.

Based on Tan and Cleaveland's notion of support set we define proof graphs, which are graph structures that represent the proof for the validity or invalidity of a fixpoint formula, but which exclude the first-order part of the proof. In fact, we show that proof graphs provide an alternative semantics for the fixpoint operator, in the sense that a fixpoint formula is true in a model if and only if there is a proof graph that witnesses this, and likewise, a formula is false in a model if and only if there is a proof graph that witnesses this.

For fixpoint formulas that encode model checking or equivalence checking problems, we show that proof graphs contain enough information to easily extract counterexamples and witnesses. In particular, we show that for LTL model checking, linear counterexamples and witnesses (traces) can be extracted, and that for $\forall$CTL* we can

extract tree-like counterexamples as described in [Cla+02]. Our method of counterexample and witness extraction is generic in the sense that for any fixpoint formula that encodes a comparison between a model $\mathfrak{A}$ and a model $\mathcal{B}$, it extracts those parts of $\mathfrak{A}$ and those parts of $\mathcal{B}$ that are needed to replicate the proof of (in)validity of the formula. We therefore can also generate counterexamples and witnesses for equivalence checking problems. In an example, we show how this is done for stuttering bisimilarity, and how the resulting counterexamples can be interpreted.

Chapters 6 and 7 extend the work presented in the following paper.

[CLW13] S. Cranen, B. Luttik, and T.A.C. Willemse. Proof graphs for parameterized Boolean equation systems. In P.R. D'Argenio and H. Melgratti, editors, *CONCUR 2013*, volume 8052 of *LNCS*, pages 470–484, Springer, 2013.

Chapter 2

Preliminaries

2.1 First-order logic

Throughout this thesis, you will find references to concepts from first-order logic. There are many variations of first-order logic; this section presents the variant used in the thesis. This section is not intended as a logic primer, it is merely included to fix some of the concepts and notation used in the thesis; the reader is presumed to already be familiar with first-order logic.

We start with the syntax, which is defined in terms of a *signature*. Throughout this thesis, we will use $\mathbb{N}$ to denote the set of natural numbers, and $\mathbb{B}$ to denote the set $\{\mathbb{t}, \mathbb{f}\}$ of Booleans.

In the remainder, we assume the existence of an infinite set of variables $\mathcal{V}$ and a disjoint set of logical symbols containing the symbols $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\forall$ and $\exists$.

Definition 2.1. *A signature Σ is a tuple $\langle \mathcal{R}, \mathcal{F}, \mathsf{ar} \rangle$ where:*

- *$\mathcal{R}$ is a set of relation symbols,*
- *$\mathcal{F}$ is a set of function symbols and*
- *$\mathsf{ar} \colon \mathcal{R} \cup \mathcal{F} \to \mathbb{N}$ is a function that assigns an arity to each symbol in the signature.*

Furthermore, $\mathcal{R}, \mathcal{F}, \mathcal{V}$ and the set of logical symbols are pairwise disjoint.

Function symbols with arity 0 are sometimes called *constants*, and 1-ary, 2-ary and 3-ary symbols are often called unary, binary and ternary, respectively.

Signatures give us a collection of syntactic elements to work with. A *structure* adds to this a *domain of discourse* that contains all the semantic objects that we wish to consider, and a mapping between syntax (the signature) and semantics (the domain of discourse).

Definition 2.2. *A structure $\mathfrak{A}$ is a tuple $\langle \Sigma, A, \mathcal{I} \rangle$ where:*

- *Σ is a signature,*
- *A is a set of objects (the* underlying set *or* domain of discourse*) and*
- *$\mathcal{I}$ is the* interpretation function *that maps each relation symbol R in Σ to a subset of $A^{\mathsf{ar}(R)}$, and that maps each function symbol f in Σ to a mapping of type $A^{\mathsf{ar}(f)} \to A$.*

If left unspecified, we will assume that the domains of discourse of the structures $\mathfrak{A}, \mathfrak{B}, \mathfrak{A}'$ etc. are called A, B, A' and so on.

The function symbols in a signature, together with the set of variable names, allow us to define tree-like structures we call *terms*. Given a signature $\Sigma = \langle \mathcal{R}, \mathcal{F}, \mathsf{ar} \rangle$ the set of terms of Σ is defined inductively as follows.

- If $x \in \mathcal{V}$, then x is a term.
- If $t_1, \ldots, t_n$ are terms, and $f \in \mathcal{F}$ with $\mathsf{ar}(f) = n$, then $f\, t_1 \ldots t_n$ is also a term.

In a structure $\mathfrak{A} = \langle \Sigma, A, \mathcal{I} \rangle$, a term t of Σ is associated with a unique element of A, if the term does not contain variables. If it does contain variables, then the element of A that is represented by t depends on the value that these variables take. We introduce the notion of *environment*, which gives a valuation to variables. Then, in the context of $\mathfrak{A}$ and an environment θ, t can be associated with a unique element from A.

Definition 2.3. *In the context of a structure $\mathfrak{A}$, an* environment *(usually called θ, η, θ', etc.) is a mapping of type $\mathcal{V} \to A$.*

It is often useful to construct new environments out of existing ones by updating the mapped value for a single variable. If $x \in \mathcal{V}$ and $a \in A$, then $\theta[x \mapsto a]$ denotes the environment that maps $y \in \mathcal{V}$ to a if $y = x$, or to $\theta(y)$ otherwise. Given environments θ and η, if $\theta(y) = \eta(y)$ for all y in some set of variables V, then θ and η are said to *agree* on the variables in V, denoted $\theta \equiv_V \eta$.

The interpretation $t^{\mathfrak{A},\theta}$ of a term t in the context of a structure $\mathfrak{A} = \langle \Sigma, A, \mathcal{I} \rangle$ and environment θ is defined as follows:

- if $t \in \mathcal{V}$, then $t^{\mathfrak{A},\theta} = \theta(t)$;
- if $t = f\, t_0 \ldots t_n$, then $t^{\mathfrak{A},\theta} = \mathcal{I}(f)(t_0^{\mathfrak{A},\theta}, \ldots, t_n^{\mathfrak{A},\theta})$.

In this manner, syntactic structures (terms) are associated with semantic objects (elements from A). Formulas of first-order logic add operators with which we can reason about these objects.

Definition 2.4. *Given a signature $\Sigma = \langle \mathcal{R}, \mathcal{F}, \mathsf{ar} \rangle$, the set of first-order formulas over Σ is defined as follows.*

- *If $t_1, \ldots, t_n$ are terms, and $R \in \mathcal{R}$ with $\mathsf{ar}(R) = n$, then $R t_0 \ldots t_n$ is a formula.*
- *If φ is a formula, then $\neg\varphi$ is a formula.*
- *If φ and ψ are formulas, then $\varphi \vee \psi$ is a formula.*
- *If φ is a formula and $x \in \mathcal{V}$, then $\exists_x\, \varphi$ is a formula.*

Formulas obtained by the first rule are called *atomic* formulas. Throughout this thesis, we will be using the following commonly used abbreviations:

$$\varphi \wedge \psi \triangleq \neg(\neg\varphi \vee \neg\psi)$$
$$\varphi \Rightarrow \psi \triangleq \neg\varphi \vee \psi$$
$$\forall_x\, \varphi \triangleq \neg\exists_x\, \neg\varphi$$

The set $\mathsf{sf}(\varphi)$ of *subformulas* of a first-order formula φ is defined inductively as follows:

$$\mathsf{sf}(\varphi) = \begin{cases} \{\varphi\}, & \text{if } \varphi \text{ is atomic} \\ \{\varphi\} \cup \mathsf{sf}(\psi_1) \cup \mathsf{sf}(\psi_2) & \text{if } \varphi \text{ is } \psi_1 \oplus \psi_2, \text{ with } \oplus \in \{\wedge, \vee, \Rightarrow\} \\ \{\varphi\} \cup \mathsf{sf}(\psi) & \text{if } \varphi \text{ is } \forall_x \psi, \exists_x \psi \text{ or } \neg\psi \end{cases}$$

We often write $\psi \sqsubseteq \varphi$ instead of $\psi \in \mathsf{sf}(\varphi)$. We use $\psi \sqsubset \varphi$ to denote $\psi \sqsubseteq \varphi \wedge \psi \neq \varphi$.

We say that a first-order formula φ *holds* in a model $\mathfrak{A} = \langle \Sigma, A, \mathcal{I} \rangle$ and environment θ, denoted $\mathfrak{A}, \theta \models \varphi$, according to the following recursive definition:

- $\mathfrak{A}, \theta \models Rt_0 \dots t_n$ if and only if $\langle t_0^{\mathfrak{A},\theta}, \dots, t_n^{\mathfrak{A},\theta} \rangle \in \mathcal{I}(R)$.
- $\mathfrak{A}, \theta \models \neg\varphi$ if and only if not $\mathfrak{A}, \theta \models \varphi$.
- $\mathfrak{A}, \theta \models \varphi \vee \psi$ if and only if $\mathfrak{A}, \theta \models \varphi$ or $\mathfrak{A}, \theta \models \psi$.
- $\mathfrak{A}, \theta \models \exists_x \varphi$ if and only if there is some $a \in A$ such that $\mathfrak{A}, \theta[x \mapsto a] \models \varphi$.

2.2 Automata

Although nowhere in this thesis we are primarily concerned with automata, they do play a central role in model checking. As such, they also are essential in some of the proofs, and in our illustrations of how model checking techniques are applied. We therefore give some very minimal definitions of commonly used concepts.

Kripke structures A *Kripke structure* is a tuple $\mathfrak{A} = \langle A, AP, \rightarrow, \ell, a_0 \rangle$ where A is a set of states, AP is a set of atomic predicates that label the states of the automaton, $\rightarrow \subseteq A \times A$ is a binary transition relation, $\ell : A \to 2^{AP}$ a state labelling function, and $a_0 \in A$ the initial state. The initial state may be omitted if is not important in the context in which the Kripke structure is used. If for every $a \in A$ there is some $a' \in A$ such that $a \rightarrow a'$, then the transition relation is said to be *total*.

Labelled transition systems A *labelled transition system* (LTS) is a tuple $\mathfrak{A} = \langle A, L, \rightarrow, a_0 \rangle$ where A is a set of states, L is a set of transition labels, $\rightarrow \subseteq A \times L \times A$ is a ternary transition relation, and a_0 the initial state. Again, the initial state may be omitted.

Büchi automata A *Büchi automaton* is a tuple $\mathcal{B} = \langle B, L, \rightarrow, F, B_0 \rangle$ where B is a set of states, L is a set of transition labels, $\rightarrow \subseteq B \times L \times B$ is a ternary transition relation, $F \subseteq B$ is a set of accepting states and $B_0 \subseteq B$ is the set of initial states of the automaton. We assume the transition relation is total in the sense that for every state $b \in B$ there are at least one $x \in L$ and $b' \in B$ such that $(b, x, b') \in \rightarrow$ (which we will usually denote $b \xrightarrow{x} b'$). We write $\mathcal{B}(b_0)$ to denote a Büchi automaton $\mathcal{B}$ with initial state b_0 (*i.e.*, $B_0 = \{b_0\}$).

Although there is a lot more to be said about Büchi automata, we will use only this minimal definition in this thesis. The use of this structure will become apparent in Section 2.4, where we show how Büchi automata can be used to specify behaviour.

Paths A *path* through a Kripke structure $\mathfrak{A}$ is a sequence $a_0 \dots a_n$ such that $a_i \in A$ for all $i \leq n$ and $a_i \to a_{i+1}$ for all $i < n$. This notion is extended to infinite paths in the obvious manner, and to labelled transition systems and Büchi automata by demanding that $\exists_{l \in L}\ a_i \xrightarrow{l} a_{i+1}$ for all $i < n$. If p is a path, then p_0 denotes the first state on the path, p_1 the second, and so on. Path concatenation is denoted by juxtaposition: if p is a finite path of length $n+1$, q is a path, and $p_n \to q_0$, then pq is again a path. Finally, we introduce notation for path suffixes: p^i denotes path p without its i first states.

2.3 Lattices and fixpoints

In the chapters that follow, we use logics and modal logics with fixpoint operators. Here we present very briefly the basic notions that underly these logics. We assume the reader is more or less familiar with these concepts, and this section therefore serves merely as a reminder.

Partially ordered sets. A binary relation R is a *partial order* on a set A if and only if it is reflexive, antisymmetric and transitive. If A is a set and $\sqsubseteq$ is a partial order on A, then $\langle A, \sqsubseteq \rangle$ is called a *partially ordered set*. Given a set $B \subseteq A$, if $a \in A$ is such that $b \sqsubseteq a$ for all $b \in B$, then a is an *upper bound* for B in A. Likewise, if $a \sqsubseteq b$ for all $b \in B$, then a is a *lower bound* for B in A. If a is an upper bound of B, and for all other upper bounds a' of B, $a \sqsubseteq a'$, then a is called the least upper bound, or *supremum*, of B, and is denoted $\bigsqcup B$. Dually, if a is a lower bound of B and all lower bounds a' of B are such that $a' \sqsubseteq a$, then a is the greatest lower bound, or *infimum*, of B, and is denoted $\bigsqcap B$. Note that these least and greatest upper bounds need not exist.

Lattices. If every $\{a, b\} \subseteq A$ has a supremum and an infimum, then $\langle A, \sqsubseteq \rangle$ is called a *lattice*, and we define a *meet* operator $\sqcap : A \times A \to A$ and a *join* operator $\sqcup : A \times A \to A$ such that $a \sqcap b$ yields the infimum of $\{a, b\}$, and $a \sqcup b$ yields the supremum of $\{a, b\}$. If every $B \subseteq A$ has a supremum and an infimum, then $\langle A, \sqsubseteq \rangle$ is called a *complete lattice*.

Fixpoints. Let $\langle A, \sqsubseteq \rangle$ be a complete lattice, and let $F : A \to A$ be a mapping that maps A onto itself. Its *fixpoints* are defined to be the elements of A for which F returns the same element again, *i.e.*, they are defined as the set $\mathsf{fp}(F) = \{a \in A \mid F(a) = a\}$.

F is called *monotone* if $a \sqsubseteq b$ implies $F(a) \sqsubseteq F(b)$ for all $a, b \in A$. By a result of Tarski [Tar55], $\langle \mathsf{fp}(F), \sqsubseteq \rangle$ is again a non-empty complete lattice if F is monotone. We can therefore define a *least fixpoint* $\mathbf{lfp}\, F = \bigsqcap \mathsf{fp}(F)$ and a *greatest fixpoint* $\mathbf{gfp}\, F = \bigsqcup \mathsf{fp}(F)$. Note that the least and greatest fixpoint are again elements of $\mathsf{fp}(F)$, because we can take the infimum and supremum in $\langle \mathsf{fp}(F), \sqsubseteq \rangle$ instead of in $\langle A, \sqsubseteq \rangle$. The least and greatest fixpoint can be computed using the following (transfinite) recursive definitions [Mos74], in which α is an ordinal.

$$\mathbf{lfp}^\alpha F = F(\bigsqcup_{\beta<\alpha} \mathbf{lfp}^\beta F) \qquad \mathbf{gfp}^\alpha F = F(\bigsqcap_{\beta<\alpha} \mathbf{gfp}^\beta F)$$

Note that in particular, $\bigsqcup_{\beta<0} \mathbf{lfp}^\beta F = \bigsqcup \emptyset = \bigsqcap A$ and $\bigsqcap_{\beta<0} \mathbf{gfp}^\beta F = \bigsqcap \emptyset = \bigsqcup A$. For large enough α, $\mathbf{lfp}\, F = \mathbf{lfp}^\alpha F$ and $\mathbf{gfp}\, F = \mathbf{gfp}^\alpha F$.

2.4 Temporal logics

When reasoning about the behaviour of systems, it is often convenient to assume that the behaviour of the system under scrutiny is described by a Kripke structure (or labelled transition system, in which not the states, but the transitions are labelled), and to formulate properties of these systems in a logic that is interpreted in the context of a particular state in that particular system. For instance, if someone says that 'all roads lead to Rome', we will most likely interpret this as 'from your current location, all paved roads (as opposed to railroads and hiking trails) lead to Rome'. To make this context explicit, we introduce common notation for all temporal logics. If $\mathfrak{A}$ represents a system (for instance, the European road network), a is a state in that system (say, a state labelled with 'Eindhoven'), and φ is a formula in some temporal logic ('all roads lead to Rome'), then we write

$$\mathfrak{A}, a \models \varphi$$

to express that φ holds for state a in $\mathfrak{A}$. In our example, we have now made explicit that we are only considering European roads, and that our starting point is Eindhoven.

To make formal reasoning about Kripke structures possible, a plethora of temporal logics is available, with a varying degree of expressiveness and readability. We present a few of the more common ones that will be used in subsequent chapters.

Büchi automata If the labels of a Büchi automaton $\mathcal{B} = \langle B, L, \to, F, B_0 \rangle$ are formulas of a temporal logic, then we can use it to describe the behaviour of a Kripke structure $\mathfrak{A} = \langle A, AP, \to, \ell \rangle$.

If $b_0 \in B_0$, then an infinite path $b_0 \xrightarrow{f_0} b_1 \xrightarrow{f_1} \dots$ is called a *run* on a state $s_0 \in A$ if there is a path $s_0 \to s_1 \to \dots$ in $\mathfrak{A}$ such that $\mathfrak{A}, s_i \models f_i$ for all i. The run is *accepting* if there are infinitely many i such that $b_i \in F$. A simple (and common) choice for such a temporal logic is to to choose $L = 2^{AP}$: one can interpret subsets of AP as temporal formulas, by defining $\mathfrak{A}, a \models P$ (for $P \subseteq AP$) if and only if $P = \ell(a)$. In the case of the ECTL* logic, described later in this section, the transitions of Büchi automata are labelled with more complicated formulas.

A *generalized Büchi automaton* is a Büchi automaton that has a set of acceptance sets $\mathcal{F}$ rather than a single acceptance set F. A run $b_0 \xrightarrow{f_0} b_1 \xrightarrow{f_1} \dots$ of such an automaton is accepting if and only if it passes through a state in every acceptance set infinitely often, *i.e.*, for all $F \in \mathcal{F}$, $\{i \mid b_i \in F\}$ is infinitely large.

CTL*, CTL and LTL We use the definition of CTL* and LTL as given in [CGP99]. We first describe the syntax of so-called *state formulas* and *path formulas*. Given some non-empty set AP of atomic predicates, the syntax of a state formula is given by the following grammar:

$$f ::= a \mid \neg f \mid f \vee f \mid \mathsf{A} g$$

In the above, $a \in AP$ and g is a path formula generated by the following grammar, in which f is again a state formula:

$$g ::= f \mid \neg g \mid g \vee g \mid \mathsf{X} g \mid g \,\mathsf{U}\, g$$

CTL formulas* are state formulas as described by the top grammar. Subformulas are defined in the same way as is done for first-order logic. The *size* of a CTL* formula f, denoted $|f|$, is the number of subformulas it contains. The following derived constants and operators are in common use.

$$\mathbf{t} \triangleq a \vee \neg a \qquad \mathsf{F} f \triangleq \mathbf{t}\ \mathsf{U}\ f \qquad \mathsf{E} f \triangleq \neg \mathsf{A} \neg f$$
$$\mathbf{f} \triangleq \neg \mathbf{t} \qquad \mathsf{G} f \triangleq \neg \mathsf{F} \neg f$$
$$f \wedge g \triangleq \neg(\neg f \vee \neg g) \qquad f\ \mathsf{R}\ g \triangleq \neg(\neg f\ \mathsf{U}\ \neg g)$$

Computation tree logic (CTL) is a subset of CTL*. It can be characterized using the derived operators, using the following grammar:

$$f ::= a \mid \neg f \mid f \vee f \mid \mathsf{AX} f \mid \mathsf{AG} f \mid \mathsf{A} f\ \mathsf{U}\ f$$

Linear temporal logic (LTL) is also subset of CTL*. *LTL formulas* are CTL* formulas of the form $\mathsf{A} g$, where the A and E operator do not occur in g, *i.e.*, the formulas defined by the following grammar:

$$f ::= \mathsf{A} g$$
$$g ::= a \mid \neg g \mid g \vee g \mid \mathsf{X} g \mid g\ \mathsf{U}\ g$$

The semantics of CTL* formulas (and LTL formulas) is defined on states in a Kripke structure $\mathfrak{A} = \langle S, \rightarrow, AP, \ell \rangle$, in which $\rightarrow$ is total. We will write $\mathfrak{A}, s \models f$ if CTL* formula f holds in state $s \in S$. If g is a path formula, then we also define $p \models g$ for every path p in $\mathfrak{A}$, meaning that g holds along p. If a is an atomic predicate, f and f' are state formulas, and g is a path formula, then the interpretation for state formulas is as follows:

$$\begin{array}{lll} \mathfrak{A}, s \models a & \text{iff} & a \in \ell(s) \\ \mathfrak{A}, s \models \neg f & \text{iff} & \text{not } \mathfrak{A}, s \models f \\ \mathfrak{A}, s \models f \vee f' & \text{iff} & \mathfrak{A}, s \models f \text{ or } \mathfrak{A}, s \models f' \\ \mathfrak{A}, s \models \mathsf{A} g & \text{iff} & p \models g \text{ for all paths } p \text{ starting in } s \end{array}$$

The interpretation for path formulas is described by the following rules, where f is a state formula, and g and g' are path formulas.

$$\begin{array}{lll} \mathfrak{A}, p \models f & \text{iff} & \mathfrak{A}, p_0 \models f \\ \mathfrak{A}, p \models \neg g & \text{iff} & \text{not } \mathfrak{A}, p \models g \\ \mathfrak{A}, p \models g \vee g' & \text{iff} & \mathfrak{A}, p \models g \text{ or } \mathfrak{A}, p \models g' \\ \mathfrak{A}, p \models \mathsf{X} g & \text{iff} & \mathfrak{A}, p^1 \models g \\ \mathfrak{A}, p \models g\ \mathsf{U}\ g' & \text{iff} & \mathfrak{A}, p^k \models g' \text{ for some } k \text{ and } \mathfrak{A}, p^j \models g \text{ for all } 0 \leq j < k. \end{array}$$

If it is clear from the context which Kripke structure should be used to evaluate a CTL* formula, we will omit it and write $s \models f$ rather than $\mathfrak{A}, s \models f$.

ECTL* CTL* can be extended to a slightly more expressive formalism by replacing path formulas by Büchi automata. Instead of $\mathsf{E} g$, with g a path formula, we define $\mathsf{E}\mathcal{B}$

to be a valid formula if $\mathcal{B}$ is a Büchi automaton labelled with ECTL* formulas, with the following interpretation:

$$\mathfrak{A}, s \models \mathsf{E}\mathcal{B} \quad \text{iff} \quad \text{there is an accepting run of } \mathcal{B} \text{ on } s$$

The subformula relation is defined as for CTL*, with the additional rule that a formula φ is a subformula of $\mathsf{E}\mathcal{B}$ if it occurs in the labels of $\mathcal{B}$. The subformula relation is still well-founded: it cannot be the case that an ECTL* formula φ contains a Büchi automaton that uses φ as a transition label.

∃CTL*, ∀CTL* and ∃ECTL* Negation is quite a powerful operator in CTL*. If its use is not restricted, both A and the derived operator E can be used. Using the A operator, one can create formulas that can only be satisfied by states with more than one outgoing transition. Formulas that do not use A, and that only use E in the scope of an even number of negations only express the existence of a path that satisfies a certain property; if such a formula holds in a state in some structure, then that structure therefore only needs to have states with at most one outgoing transition. ∃CTL* is the subset of CTL* that contains only such formulas. Obviously, it is less expressive than CTL* itself. Its dual is ∀CTL*, the subset of CTL* in which only formulas are allowed in which E occurs in the scope of an odd number of negations (*i.e.*, only the A operator is used). Its distinguishing power is equal to that of ∃CTL*, because the negation of an ∀CTL* formula an ∃CTL* formula. Similarly, ∃ECTL* is ECTL* restricted to formulas in which E only occurs in the scope of an even number of negations.

Chapter 3

FlexRay

In the year 2000, a consortium was established with the goal to design a new, time-triggered communication protocol for use in the automotive industry that would outperform CAN and TTP in both speed and reliability. At the end of 2009, the consortium was disbanded, leaving a final version of a time-triggered protocol called FlexRay. The final protocol definition, a 336 page document, became available in 2011, and is currently being transformed into an ISO standard.

Already in 2006, the first commercially available cars were equipped with FlexRay networks, enabling new algorithms for vehicle control because of its higher bandwidth.

Since FlexRay will be the basis for communication in many vehicles to come, we would like to establish that the protocol is correct, *i.e.*, that a system that is implemented according to the latest specification behaves predictably and shows no undesirable behaviour. We base our notion of correctness on the requirements document [Fle05c] that was composed by the FlexRay consortium.

The FlexRay protocol requires that nodes are synchronised in order to communicate. The procedure of starting up a FlexRay network therefore is of particular interest, because it involves a distributed algorithm that should reach such a synchronised state in a reasonable amount of time. This procedure should work for any given startup scenario, and should be to some extent fault-tolerant.

We choose to formalise the FlexRay protocol by means of a model written in the mCRL2 specification language [GM14]. This language has extensive support for the use of data in models, and allows us to create a concise model that stays close to the specification. Fault tolerence is checked by explicitly modelling a number of faults that should be allowed to occur, according to the requirements document, and requiring the same properties to hold as for fault-free scenarios.

We have created a model that captures the details of that part of a node that orchestrates its operation while the node is starting up. Analysis of the model reveals a scenario in which the network does not correctly deal with a single failing node. To our knowledge, this scenario was not documented before.

The structure of this chapter is as follows. We start by giving a brief overview of the FlexRay protocol, and describe how the startup procedure works. We then briefly discuss the requirements that the startup procedure should satisfy. After that, we show how we arrived at our model: we discuss the abstractions that we applied and we demonstrate how mCRL2 fragments are related to the protocol specification.

We then describe the method used to verify the presented model, and subsequently present the verification results. We discuss related research, before wrapping up with some conclusions and suggestions for future work.

3.1 FlexRay

We base our analysis on the 3.0.1 version of the protocol [Fle09]. A FlexRay network consists of a number of nodes that are each connected to one or two communication mediums, called *channels*. The two channels may be used to create redundancy, or to connect to two different sets of nodes. A channel may itself consist of a number of infrastructure components, but can be as simple as a pair of copper wires. FlexRay is a *time-triggered* protocol, which is to say that a clock that is synchronised across the network dictates which node has the right to write messages to a communication medium. This in contrast to for instance CAN, which is *event-triggered*: whenever the event occurs that a node wishes to send a message, the CAN protocol decides at that moment whether that node is allowed to do so, based on some priority scheme.

A *schedule* records which node has access to the medium at what time. The schedule is an access scheme that defines for a finite period (which is called a *cycle* in the protocol specification) the allocation of bandwidth to network nodes. Indefinite repetition of the schedule allows any node to decide at any moment in time which node is allowed to write to the medium.

This scheme is only strictly followed in that part of the schedule that is called the *static segment* (although we should note that there are features that allow a user to slightly deviate from the scheme). The FlexRay requirements document [Fle05c] states that the aim of the protocol is to provide both 'deterministic' communication and 'on-demand' communication. To this end, part of the schedule may be left undecided; this part is called the *dynamic segment*. A priority based selection scheme is implemented on top of a time-triggered access scheme to dynamically allocate bandwidth to nodes that need it in the dynamic segment. We will not consider the use of a dynamic segment, and will therefore not describe the details of its implementation.

In the static segment of the schedule, time is divided into *slots*, and each slot is allocated to a network node that is allowed to send data in that slot. Data is sent in structured packets called *frames*. Some frames play a special role in the protocol: they can be marked as *sync frame* or as *startup frame*. A sync frame is a frame that is used by all nodes in the network to adjust the local view on the global clock. This is done using a variant of the clock synchronisation algorithm of Lundelius and Lynch [LL84]. A startup frame is a sync frame that is allowed to be sent during startup of the network, which we describe in more detail in the next section.

To communicate over a FlexRay network, the network must first be brought to a state in which it is synchronised, and aware of the communication schedule and the current position in that schedule. Firstly, the network is woken up; a special symbol is sent over the channel that causes nodes to switch to an active mode.

When woken up, a distinction is made between *coldstart* nodes and non-coldstart nodes. Coldstart nodes will attempt to initiate communication on a silent bus when

woken up, where non-coldstart nodes will wait until they detect ongoing communication and then integrate. For each node (coldstart or non-coldstart) we call the period in which it is awake and trying to establish communication or trying to integrate into existing communication the *startup phase* of that node.

During this startup phase, clock synchronisation is initialised and some consistency checks are done. In the next section we describe in more detail how this is done. When startup is completed, all nodes in the network should have reached agreement on what the global time is. Communication then proceeds according to the aforementioned schedule. We note here that the term *cycle* for one execution of the schedule is somewhat confusing here, because nodes keep track of a cycle counter, which is reset periodically (with a period smaller than 65 cycles), and which must correspond to the other nodes' cycle counters. Hence the behaviour of a node is only truly cyclical in this period of cycles. The structure of a communication cycle is however always the same, and we will therefore use the term communication schedule to refer to this structure.

3.1.1 The startup phase

We study the behaviour of FlexRay networks during the startup phase of the protocol. In this phase, a distinction is made between *coldstart* nodes and regular nodes. Coldstart nodes are the only nodes that are allowed to start sending data on the bus if there is no activity on the bus yet. A FlexRay network can be configured to have any number of coldstart nodes, with a minimum of three (or two if the network consists of only two nodes).

When the network is awake and a coldstart node is requested by a client application to start communication, the node will start by listening to the bus for a duration that is equal to the length of two schedule cycles, to detect ongoing traffic. Note that we cannot yet speak of real cycles, because there is no shared time base. We will however not always be this strict and will sometimes say 'cycle' when we mean 'the local estimation of the duration of a cycle', and we will similarly speak of 'bits' and 'slots'.

If no communication is detected during this first listening phase, the coldstart node will decide to send a *collision avoidance symbol*, or CAS. The CAS is a signal to the other nodes to indicate that some node is trying to initiate communication: if another (coldstart) node sees a CAS, it will wait for another two cycles, expecting to hear more

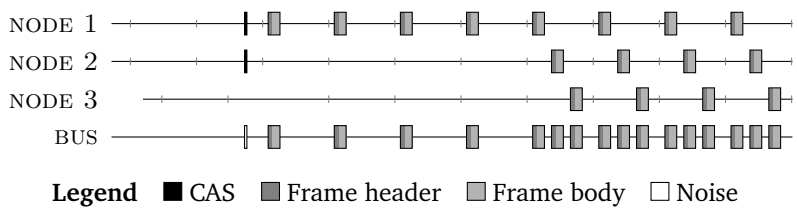


Figure 3.1: Three nodes starting up. The BUS line is the combined signal of the three nodes.

from the node that sent the CAS.

It may be the case that two or more nodes simultaneously decide to send a CAS. To resolve this situation, a leader is selected based on the schedule. The first node scheduled to send a startup frame will become the leader. This is implemented as follows: as soon as a coldstart node has started sending the CAS, it moves to a collision resolution stage. During four cycles, it sends its startup frames according to the schedule, but if it sees a frame header on the bus, it aborts its startup attempt and returns to the listening stage. The result is that the first to send a frame after sending the CAS is the node that will initiate communication. Because the CAS was sent simultaneously, the clocks of the competing nodes are synchronized (within a certain error margin), which guarantees that the frames they send do not overlap and are therefore readable by the receiving nodes.

After four cycles, the initiating node checks during two consecutive cycles whether it sees a frame from another node. If this is the case, then the startup phase ends for this node. If only one frame is seen, then the initiating node considers the startup attempt failed, and it goes back to the listening stage. If no frames were seen, then apparently no other nodes followed the initiative, so it is assumed that there simply were no other nodes ready to start communication yet. The initiating node waits for one cycle, and then resends the CAS and repeats the procedure.

If, in the listening phase, ongoing communication was detected, then a node will attempt to join in by first waiting for two consecutive frame headers from the same node to synchronise the clock with. It then checks during two cycles that either the node it synchronised on is still sending frames, or that at least two nodes are sending frames each cycle. If this is the case, it enters normal operation, if it is not, then it aborts its startup attempt and returns to the listening stage.

An example run of a fault-free startup is shown in Figure 3.1, where nodes 1 and 2 start simultaneously, and node 3 joins in a little later.

3.1.2 Requirements

The FlexRay protocol defines a startup procedure by specifying the local behaviour of a FlexRay node. It is therefore not immediately clear how the startup phase can be defined at the level of a FlexRay network, and what this startup phase should ensure. We take inspiration from the FlexRay requirements document [Fle05c] in which we find requirements on a more global level. Regarding startup of networks, the requirements document specifies a number of faults that the system must be able to deal with when starting up:

> The wake-up and start-up of the ‘communication system’, the integration of ‘nodes’ powered on later and the reintegration of failed ‘nodes’ shall be fault-tolerant against: the temporary/permanent failure of one or more ‘communication modules’ (down to one module sending in the static part for mixed or pure static configurations), the temporary/permanent failure of one or more communication ‘channel(s)’ in a redundant configuration, and the loss of one or more ‘frames’. — [Fle05c], *p. 84*

The fault-tolerance mentioned here is interpreted differently for different types of faults. This classification is given a bit further in the same document:

> To say that "a cluster is able to start up in the presence of a fault" has a meaning that depends on the type of fault.
>
> – Fault class 1: The fault is associated to a channel or a star: All nodes which are intended to participate in communication and are connected to the other channel reach a state where they communicate to one another as scheduled. They reach this state within a defined maximum time.
> – Fault class 2: The fault is associated to a node: *All* fault-free nodes which are intended to participate in communication reach a state where they communicate to one another as scheduled. They reach this state within a defined maximum time.
> – Fault class 3: Transient fault: All nodes which are intended to participate in communication reach a state where they communicate to one another as scheduled. They reach this state within a defined maximum time (For a value see the requirements specification).

The requirement then gives some faults that FlexRay networks should be robust against. In some cases, the use of both FlexRay channels is required to ensure correct operation. The scope of this thesis only covers single-channel networks, and therefore all faults in class 1 are considered fatal (if the only used channel is not working, then no communication can take place).

The class 2 and 3 faults within the scope of our investigations are the following (we again quote, and numbering corresponds to the numbering in [Fle05c]).

> 3 A single I/O signal/pin connecting two of the components communication module, bus guardian or bus driver has become disconnected inside one of the nodes in the cluster. e.g. *[sic]* bus driver's Transmit Enable input disconnected.
> 3b signal wave form degraded on one channel [only has to work in certain circumstances].
> 4 A single arbitrary node in the cluster is not communicating on all attached channels for whatever reason. It doesn't transmit anything, or it starts to transmit somewhat later than the other nodes.
> 5 A single clock oscillator in the cluster erroneously runs at the wrong frequency. *[...]*
> 6 A node (e.g. coldstart node) cannot receive any communication element on all its attached channels.
> 8 A star cannot receive any communication element of a certain branch [only has to work in certain circumstances].
> 9 A periodic reset event is present in a node.
> 12 One single bit of a communication element on one channel flips.

13 For a given time of less than one frame length, all present channels are forced to an arbitrary pattern.
14 A bus driver in a node cannot receive anything.
15 A bus driver in a node cannot transmit anything.
17 A coldstart node sends sporadically CASs. After occurrence, the fault does not manifest itself for at least 10 communication cycles.
18 No node is operational except of 2 fault free nodes and these two nodes are assigned to perform startup ("coldstart nodes"). (Req ID 326)

Our aim is to verify that in the listed cases, the correctly functioning nodes in a FlexRay network will indeed 'communicate to one another as scheduled'. This leaves quite some room for interpretation. For instance, in scenario *17*, a faulty node keeps disturbing the bus every once in a while. While it is doing so, the other nodes can obviously be prevented from communicating according to schedule. We therefore give our own definition of what it means to start up successfully, based on the requirements above:

> *The network has successfully started if the startup procedure has terminated successfully, and during one cycle in which none of the non-faulty nodes are executing their startup procedure, every frame that is sent by a non-faulty node is received by all other non-faulty nodes, unless a faulty node or transient fault prevents reception.*

The verification carried out in this chapter is based on this definition of correct startup behaviour.

3.2 Model

The protocol specification defines the FlexRay protocol in terms of SDL (Specification and Description Language, [ITU99]) diagrams and accompanying text, which, as stated in the introduction, is intended to provide "a reasonably unambiguous description of the mechanisms and their interactions" involved in the protocol. Furthermore, "an actual implementation should have the same behavior as the SDL description, but it need not have the same underlying structure or mechanisms". We are interested in this behaviour, and therefore construct a model that we can directly relate to the SDL description. Our aim is to create a model that is as close to the SDL description as possible, so that the behaviour of the model can be straightforwardly interpreted. We illustrate with snippets of mCRL2 code how this is done in Section 3.2.3.

A single FlexRay node consists of 12 concurrently running, interacting processes, called *controller host interface* (CHI), *protocol operation control* (POC), *macrotick generation* (MTG), *clock synchronization startup* (CSP-A, CSP-B), *clock synchronization processing* (CSP), *media access control* (MAC-A, MAC-B), *frame and symbol processing* (FSP-A, FSP-B) and *coding/decoding* (CODEC-A, CODEC-B). Some of these processes are dedicated to serve a single channel (A or B), and are hence duplicated. These processes are shown in Figure 3.2. The discrete behaviour during the startup phase is

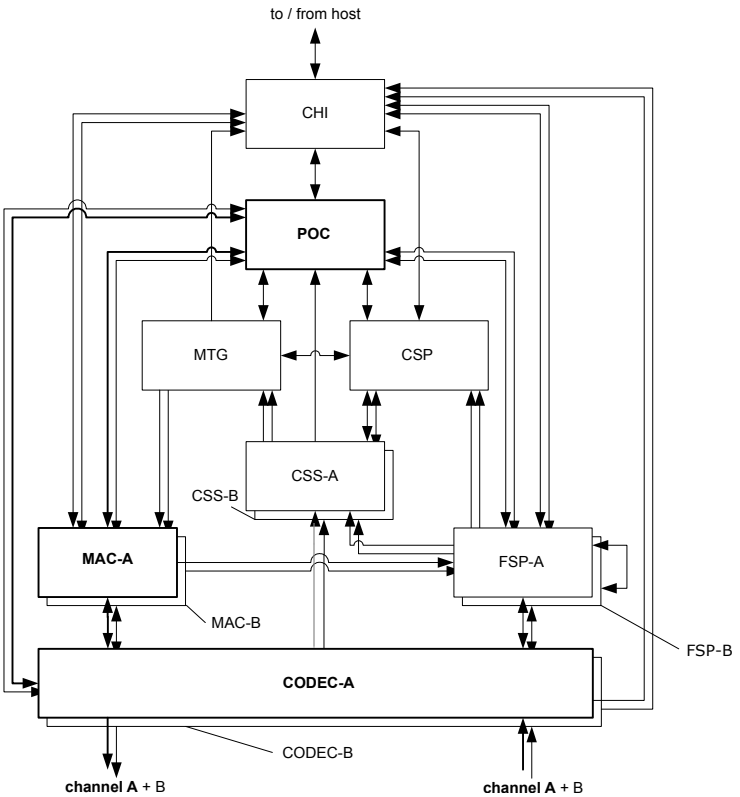


Figure 3.2: The context of the POC process (based on Figure 2–2 in [Fle09]). Arrows indicate (directed) communication between components. The components that we model are printed in boldface.

governed by the POC process. We therefore aim to model this process in detail, while abstracting away as much as possible from the other processes. The parts of the nodes that we do model are printed in boldface in Figure 3.2.

3.2.1 Abstractions

We are, as stated before, interested in the discrete behaviour of a FlexRay network during the startup phase. This means we do not want to take into account timing aspects such as propagation delays, clock speed and so on. Like time, data also influences the behaviour of the protocol, but again we desire a limited level of detail: only those features of the data structures used in FlexRay which influence the decisions made in the POC process are modelled. We describe the abstractions we use in more detail below.

Environment During startup, the CHI process is used to feed back information to the client application. It may also influence the POC process by enabling the so-called

coldstart inhibit mode, which causes a coldstart node to not actively start communication by sending a CAS. We assume none of the nodes are ever put in this mode, and hence leave out the CHI altogether.

The FSP process performs validity checks on the symbols that are decoded by the coding / decoding process. The only way in which FSP influences the POC process is by emitting a *fatal protocol error* signal, which happens when FSP detects that the node that it is part of is sending across a boundary in the schedule. We assume that CODEC and MAC processes function correctly, which should prevent this error from occurring. We therefore do not model FSP.

Communication and calculation Component interactions are modelled in the SDL description as signals being sent from one component to another. We make the assumption that messages are received the moment they are sent. This eliminates the need for (possibly unbounded) queues to model interactions between processes. Similarly, we assume that calculations take no time to complete.

Time In [Mal08; MN10], Malinský and Novák present a model of the startup phase of a FlexRay network. Their approach is to make no assumption about the correct working of the clock synchronisation protocol, and they subsequently show that several parameters of their model can be adjusted in such a way that the network model does not properly start up. Although our choice of modelling language does allow us to model timed behaviour, our chosen approach to stay as close to the specification as possible severely restricts the timing aspects we can model. This is mainly due to the fact that timing differences give rise to an enormous state space blowup, and it is often infeasible to prove that the resulting extra states are confluent, bisimilar or otherwise related before expanding the entire state space. Due to this blowup, Malinský needs a substantial amount of manual reasoning to create a model that is small enough to perform verification on, and Zhang [Zha06] uses a hand crafted Isabelle/HOL model to verify timing properties. In both cases, it is difficult to establish that their models faithfully represent the FlexRay specification.

Our approach is fundamentally different. We assume that the nodes in a network are always synchronised, so that we may abstract away from the clock synchronisation mechanism altogether and instead focus on the discrete behaviour of the startup protocol. This approach is similar to that of [Ste05b]. As the clock synchronisation mechanism is rather data intensive (it stores time stamps and performs calculations on them), this abstraction decreases the size of the model significantly, at the cost of not begin able to detect errors in the clock synchronisation mechanism.

We implement a discrete clock by means of a synchronisation barrier: all processes synchronise every clock tick. The resolution of the clock is chosen to match the duration of a single bit (`gdBit` in [Fle09]). In the mCRL2 modelling language, this amounts to allowing some actions to occur only in combination with a matching action from every other concurrently running process (details are discussed in Section 3.2.3). There is no need for a separate clock process.

Data Although we are modelling time at a resolution suitable to model every bit that is communicated in the protocol as-is, doing so leads to a state space that is much too large. For example, a frame containing 16 bits of data requires 80 bits

on the bus. For a network of three nodes, each sending a single frame, not taking into account any time in which the bus should be idle, simply summing up the initial phasing of nodes (assuming they start within one cycle of each other) would already take millions of states, and modelling every frame that could possibly be sent would be out of the question. We therefore try to compress the bit patterns into a minimal form that preserves some of the properties of real bit patterns on the bus.

Four types of symbols are of interest during startup: the CAS, frame headers, frame bodies and the *channel idle recognition point* (CHIRP). The latter is not a pattern that is actively sent, but a symbol that is decoded (meaning that a CHIRP event is generated by the CODEC) when no node sends data for a certain amount of time. Whenever a CHIRP is decoded by a node, it considers the bus to be idle until it detects that data is sent over the bus. We have chosen the symbols such that we can model the events that POC responds to. For instance, we need to be able to model the event that a frame header has been decoded from the bus, because the SDL diagrams in [Fle09] prescribe that such an event should trigger a reaction from POC.

To limit the size of the state space, we no longer require the amount of bits to be realistic: we allow frames consisting of a one-bit frame header and a one-bit frame body, and the CAS (which is usually at least 11 bits long) can be only two bits long. We design the model such that we can choose the size of our symbols arbitrarily.

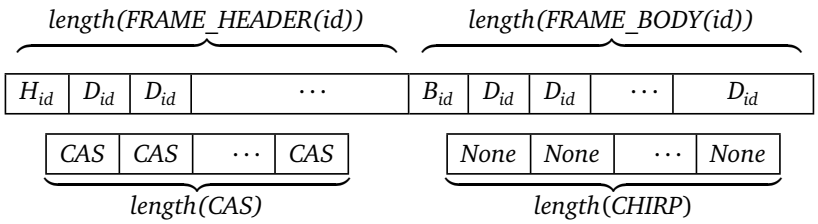


Figure 3.3: Encoding of symbols on the bus.

In our model, each bit on the bus can have one of six values: H_{id}, D_{id}, B_{id}, *CAS*, *None* or *Noise*. The encoding of symbols is shown schematically in Figure 3.3. For frame data, every bit also carries an *id* field that identifies its sender. This is used as an abstraction of the CRC checksum in the frame header and frame body: we assume the CRC is perfect, *i.e.*, that it passes if and only if the series of bits is equal to the sequence that was originally sent. We model this by checking that all bits in the sequence are sent by the same sender, and make sure that each *id* is only used by one node in our model.

We assume that the simultaneous sending of two bits that are not of value *None* will always result in *Noise*. In existing implementations, the CAS is a sequence of 'dominant' bits, so two simultaneously sent CAS signals result in a valid CAS symbol again. The protocol does however not require this. In this chapter we choose to not take into account this notion of dominance, so nodes have maximal potential to disturb each other's transmissions. We have performed verification on models where we have taken dominance into account, but this did not yield different results.

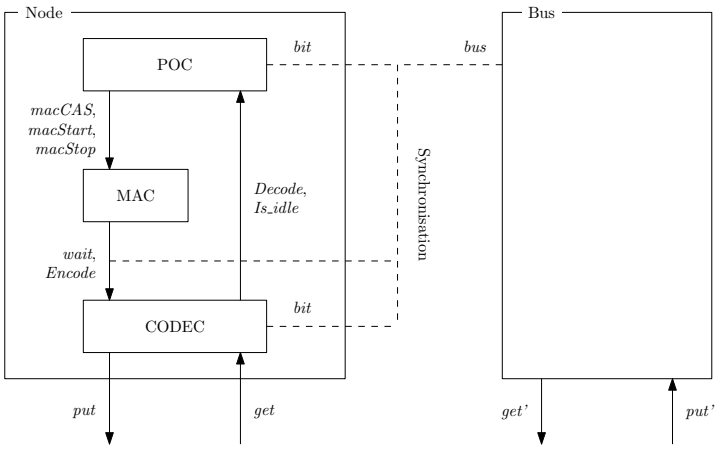


Figure 3.4: Detail of the `Node` and `Bus` processes.

3.2.2 Structure

We model a network consisting of three coldstart nodes that are all attached to a single channel in a bus topology. The channel is modelled by a process called *Bus*, which runs in parallel with three *Node* processes, each representing one coldstart node. The bus and the nodes synchronise on every clock tick. In between clock ticks, the bus goes through a writing phase, in which every node may write data to the bus using a *Put* action, and a reading phase, in which every node retrieves the combined signal using a *Get* action.

Figure 3.4 shows the structure of a node and how it is connected to the bus. Each node process consists of three communicating processes running in parallel: the *POC* process, which models the *Process Operation Control* SDL process, the *MAC* process, which (coarsely) models the *Media Access Control* SDL process and the *CODEC* process, which is a coarse model of the *Coding/Decoding* SDL process for one channel. The arrows indicate communication, and are labelled by the actions in the model that implement this communication. The dotted lines show which actions are synchronised every clock tick (the *bus, bit, wait* and *Encode* actions can only occur on a clock tick).

The MAC process is responsible for dispatching encoding requests to the CODEC. It can be ordered by POC to send a CAS, after which it starts sending frames periodically, or, in the case of integration into existing traffic, it can be requested to start sending frames immediately (but according to the schedule). When a startup attempt fails, it can be requested to go back to an inactive mode again.

The CODEC is modelled as a process that either reads from or writes to the bus. When in reading mode, it processes bits it reads from the bus, and does not write anything to the bus (which is implemented as writing silence to the bus). When it is in writing mode, bits it reads from the bus are ignored, and it writes an encoding of the last requested symbol to the bus. This can be seen from Figure 3-18 in [Fle09], where decoding is explicitly halted when an encoding request is received by the CODEC

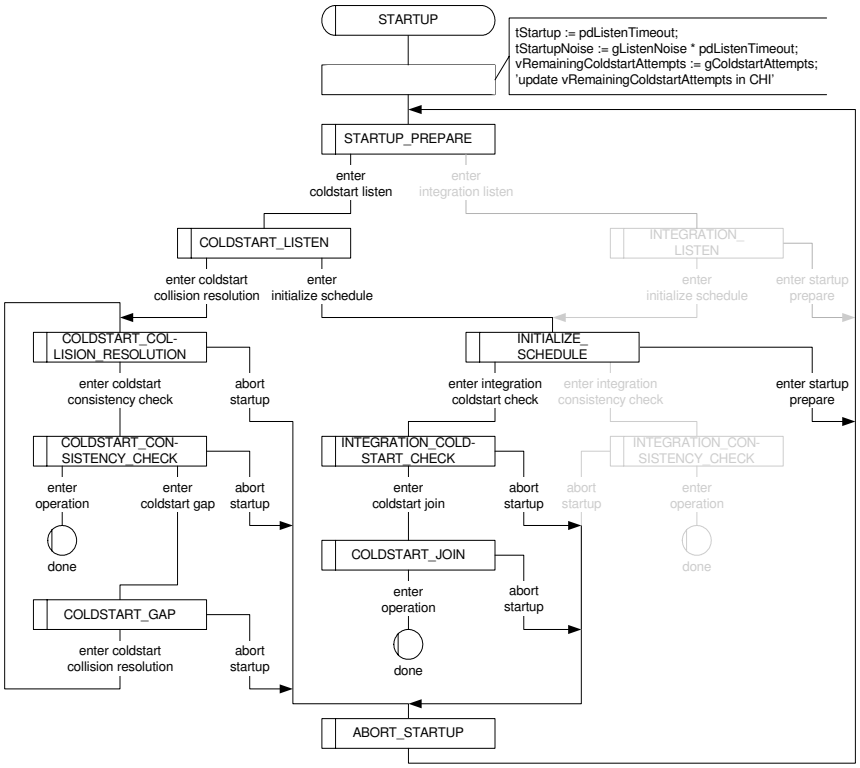


Figure 3.5: The POC startup phase, taken from [Fle09]. Grey parts are excluded from our model.

process.

The POC process is defined by a number of SDL procedures that call each other. We follow this structure by modelling each SDL procedure by its own mCRL2 process. Every process can be seen as a superstate that is input enabled with respect to the signals coming from MAC and CODEC. Only SDL procedures that are executed by coldstart nodes are modelled. Figure 3.5 shows the SDL diagram from [Fle09] that describes the top-level structure of the POC process, and shows which parts of the process we have modelled.

Decisions in these procedures that involve signals coming from the clock synchronisation mechanism—POC for instance uses timers and signals that are generated at the start of a cycle—are emulated using minimal local administration (usually a counter that is incremented after every clock tick).

3.2.3 The mCRL2 model

The complete model with which the results in this chapter can be reproduced is included in the mCRL2 distribution[1], version 201409.1, and can be found in the examples folder in the category 'industrial'. In this section, we explain some of the details of the model.

Extended syntax In order to create a more structured specification, an extension of the mCRL2 syntax was used. This syntax is not meant to provide a semantic extension of the language, but merely to syntactically abbreviate some common constructs so as to remove clutter. We therefore merely give an explanation of how to read the syntax, but do not recommend using it in any other setting.

The syntax is best explained using an example. Consider the following specification:

```
proc X = nested(n: Nat) {
  initial state A =
    a(n) . B(0);
  state B(m: Nat) =
    m < 10 -> b . B(m + 1)
  + m == 10 -> b . X(n + 1)
}
```

This specification is translated to the following, pure mCRL2 specification.

```
proc X(n: Nat) = X'A(n: Nat);
proc X'A(n: Nat) = a(n) . X'B(n, 0);
proc X'B(n: Nat, m: Nat) =
  m < 10 -> b . X'B(n, m + 1)
+ m == 10 -> b . X(n + 1);
```

The intuition is that every **nested** block defines some parameters that are shared by the states defined in it. The states are mCRL2 process specifications, and can therefore again be defined in terms of nested blocks.

A state may add parameters to the list provided in its associated **nested** statement. Duplicate parameter names are not allowed. When a process or state name is used within a state, a name lookup is done first in the scope of the current **nested** block, then in the one above that etc.

Obviously, name clashes are a big problem in this construction. The models presented in this document therefore use unique variable and process names in such a way that the translation scheme does not cause any ambiguity.

Bus model We wish to verify properties of a network with a deaf node (a node that cannot read anything from the bus) and a mute node (a node that cannot write anything to the bus). The easiest way to deal with this is to see such faulty behaviour as a trait of the physical bus.

The physical bus is modelled as a process that reads, per time slot, a signal from all connected nodes that are not mute and delivers the combination of all those signals

[1] The tool kit can be downloaded from http://www.mcrl2.org.

back to all connected nodes that are not deaf.

```
proc Bus(r, w: Sender, s: Signal) =
  (r <= NODES + 1) -> (
    sum s': Signal . put'(r, s') .
      Bus(r + 1, w, if(r == mute, s, combine(s, s')))
  ) <> (
    (w <= NODES + 1) -> (
      get'(w, if(w == deaf, NONE, s)) . Bus(r, w + 1, s)
    ) <> (
      bus(s) . Bus(1, 1, NONE)
    )
  );
```

Every node is identified with a value from `Sender`. The `Signal` data type contains the values that bits on the bus can take (as explained on page 35).

If a node is not sending, it is modelled as a node that is sending silence. The combination of a signal and silence is that signal. The combination of two signals is defined to be noise.

When a bit has been sent and received in this manner, the `bus` action marks the progress of time and the process repeats itself (it is part of the clock tick barrier).

In the full model, two variants of the bus are included. One of them reads in an arbitrary order, and one reads in a fixed order. The former was used for verification purposes (because it is the more general model), but the model as listed above was used to generate the traces in Section 3.3, as it is quicker. The reason the model that performs reads and writes in an arbitrary order is more time-consuming to generate is that it generates more states, and uses set operations where the ordered model needs only enumerated and integral types.

POC model Because the FlexRay protocol specification is already quite formal (in the sense that a systematic notation is used to describe processes), we set out to create our model in a way that remains as close to the specification as possible. Because a large part of the specification is abstracted away, we cannot use an automated conversion to an mCRL2 model, but where our model attains the same degree of detail as the original specification, we attempt to make our translation as systematic as possible. We have taken the diagrams from Chapter 7.2 in [Fle09] as the basis for our model. The semantics of these diagrams is given in [ITU99].

To create a model that would not inevitably blow up to unreasonable proportions, we use a slightly simplified version of these semantics. We assume that the calculations in the diagrams do not take time; moving from one state to the next can then be seen as an instantaneous change. Secondly, we assume that signals do not take time to be delivered. In [ITU99], event queues are used to capture the semantics of these phenomena. This would lead to a very large state space, as the state of the queues must be taken into account. Our simplifications eliminate the need for queues. The downside is that not all possible sequences of events that may occur in reality are modelled; for instance, in our model, signals from other processes only arrive when the receiving process is ready to process them.

We note here that the FlexRay specification does not claim to use the precise SDL

semantics (in fact, it claims that it may not do so). Given that the specification is at the detail level of a reference implementation, this is not surprising, as the use of event queues would be infeasible in a hardware implementation. The specification does however not give any guidelines on how to interpret the diagrams, which is why we have taken the official SDL semantics as a starting point.

Using the simplified semantics, the SDL diagrams in Chapter 7.2 of [Fle09] were translated into mCRL2 code (using a slightly altered syntax, see appendix 3.2.3). As an example, we show the part of the mCRL2 code that models the 'coldstart collision resolution' diagram in the specification (Figure 7–16 in [Fle09]):

```
state ColdstartCollisionResolution(timer: Nat) =
(
  % Inputs from CODEC
  decode(CAS) . bit . AbortStartup()
+ sum id': Sender . decode(FRAME_HEADER(id')) .
                   hdr(id, id') . AbortStartup()
+ sum id': Sender . decode(FRAME(id')) .
                   ColdstartCollisionResolution()

  % Wait for four cycles
+ (timer < FRM_START(id) + CYCLE_length * 4 - 1) ->
  bit . ColdstartCollisionResolution(timer=timer + 1)
+ (timer >= FRM_START(id) + CYCLE_length * 4 - 1) ->
  bit . cs_cons . ColdstartConsistencyCheck(0)
)
```

Looking at SDL diagram 7–12, we see a single state in which the POC may respond to a number of different events:

– *SyncCalcResult*. This event is emitted by the clock synchronisation process, just before a new communication cycle starts. We assume a global clock with a resolution of one bit, of which the ticks are modelled by the `bit` actions. Rather than waiting for four *SyncCalcResult* events, we instead look at the global clock to decide when four cycles have passed.
– *header received on A/B*. This event is emitted by CODEC right after a frame header has been received. It is modelled using the `decode` action with a `FRAME_HEADER` parameter.
– *symbol decoded on A/B*. This event is emitted by CODEC right after a collision avoidance symbol or media test symbol is decoded. It is again modelled using the `decode` action, this time with a `CAS` parameter.

In the mCRL2 model, we need to absorb `decode(FRAME(id'))` events (not doing so would prevent the source from sending the event, thus potentially leading to deadlock states). In the semantics of SDL [ITU99], this corresponds to discarding an event from the event queue when it cannot be processed.

To allow time to progress in a state, we must additionally allow the `bit` action to occur. The above piece of code implements a timeout of four cycles by letting time progress for four cycles, and then moving to the 'coldstart consistency check' state. If

a CAS or frame header is received before the timeout occurs, the startup is aborted.

MAC Media access control is modelled by the mCRL2 process below. It starts in inactive mode (not shown) and can then receive `macCAS`, `macStart` and `macStop` commands.

```
proc MAC(id: Sender, togo: Int, active: Bool) =
  macCAS . encode(CAS) .
    MAC(active=true, togo=FRM_START(id) + length(CAS))
+ macStart . MAC(active=true, togo=FRM_START(id))
+ macStop . wait . MAC(active=false)
+ active -> (
    (togo > 0) -> (
      wait . MAC(togo=togo - 1)
    ) <> (
      encode(FRAME_HEADER(id)) . MAC(togo=CYCLE_length)
    )
  ) <> wait . MAC();
```

When a `macCAS` command is received, it requests the CODEC to send a collision avoidance symbol and then switches to active mode. In the specification, MAC waits for one macrotick before sending the CAS, but it seems that the waiting for the macrotick is not intended as a delay, but merely as a check that the clock synchronisation process has started. Since we assume a global clock, we do not need to model this delay.

When a `macStart` command is received, the MAC proceeds to active mode, and when it receives a `macStop` command, it goes back to inactive mode. In active mode, MAC periodically requests CODEC to encode a frame.

CODEC The CODEC is modelled as a process that either reads from or writes to the bus. Its task is to translate symbols into bit patterns, and *vice versa*. When in reading mode, it processes bits it reads from the bus, and writes silence to the bus (*i.e.*, it does not write anything to the bus). When it is in writing mode, bits it reads from the bus are ignored, and it writes an encoding of the last requested symbol to the bus.

Remainder As can be seen from Figure 3.2, the POC process communicates with all other processes. We covered MAC and CODEC, and we argue that we may safely omit models for the remaining processes. By assuming a global clock, we can avoid modelling the macrotick generation, clock synchronisation startup and clock synchronisation processes. Furthermore we assume that nodes are never in coldstart inhibit mode. This eliminates the influence of the controller host interface, so we can also omit a model for that process. During startup, only the clock synchronisation depends on events generated by frame and symbol processing, so this process is also ignored.

3.2.4 Verification

Our goal is to verify that our model satisfies the requirement from Section 3.1.2, *i.e.*, that the network starts up correctly in the presence of certain faults. The faults that are within the scope of our investigations can all be seen as instances of a few general

problems: either a node is not able to send anything, a node is not able to receive anything, or the bus misbehaves in such a way that symbols are not always transmitted correctly. Only the periodic resetting of a node requires the node to display slightly more complicated behaviour. Two faults described in [Fle05c] comprise a node sporadically disturbing the bus; from the perspective of the correctly functioning nodes, however, this is not different to having a noisy bus.

Since for the POC a noisy signal is observably equal to no signal at all (the CODEC simply does not generate events), we model a limited set of scenarios. Each of the descriptions below describes two scenarios: one in which the node with the lowest identifier is the faulty node, and one in which another node is faulty. This is necessary because the protocol relies on a leader election mechanism that is not quite symmetric: although the process descriptions for startup are the same for every node, the leader that will be elected depends on the configuration of the nodes. The candidate configured with the lowest identifier will be elected as leader. At least one failure scenario (*viz.* the resetting node scenario below) [Ste05a] is only possible if the node with the lowest identifier is the faulty node.

In this manner, the following categories of scenarios are modelled.

Two nodes A faulty node does not switch on at all, so effectively there are only two nodes present in the network.

Silent node A faulty node is not able to send anything. Although we do model this separately, we note that this scenario is equivalent to the two-node scenario if we are not interested in the behaviour of the faulty node. We include this scenario because it shows that the silent node is still able to integrate into the communication correctly, albeit in a read-only mode.

Deaf node A faulty node does not receive anything.

Resetting node A node resets itself periodically.

Noisy channel Signals sent by nodes are corrupted on the channel. We use a noise model that consists of a burst length and a maximum backoff time. The burst length determines the maximum number of sequential bits that are corrupted, the maximum backoff time determines the maximum number of sequential bits that pass through the channel unaltered. Due to practical limitations, we were only able to model this scenario in a two-node scenario.

For each of these scenarios, we check that the correctly functioning nodes start up. We do this by checking three properties. The first is absence of deadlock; a reachable deadlock would indicate an error in the model, rather than in the FlexRay protocol, for the construction of the model is such that time is always allowed to progress.

Absence of deadlock is checked while traversing the state space. The other two properties are used to check whether the model violates our definition of correctness from Section 3.1.2. In fact, we encode a slightly stronger definition: every non-faulty node successfully terminates their startup procedure exactly once, and after all non-faulty nodes have terminated their startup procedure, they will keep communicating

according to schedule *forever*. When we present the results of our verification, we will see that weakening these properties to only require that communication according to schedule lasts for at least one cycle only affects the results of the 'noisy channel' scenario. On the other hand, for the scenarios in which the properties are satisfied, this stronger formulation gives us more confidence that the protocol is correct (and that our model is correct).

These last two properties are formulated in the first-order modal L_μ (see, *e.g.*, [GM99; GM14]). For brevity, we use mathematical syntax rather than concrete mCRL2 L_μ syntax, and constructs to help the mCRL2 toolset (*e.g.*, to prevent quantifiers from being expanded forever) are left out. The $[A]\varphi$ operator expresses that φ must hold after every sequence of actions satisfying the regular expression A. These regular expressions have sets of actions as their alphabet. The set of all actions occurring in the system is denoted *Act*; the complement (w.r.t. *Act*) of a set of actions A is denoted $\overline{A}$. The singleton set $\{a\}$ is denoted by a.

It is important to note that these formulas only represent the intended properties correctly if the system they are checked on is deadlock free, as otherwise the $[A]\varphi$ subformulae might trivially hold.

The second property asserts that eventually all correctly functioning nodes enter normal operation exactly once, an event that is flagged by the *enter_operation* action. It is expressed by the following formula, in which N is the total number of nodes and C is the set of correctly functioning nodes:

$$
\begin{aligned}
&\mu X(r : 2^{\mathbb{N}} = C).\\
&(\\
&\qquad r \neq \emptyset\\
&\quad \wedge \quad (\forall_{i:\mathbb{N}}\, [\mathit{enter\_operation}(i)](i \in r \wedge X(r \setminus \{i\})) \vee (i \notin C \wedge X(r)))\\
&\quad \wedge \quad [\overline{\{\mathit{enter\_operation}(i) \mid i \in \mathbb{N}\}}]X(r)\\
&) \vee (\\
&\quad r = \emptyset \wedge [\mathit{Act}^* \cdot \{\mathit{enter\_operation}(i) \mid i \in C\}]\mathbf{f}\\
&)
\end{aligned}
$$

The set r keeps track of which correctly functioning nodes are still running their startup procedure, and is initially assigned the value C. The least fixpoint X ensures that all paths along which $r \neq \emptyset$ are finite. The subformula

$$(\forall_{i:\mathbb{N}}\, [\mathit{enter\_operation}(i)](i \in r \wedge X(r \setminus \{i\})) \vee (i \notin C \wedge X(r))$$

removes i from r when along such a path the $\mathit{enter\_operation}(i)$ action is encountered. By requiring that $i \in r \vee i \notin C$ after every *enter_operation* action, we enforce that every process may execute this action only once. The conjunct that follows this subformula expresses that along these paths, actions other than *enter_operation* may occur (although they do not affect r).

From states in which X holds, we can reach a state in which the following holds:

$$r = \emptyset \wedge [\mathit{Act}^* \cdot \{\mathit{enter\_operation}(i) \mid i \in C\}]\mathbf{f}.$$

Because of how we defined r to change along the path to this state, we know that when we arrive in this state, every node in C has executed its corresponding *enter_operation*

action once, and in which the second conjunct holds, which says that no node from C will ever do an *enter_operation* action again ($\mathbf{f}$ holds for all paths satisfying the regular expression $Act^* \cdot \{enter\_operation(i) \mid i \in C\}$, *i.e.*, such paths are not allowed).

The last property says that eventually all correctly functioning nodes will keep receiving each other's messages. Even though our model is not intended to model the ongoing traffic after startup, we have constructed our model in such a way that this property should hold. If this property does not hold, then it is likely that the nodes did not synchronise correctly. It is characterized by the following formula:

$$
\begin{array}{l}
\mu X\,.\,[Act]X \vee \\
\nu Y(s : Symbol = firstsymbol)\,. \\
\mu Z(r : 2^{\mathbb{N}} = C \setminus sender(s))\,. \\
(\\
\qquad r \neq \emptyset \\
\quad \wedge \quad (\forall_{i:\mathbb{N},s':Symbol} \\
\qquad\quad [Decode(i,s')]((i \in C \wedge s = s' \wedge i \in r \wedge Z(r \setminus \{i\})) \vee \\
\qquad\qquad\qquad (i \notin C \wedge Z(r))) \\
\qquad) \\
\quad \wedge \quad [\overline{\{Decode(i,s') \mid i \in \mathbb{N} \wedge s' : Symbol\}}]Z(r) \\
) \vee (\\
\qquad r = \emptyset \\
\quad \wedge \quad Y(nextsymbol(s)) \\
)
\end{array}
$$

Fixpoint X holds in every state where always eventually Y will hold. We assume that *firstsymbol* and *nextsymbol* are mappings (constants are treated as mappings of arity zero) that define the FlexRay schedule, *i.e.*, they define a repetitive pattern of frame headers and frame bodies that we expect to see on the bus. Then Y is true in states from which all correct nodes will decode *firstsymbol* first, followed by *nextsymbol*(*firstsymbol*), etcetera: it represents an infinite repetition of finite paths along which the currently scheduled symbol is decoded. The sender of a symbol is excluded from the set of recipients. The subformula

$$[Decode(i,s')]((i \in C \wedge s = s' \wedge i \in r \wedge Z(r \setminus \{i\})) \vee (i \notin C \wedge Z(r)))$$

makes sure that correct nodes can only decode the right symbol, and can do so only once (by removing them from r), but allows faulty nodes to decode arbitrary symbols.

3.3 Results

The results in this section were initially obtained using the July 2011 release of the mCRL2 toolset. Verification of the properties was done by linearising the mCRL2 specification and combining it with the formulae to form parameterised Boolean equation systems. These were instantiated to Boolean equation systems, which were in turn reduced modulo stuttering equivalence on parity games. The resulting smaller equation

systems were then solved. A description of this procedure can be found in [CKW11], and is also described in Chapter 5 of this thesis.

With the 201409.1 release of the mCRL2 toolset, which contains the FlexRay specification in its catalogue of example specifications, the parity game reduction is no longer necessary to speed up the process. Instead, a more efficient parity game solver [Ver13] can be used to obtain these results. Detailed instructions on how to reproduce the results in this section can be found in the readme-file that accompanies the specification.

We note that it is also possible to check eventual startup and eventual communication by visual inspection. By hiding all actions but *enter_operation* and then reducing the state space using branching bisimulation, the first property can be checked by searching for cycles in the reduced state space. The second property can be checked manually by hiding all but *Decode*, reducing the state space using divergence preserving branching bisimulation and then visually inspecting all strongly connected components.

All three properties hold for the 'no faulty nodes' and 'two nodes' scenarios. The other fault scenarios we discuss seperately. The figures that illustrate each scenario are generated from traces in our model.

Mute node All three properties hold on the system. Manually inspecting the branching-bisimulation reduced state space reveals that the failing node can in this case enter normal operation using the wrong schedule (see Figure 3.6.a). The clock synchronisation process will allow this scenario, and frame and symbol processing will also not detect the mistake while the startup protocol has not finished. The mistake is harmless, however, because the silent node cannot disturb ongoing communication, and does not violate any requirement because it is the failing node that suffers the consequences. As soon as normal operation is entered, the clock correction process or the frame and symbol processing process of the faulty node will notice the error. A next attempt to integrate will succeed, because there is then already ongoing traffic.

Deaf node The state space is deadlock free, but neither of the properties hold, because there is a possibility of the network not starting at all. This violates requirement 2111 nr. 6 in [Fle05c]. Figure 3.6.b shows such a scenario.

The deaf node can happen to align its frames with those of another startup node, causing only the headers of the other node to be readable on the bus. The non-faulty node that is broadcasting startup frames will not detect that every sent frame is corrupted by the faulty node. Because the non-faulty node's frame headers are untouched, all other nodes will wait until it gives up after the maximum number of startup attempts.

Although this scenario is a valid trace in our model, it exposes an inaccuracy in the model: because we did not model the non-coldstart behaviour, Node 3 simply stops after it spent its maximum number of coldstart attempts (three in this case). In reality, it would switch to an integrating mode, and would still be able to start up the network together with Node 1.

This inaccuracy was noted by the authors of [Kor+13], in which the scenario is changed slightly to expose the faulty behaviour: if each node starts with two remain-

ing coldstart attempts (this is the minimal allowed configuration value), then Node 1 has spent one attempt after this scenario, and has one attempt left. The FlexRay protocol demands that at least two coldstart attempts are available in order to initiate a coldstart, which results in Node 1 switching to integration mode just like Node 2 and 3. We note that in the case of three or more coldstart attempts for at least two non-faulty nodes, the network will always start up because at least one node will be left with enough attempts to initiate a coldstart again.

The scenario was reproduced in our model, taking into account the ratios of the lengths of symbols and idle times on the bus (also taking into account overhead like frame / byte start sequences), and taking into account bus idle time that is enforced by the protocol (more specifically, the 'action point offset' and the idle time between the frame end sequence and the next slot boundary). The result is shown in Figure

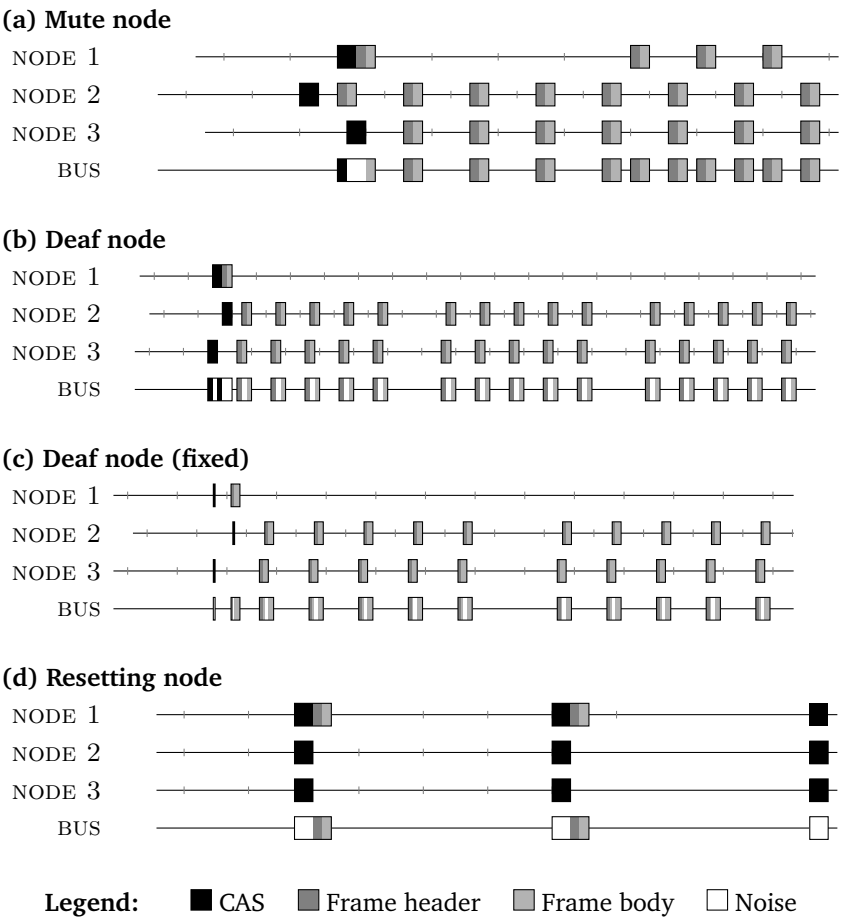


Figure 3.6: Traces extracted from the FlexRay startup phase model.

3.6.c, where the following key configuration values are used.

`gdSampleClockPeriod`	0.0125 μs	`pSamplesPerMicrotick`	2
`pMicroPerMacroNom`	240	`gdTSSTransmitter`	9 gdBit
`gdStaticSlot`	4 MT	`gPayloadLengthStatic`	3

Resetting node The state space is deadlock free, but neither of the properties hold. This violates requirement 2111 nr. 9 in [Fle05c]. Although this scenario was already known (it was described in [Ste05b]), the emergence of the trace in Figure 3.6.d gives us confidence that our model is correct. The trace shows that the leading node may cause startup of the network to fail by resetting itself every time it has sent a frame. In fact, it would just have to send the frame header, but the way we modelled our reset behaviour does not allow this.

It should be noted that in this scenario it is required that node 1 be the faulty node, which is not necessary in the scenarios for deaf and mute nodes.

Noisy channel The results depend very much on the parameters of the noise model. For an arbitrary noise pattern, it is obvious that the system will not start up. The channel could simply decide to corrupt all traffic going through it. The noise model we chose guarantees that some information will come through. Checking exactly for which values of maximum burst size and maximum backoff period the system starts correctly is too big a task, but simply trying a few settings soon gives an idea of how robust the system is. We made the following observation.

If there is noise on the channel for too long while nodes are trying to commence the startup procedure, then obviously startup may fail. The interesting scenarios are those in which some information can be communicated. However, if the minimum backoff time is less than the time needed for fault-free startup, then one of the sync frames of the leading coldstart nodes can always be corrupted, causing either the schedule initialisation or the consistency check of the other nodes to fail. If the presence of noise is the only anomaly in the system, then the minimum backoff time being at least the time required for fault-free startup is enough to guarantee that the system will come up.

It is interesting to note that in the verification of these properties, memory usage was not the bottleneck; the largest model used in our verification was that in which the resetting node was modelled, which consisted of around 26 million states and 76 million transitions. Generating this state space is rather time-consuming however, most likely due to the multi-way communication used to model the clock tick, which can give rise to rather large guard expressions.

The verification of properties on the network via instantiation of parameterised Boolean equation systems suffers from a similar problem. Although solving the generated equation system can be done quickly, generation takes a lot of time. This is currently preventing us from performing verification on models of networks with more than three nodes.

3.4 Related work

The FlexRay protocol has been studied quite extensively, from different perspectives. In this section we give a brief overview of previous studies known to us, and describe the aspects that these studies cover.

The German Verisoft project[2] is a project in the automotive realm that has the verification of the FlexRay protocol as a sub-goal. Within this project, Kühnel et al. aim to provide a framework in which distributed applications can be verified if they use a combination of an OSEKTime compliant (real-time) operating system, FTCom (a fault-tolerant communication layer for OSEKTime operating systems) and FlexRay communication [KS06b]. They use the FOCUS [BS01] language as a basis for their toolkit. A model is created on a specification that is "based on" the FlexRay protocol specification version 2.0. They apply the following abstractions to simplify their model [KS06a]:

1. All clocks are synchronized, i.e. clock synchronization is not modelled.
2. Start-up behaviour is not modelled (because the clocks are synchronised).
3. The coding/decoding process is not modelled.
4. Bus guardians are not modelled.
5. Only the static segment of a communication cycle is modelled, not the dynamic segment.
6. Slots are assumed to have a size of one tick in FOCUS.
7. Lossiness of the channel is not modelled.
8. Single-channel nodes are modelled.

The above lists the aspects of FlexRay that are not modelled, but we could not find a textual explanation of the part of FlexRay that is incorporated in the model. Instead, a specification in FOCUS is given directly, and it is shown—using a theorem prover—that the used interface specification of the FlexRay component is indeed refined by the presented model [Spi06].

The clock synchronization mentioned in abstraction 1 is a modification of a clock synchronisation protocol described by Lundelius and Lynch [LL84]. Barsotti et al. have verified (amongst other protocols) the latter [BNT07; Fon+05] using a combination of a theorem prover and an SMT[3] solver. Zhang notes, however, that the correctness of the FlexRay clock synchronisation protocol does not trivially follow from the correctness of Lundelius and Lynch's algorithm [Zha06].

The start-up behaviour (not modelled because of abstraction 2) is addressed by Malinsky in [Mal08; MN10]. He uses UPPAAL to create a timed-automata representation of a system consisting of two *coldstart* nodes and one non-coldstart node. Using a few different settings for a number of FlexRay parameters, this system is checked for deadlock, and it is checked that the system starts up normally. It should be noted that this setup is not a valid FlexRay setup, because in a network with three nodes all nodes must be coldstart nodes, according to [Fle09]. However, the requirements document

[2] http://www.verisoft.de/SubProject6.html
[3] Satisfiability Modulo Theories

[Fle05c] does state that startup must succeed when, due to a fault, only two coldstart nodes are active.

Local bus guardians [Fle05b] are considered by Zhang, who proves three functional properties under the assumption of synchronised clocks [Zha08]. This partially addresses abstraction 4, although central bus guardians [Fle05a] are not taken into account. Zhang does however mention research done towards systems employing a central bus guardian in another industry standard, Time-Triggered Architecture[4].

Pop et al. have looked into properties related to the dynamic segment in a FlexRay communication cycle as meant in abstraction 5. Their analysis is however of a rather different kind, as they provide a schedulability analysis based on a correctly working protocol [Pop+08]. Such results can be extremely useful in proving correctness of a distributed application, as it may guarantee that—under given circumstances—data will travel through the system.

Botaschanjan et al. present a methodology that uses the framework developed in the Verisoft project [Bot+06]. The method aims at formally modelling tasks (a concept taken from the automotive industry) in the FOCUS tool. The application logic can then be verified, while automatic, verified code generation should ensure that these properties still hold in the deployed system. Later this approach is refined and formalised [Bot+08].

Steiner uses the SAL model checker to find failures in the startup protocol [Ste05a; Ste05b]. He identifies a scenario in which the system does not start up due to a single fail-silent node.

3.5 Closing remarks

We have modelled a 3-node FlexRay network during communication start-up, using the mCRL2 modelling language. The core of the model was constructed by translating SDL specifications of the *process operation control* process on every node, which implements most of the discrete behaviour of a node during start-up. We formulated two properties on the model in the first-order modal μ-calculus.

Analysis of the model revealed two violations of the FlexRay requirements, one of which is a scenario that was not known before. This error could be detected because our model captures more details of the specification than the models used in [Ste05a].

It would be interesting to check the same properties on a 4-node network, as the newly uncovered fault scenario seems to rely on the current, very minimal setup. Currently the verification of a 4-node network is too time consuming. The culprit seems to be the multi-way communication that is used to implement our assumption of synchronously running nodes. We consider it future work to see if this problem can be avoided by choosing a different synchronisation method, or if it can be remedied by preprocessing the specifications that are currently processed directly by the mCRL2 toolset.

If the aforementioned scaling issues can be overcome, it would be interesting to also extend the model by adding more detail to the MAC, FSP and CODEC processes

[4] http://www.vmars.tuwien.ac.at/projects/tta

defined in the protocol specification, in the same manner as was now done for the POC process.

Although the FlexRay standard is declared final, and therefore the issues that were discovered can not be addressed by changing the standard, the use of (local or central) bus guardians can prevent the described failures. Kordes *et al.* describe how such an approach can be used to prevent the failures described in this chapter [Kor+14].

Chapter 4

A succinct translation from CTL* to FO-L_μ

In the previous chapter, we used mCRL2, a process algebraic behavioural specification formalism endowed with data and time, which is used extensively to model real life systems [Cra+13; GM14]. Properties about the behaviour of processes are described in variant of L_μ that is enriched with data and time, which we shall call *first-order modal μ-calculus*, or simply FO-L_μ [GW05a]. L_μ [Koz83; Eme97] is an extension of Hennessy-Milner logic [HM80] with least and greatest fixpoint operators.

By using data in mCRL2, one can specify state machines with an infinite action alphabet, giving rise to the need for a formalism that can express behavioural properties over such systems. FO-L_μ, in which quantification over data can be used and in which fixpoint variables may have data parameters, fulfills this need. Like L_μ, it is very expressive, but it is more practical because it is syntactically less minimalistic. Over the years, theories [GM99; GW05b] and open-source tools have been developed to verify properties in FO-L_μ.

Already in 1986, Emerson and Lei suggested that L_μ might serve as a uniform model checking framework, and showed that CTL can be translated succinctly into L_μ. However, they also noted that the only known translation from CTL* to L_μ was not succinct [EL86]. But if L_μ is to become a framework for model checking, it is certainly of importance that system properties can be expressed in a formula that is roughly comparable in size with a CTL* formula.

The original translation that Emerson and Lei mentioned consisted of the composition of an unpublished translation from CTL* to PDL-Δ by Wolper, and a translation from PDL-Δ to L_μ [EL86]. A simpler translation procedure was proposed in [Dam92], but this translation still yields formulae doubly exponential in the size of the input formula. Only in 1996, this translation was improved upon by Bhat and Cleaveland with an algorithm that translates CTL* to an equational variant of L_μ, only causing a single exponential blowup [BC96].

In this chapter, we show that a linear translation to the first-order modal L_μ is possible using only very simple data types. We use a strategy similar to that of Bhat and Cleaveland: we first focus on translating LTL, and then extend the translation to handle CTL* formulae.

From the context of the mCRL2 toolkit, there is also a very practical reason to have

a succinct translation from LTL or CTL*. For those unfamiliar with the modal L_μ, or for those who favour these logics over L_μ (and admittedly, many people do so at the time of writing), a linear translation enables us to easily use available L_μ-checkers to verify properties formulated in these other formalisms also. To support this, we show that model checking a translated formula is as efficient as the most efficient known algorithms for model checking the original.

This chapter is structured as follows. Section 4.1 shows how an LTL formula can be represented by a Büchi automaton. In section 4.2, we present the first-order modal μ-calculus FO-L_μ to which LTL formulae are translated in section 4.3. This translation is then lifted to one for CTL* in section 4.4.

4.1 Translation from LTL to Büchi automata

In this chapter, we assume that all Kripke structures are finite, and have a total transition relation (we are using the CTL* definition from Section 2.4, which is only defined for Kripke structures with a total transition relation).

As we stated in the introduction, we start by translating LTL formulas to FO-L_μ. Recall from the preliminaries that LTL is the subset of CTL* formulas of the form $\mathsf{A}f$, in which A and E do not occur in f.

Below we sketch how to create a generalized Büchi automaton $\mathcal{B}$ for an LTL formula $\mathsf{A}f$ of which the accepted language consists of all words (paths) that satisfy $\neg f$. If there is an accepting run of this Büchi automaton on some Kripke structure, then that Kripke structure must have a path that satisfies $\neg f$. If no accepting run can be found, it is therefore safe to conclude that $\neg f$ does not hold on all paths, and therefore, because LTL is closed under negation, f must hold on every path. The definitions below construct a Büchi automaton with this property for an LTL formula. These definitions are equivalent to those in [BK08], to which we refer for a full explanation of this model checking technique.

If $\mathsf{A}f$ is an LTL formula, then we define the *closure* of f, denoted $\mathsf{cl}(f)$, to be the set of all subformulae of f and their negation. Double negations are omitted, *i.e.*, formulae of the form $\neg\neg g$ are represented by g. For example, $\mathsf{cl}(a \mathsf{U} \neg b)$ is defined to be the set $\{a, \neg a, \neg b, b, a \mathsf{U} \neg b, \neg(a \mathsf{U} \neg b)\}$.

Given an LTL formula $\mathsf{A}f$ over atomic predicates AP, the *LTL automaton* for f is the generalized Büchi automaton $\mathcal{B}_f = \langle B, 2^{AP}, \rightarrow, \mathcal{F}, B_0 \rangle$, defined as follows.

- B is the largest subset of $2^{\mathsf{cl}(f)}$ such that for all $S \in B$ we have the following:
  - $\neg g \in S$ iff $g \notin S$
  - $g \wedge h \in S$ iff $g \in S$ and $h \in S$
  - if $h \in S$ and $g \mathsf{U} h \in \mathsf{cl}(f)$, then $g \mathsf{U} h \in S$
  - if $g \mathsf{U} h \in S$ and $h \notin S$, then $g \in S$
- $\xrightarrow{A}$ is the largest subset of $B \times 2^{AP} \times B$ such that for all $S \xrightarrow{A} S'$:
  - $A = S \cap AP$
  - $\mathsf{X}g \in S$ iff $\mathsf{X}g \in B$ and $g \in S'$

- $g \cup h \in S$ iff $g \cup h \in B$ and either $h \in S$, or $g \in S$ and $g \cup h \in S'$

– $\mathcal{F} = \{F_{g \cup h} \mid g \cup h \in \mathsf{cl}(f)\}$, where $F_{g \cup h} = \{S \in B \mid g \cup h \notin S \text{ or } h \in S\}$
– $B_0 = \{S \in B \mid \neg f \in S\}$

Theorem 4.1. *Let $\mathfrak{A} = \langle A, AP, \to, \ell \rangle$ be a Kripke structure, $a \in A$ and let $\mathsf{A}f$ be an LTL formula over AP. Then $\mathfrak{A}, a \models \mathsf{A}f$ if and only if $\mathcal{B}_f$ has no accepting run on a.*

Proof. The generalized Büchi automaton $\mathcal{B}_f$ is identical to $\mathcal{G}_{\neg f}$ defined in Theorem 5.37 in [BK08]. This theorem says that the infinite words of $\mathfrak{A}$ that satisfy $\neg f$ are exactly those in the ω-language of $\mathcal{G}_{\neg f}$. In other words, for every infinite path $a_0 \to a_1 \to \dots$ in $\mathfrak{A}$ that satisfies $\neg f$, there is an accepting path $b_0 \xrightarrow{\ell(a_0)} b_1 \xrightarrow{\ell(a_1)} \dots$ in $\mathcal{B}_f$, and *vice versa*. The path $b_0 \xrightarrow{\ell(a_0)} b_1 \xrightarrow{\ell(a_1)} \dots$ is an accepting run of $\mathcal{B}_f$ on $\mathfrak{A}$. □

An LTL automaton $\mathcal{B}^G = \langle B^G, L, \to^G, B_0^G, \mathcal{F} \rangle$ can be transformed to a normal Büchi automaton by making a copy for every acceptance set and linking those copies together cyclically. This construction is also explained in [BK08]. We give a precise definition here. Suppose that $k = |\mathcal{F}|$ for some k and $f : \{0, \dots, k-1\} \to \mathcal{F}$ enumerates $\mathcal{F}$ in an arbitrary way. A Büchi automaton that is equivalent to $\mathcal{B}^G$ is given by $\mathcal{B} = \langle B, L, \to, B_0, F \rangle$, where:

$$
\begin{aligned}
B &= B^G \times \{0, \dots, k-1\} \\
B_0 &= \{\langle S, 0 \rangle \in B \mid S \in B_0^G\} \\
F &= \{\langle S, i \rangle \in B \mid S \in f(i)\}
\end{aligned}
$$

and where $\to$ is defined as follows:

$$
\langle S, i \rangle \xrightarrow{A} \langle S', j \rangle \text{ iff } S \xrightarrow{A}{}^G S' \text{ and } \begin{cases} j = i & S \notin f(i) \\ j = (i+1) \bmod k & S \in f(i) \end{cases}
$$

The resulting Büchi automaton accepts the same language as $\mathcal{B}^G$.

4.2 First-order L_μ

In this section we introduce a first-order extension of L_μ, which we call FO-L_μ. It is a state-based variant of the (action-based) logic described in [GM99], and a first-order extension of the logic described in [BC96]. The logic is essentially a many-sorted first-order logic, extended with the box modality from Hennesy-Milner logic and a fixpoint operator.

Formulas of FO-L_μ are interpreted on a Kripke structure, *i.e.*, if φ is a FO-L_μ formula and $\mathfrak{A}$ is a Kripke structure, we may ask whether φ holds in a state s of $\mathfrak{A}$. As for other temporal logics, we denote the statement that φ holds in s by $\mathfrak{A}, s \models \varphi$.

Before we formally define the logic, we will first illustrate the concept with some examples. Consider the formula $\varphi(N)$, interpreted over a Kripke structure $\mathfrak{A}$, and

defined as follows, in which $\approx$ denotes the equality relation on natural numbers (to avoid confusion with other syntactic elements).

$$\varphi(N) \triangleq \mu X(n\colon \mathbb{N} = N)\,.\, n \approx 0 \vee (p \wedge [\cdot]X(n-1)) \vee [\cdot]X(n)$$

This formula holds in any state from which p holds at least N times on every path starting in that state. The fixpoint operator assigns to X a predicate such that $X(n)$ equals the expression right of the period for all $n \in \mathbb{N}$, and then the initialization '$= N$' expresses that the entire formula denotes the predicate $X(N)$ obtained in this way.

Note that $X(0)$ always holds (because of the $n \approx 0$ disjunct). Therefore, $\mathfrak{A}, s \models \varphi(0)$ for every state s. $X(1)$ holds in any state in which p holds: the disjunct $p \wedge [\cdot]X(n-1)$ requires that p holds in the current state, and that $X(n-1)$ (in this case: $X(0)$) holds in all successor states. As $X(0)$ holds in all states, so in particular for all successor states. So $\mathfrak{A}, s \models \varphi(1)$ for at least all s labelled with p. $X(1)$ also holds in states from which all paths reach a state labelled with p: if a state s can only reach states s' for which $X(1)$ holds in one step, then the disjunct $[\cdot]X(n)$ is true, which means that $X(1)$ must also hold for s. Inductively continuing this argument, $X(1)$ must therefore hold for all states that eventually reach a state labelled with p. The least fixpoint operator μ requires that $X(1)$ is chosen as small as possible, so $X(1)$ holds only for the states we just identified. Another induction shows that for any N and s, $\mathfrak{A}, s \models \varphi(N)$ if and only if all paths from s visit a p-labelled state at least N times.

In a similar way, we can define a property that says that from a state, one can stay in a p-labelled state for any finite amount of time, after which a state is visited that is no longer labelled with p:

$$\forall_{N:\mathbb{N}}\ \mu X(n\colon \mathbb{N} = N)\,.\, (\neg p \wedge n \approx 0) \vee (p \wedge \langle\cdot\rangle X(n-1))$$

Formulas of the form $\langle\cdot\rangle\psi$ are the dual of $[\cdot]\psi$: they express that the current state has at least one successor for which ψ holds. A similar reasoning can now be followed as before to see that this formula expresses the desired property.

Note that in the examples above, we used quantification over natural numbers, a relation on natural numbers (equality), and a function from natural numbers to natural numbers (the $-$ operator). Furthermore, we used quantification over a sort (quantification over natural numbers, as opposed to quantification over the Booleans). To give a semantics to expressions like these, we will associate with every formula a structure $\mathfrak{D} = \langle \Sigma, \mathbb{D}, \mathcal{I} \rangle$, which will be used to interpret them. Sorts are then interpreted as subsets of $\mathbb{D}$, and quantification in formulas then ranges over these subsets. In this chapter, we will always be using the same structure. This structure, along with some notation and definitions to deal with our multi-sorted setting, is detailed in the next section.

4.2.1 Data

We assume throughout the chapter that we are working in the context of a structure $\mathfrak{D} = \langle \Sigma, \mathbb{D}, \mathcal{I} \rangle$ with signature $\Sigma = \langle \mathcal{R}, \mathcal{F}, \mathsf{ar} \rangle$, for which $\mathcal{R} = \emptyset$. We deviate from our

usual notation to stay closer to notational conventions used in other papers about modal μ-calculi, and write $[\![t]\!]$ rather than $t^{\mathfrak{A}}$ to denote the interpretation of t in $\mathfrak{A}$.

Because we are working in a many-sorted setting, we extend Σ with an additional set $\mathcal{S}$ of sorts (here we deviate from our usual notion of signature). With each sort $D \in \mathcal{S}$, we associate a semantic set $[\![D]\!] \subseteq \mathbb{D}$.

Function symbols have a sort; that is, every function symbol f with arity $\mathsf{ar}(f) = n$ is assigned a sort $D_1 \times \ldots \times D_n \to D_0$ with $D_i \in \mathcal{S}$ for all $0 \leq i \leq n$ (so in particular, if $\mathsf{ar}(f) = 0$, then its sort is some $D_0 \in \mathcal{S}$). Terms are built in the usual way, but with one extra restriction. For a term $f(t_1, \ldots, t_n)$, in which f has sort $D_1 \times \ldots \times D_n \to D_0$, we require that $[\![\mathsf{sort}(t_i)]\!] \subseteq [\![D_i]\!]$ for all $1 \leq i \leq n$.

We assume the existence of a function sort on terms, that assigns to every variable $v \in \mathcal{V}$ a single sort $\mathsf{sort}(v) \in \mathcal{S}$. For a composite term $t = f(t_1, \ldots, t_n)$ with outer symbol f of sort $D_1 \times \ldots \times D_n \to D_0$, we define $\mathsf{sort}(t) = D_0$.

To be able to define a fixpoint operator, we introduce a set $\mathcal{X}$ of second-order variables. Each $X \in \mathcal{X}$ is assigned an arity $\mathsf{ar}(X)$ and a sort $D_1 \times \ldots \times D_{\mathsf{ar}(X)}$.

Rather than having a single environment that assigns values to variables, we have separate environments for first-order variables and for second-order variables, following the notation in [GW05a]. So-called data environments assign values from $[\![\mathsf{sort}(v)]\!]$ to first-order variables $v \in \mathcal{V}$. Data environments are usually called δ, δ', and so on.

Predicate variables must be interpreted in the context of a Kripke structure. Given a Kripke structure with states S, predicate environments assign a function of the type $[\![D_1]\!] \times \ldots \times [\![D_n]\!] \to 2^S$ to every $X \in \mathcal{X}$ of sort $D_1 \times \ldots \times D_n$.

We write $\delta[d \mapsto v]$ to denote the data environment δ' for which $\delta'(d') = \delta(d')$ for all $d' \neq d$ and $\delta'(d) = v$. Similar notation is used for predicate environments.

We assume the existence of a sort B and constant function symbols $\mathbf{t}, \mathbf{f}$, representing the Booleans, with $[\![B]\!] = \mathbb{B} = \{\mathbb{t}, \mathbb{f}\}$ and a sort N, representing the natural numbers, with $[\![N]\!] = \mathbb{N}$. For the natural numbers, we assume that we have constants $\mathbf{0}, \mathbf{1}$, etc., such that $[\![\mathbf{0}]\!]^\delta = 0$, $[\![\mathbf{1}]\!]^\delta = 1$, and so on.

4.2.2 Syntax and semantics

FO-L_μ formulas reason about Kripke structures, that is, they partition the states of a Kripke structure into a set for which the formula holds, and a set for which it does not hold. The syntax and semantics of FO-L_μ therefore refer to the Kripke structure that it is evaluated on. For the rest of the chapter, fix a finite Kripke structure $\mathfrak{A} = \langle A, AP, \to, \ell \rangle$. We will assume that every FO-L_μ formula we encounter is evaluated on $\mathfrak{A}$.

Syntax The syntax of a FO-L_μ formula is defined by the following grammar:

$$\varphi, \psi ::= t \mid X(t_1, \ldots, t_n) \mid \neg\varphi \mid \varphi \wedge \psi \mid [\cdot]\varphi \mid \forall_{d:D}\ \varphi \mid \mu X(d_1 : D_1 = t_1, \ldots, d_n : D_n = t_n)\,.\,\varphi$$

In the above, t, t_1, etc. are terms, $d, d_1, \ldots \in \mathcal{V}$ are first-order variables, $D, D_1, \ldots \in \mathcal{S}$ are sorts and $X \in \mathcal{X}$ is a fixpoint variable. We require that X has sort $D_1 \times \ldots \times D_n$ and $\mathsf{sort}(d_i) = D_i$ and $[\![\mathsf{sort}(t_i)]\!] \subseteq [\![D_i]\!]$ for all $1 \leq i \leq n$.

Additionally, we require that $\mathsf{sort}(t) \subseteq \mathbb{B} \cup AP \cup 2^{AP}$. Intuitively, t is a predicate on states: $\mathbb{t}$ hold in all states, $\mathbb{f}$ holds in no state, $p \in AP$ holds in those states a such that $p \in \ell(s)$, and $P \subseteq AP$ holds in those a such that $P = \ell(s)$.

The fixpoint operator has the same binding strength as the universal quantifier; both bind weaker than all other operators. When a fixpoint variable has arity zero, then the parameter list, including the parentheses, is omitted.

The *size* of a formula φ, denoted $|\varphi|$, is the number of subformulae it contains. The size of data expressions is defined in the same manner.

Semantics The interpretation of a FO-L$_\mu$ formula φ in the context of $\mathfrak{A}$, a data environment δ, a predicate environment θ is denoted by $[\![\varphi]\!]^{\theta\delta}$. It defines the set of states of $\mathfrak{A}$ for which the formula holds, given specific values for δ and θ. The fact that a formula holds in a state a of $\mathfrak{A}$, in the environments δ and θ, *i.e.*, the fact that $a \in [\![\varphi]\!]^{\theta\delta}$, is denoted $\mathfrak{A}, a \models_{\theta,\delta} \varphi$. If $\mathfrak{A}, a \models_{\theta,\delta} \varphi$ for every θ and δ (this holds for any closed formula), then we write $\mathfrak{A}, a \models \varphi$. Since we have fixed $\mathfrak{A}$ for this chapter, we will usually omit it.

The formal definition of $[\![\varphi]\!]^{\theta\delta}$ is given in Table 4.1. In this table, the endofunction $\mathbf{T}$ calculates, given a value for X, the function from $[\![D_1]\!] \times \ldots \times [\![D_n]\!]$ to 2^S that φ induces. The least fixpoint of the function is then used to give an interpretation to X, effectively equating the interpretations of $X(t_1, \ldots, t_n)$ and $\varphi[t_1/d_1]\ldots[t_n/d_n]$ in the scope of the fixpoint binder (where t_i/d_i denotes the syntactic replacement of unbound occurrences of d_i by t_i in terms occurring in φ).

We use the following standard abbreviations to denote some useful derived operators, where $\varphi[\neg X/X]$ stands for the expression φ in which every occurrence of $X(t_1, \ldots, t_n)$ for unbound X has been replaced by $\neg X(t_1, \ldots, t_n)$:

$$\begin{aligned}
\varphi \vee \psi &\triangleq \neg(\neg\varphi \wedge \neg\psi) \\
\varphi \Rightarrow \psi &\triangleq \neg\varphi \vee \psi \\
\varphi \Leftrightarrow \psi &\triangleq (\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi) \\
\langle\cdot\rangle\varphi &\triangleq \neg[\cdot]\neg\varphi \\
\exists_{d:D}\ \varphi &\triangleq \neg\forall_{d:D}\ \neg\varphi \\
\nu X(d_1 : D_1 = t_1, \ldots)\,.\,\varphi &\triangleq \neg\mu X(d_1 : D_1 = t_1, \ldots)\,.\,\neg\varphi[\neg X/X]
\end{aligned}$$

It is important to note that the least fixpoint of $\mathbf{T}$ in Table 4.1 does not always exist. However, if in the above definition, φ can be transformed to *positive normal form* [BS06], in which negation only occurs on the level of atomic propositions and in which all bound variables are distinct, then the existence of such a fixpoint is guaranteed. Let $\mathbb{X} = [\![D_1]\!] \times \ldots \times [\![D_n]\!]$. The claim that this fixpoint exists is justified by the fact that we can define an ordering $\sqsubseteq$ on $\mathbb{X} \to 2^S$ (which is the domain of $\mathbf{T}$) such that $F \sqsubseteq G$ if and only if $F(v_1, \ldots, v_n) \subseteq G(v_1, \ldots, v_n)$ for all $v_1 \in [\![D_1]\!], \ldots, v_n \in [\![D_n]\!]$. Then $\langle \mathbb{X} \to 2^S, \sqsubseteq \rangle$ is a complete lattice and because $\mathbf{T}$ is monotonic over this lattice (see [GM99]), Tarski's theorem [Tar55] can be applied to establish that it has a least fixpoint. Furthermore, this fixpoint may be approximated by applying $\mathbf{T}$ a number of times to the infimum of the lattice (in case of a least fixpoint) or to the supremum of the lattice (for a greatest fixpoint).

4.3 Translating LTL to FO-L$_\mu$

In this section we provide a translation of LTL to FO-L$_\mu$. We first remark that this translation is not straightforward, in the sense that a simple syntactic translation procedure has not been found. We illustrate the difficulty with a small example. Consider the following two standard translations from LTL to L$_\mu$.

$$\mathsf{A}\, p\, \mathsf{U}\, q \overset{\text{trans}}{=} \mu X \,.\, q \vee (p \wedge [\cdot]X) \qquad \mathsf{A}\, p\, \mathsf{R}\, q \overset{\text{trans}}{=} \nu X \,.\, q \wedge (p \vee [\cdot]X)$$

The above translations appear often in literature, and at first sight seem very convenient. For example, it is easy to see that a translation for $\mathsf{AF}q = \mathbf{t}\,\mathsf{U}\,q$ and $\mathsf{AG}q = \mathbf{f}\,\mathsf{R}\,q$ can be obtained from the above by simply substituting p with $\mathbf{t}$ and $\mathbf{f}$ respectively, yielding $\mu X \,.\, q \vee [\cdot]X$ and $\nu X \,.\, q \wedge [\cdot]X$ respectively. The CTL formula $\mathsf{AFAG}\ q$ can be obtained in a similar manner, yielding $\mu X \,.\, (\nu Y \,.\, q \wedge [\cdot]Y) \vee [\cdot]X$. However, the LTL formula $\mathsf{AFG}q$ cannot be obtained by the same simple syntactic replacing, as we only have patterns to translate the state formulas $\mathsf{AF}q$ and $\mathsf{AG}q$, but not for the path formula $\mathsf{G}q$ (for a more detailed treatment on the expressive power of CTL and LTL, including an explanation of this specific case, see [HR04]). The following L$_\mu$ formula is a proper translation of $\mathsf{AFG}q$.

$$\mu X \,.\, \nu Y \,.\, (q \wedge [\cdot]Y) \vee [\cdot]X \tag{4.1}$$

$$[\![t]\!]^{\theta\delta} \triangleq \begin{cases} A, & [\![t]\!]^\delta = \mathbb{t} \\ \emptyset, & [\![t]\!]^\delta = \mathbb{f} \\ \{a \in A \mid [\![t]\!]^\delta \in \ell(a)\}, & [\![t]\!]^\delta \in AP \\ \{a \in A \mid [\![t]\!]^\delta = \ell(a)\}, & [\![t]\!]^\delta \subseteq AP \end{cases}$$

$$[\![X(t_1,\ldots,t_n)]\!]^{\theta\delta} \triangleq \theta(X)([\![t_1]\!]^\delta,\ldots,[\![t_n]\!]^\delta)$$

$$[\![\neg\varphi]\!]^{\theta\delta} \triangleq A \setminus [\![\varphi]\!]^{\theta\delta}$$

$$[\![\varphi \wedge \psi]\!]^{\theta\delta} \triangleq [\![\varphi]\!]^{\theta\delta} \cap [\![\psi]\!]^{\theta\delta}$$

$$[\![[\cdot]\varphi]\!]^{\theta\delta} \triangleq \{a \in A \mid \forall_{a' \in A}\, a \to a' \Rightarrow a' \in [\![\varphi]\!]^{\theta\delta}\}$$

$$[\![\forall_{d:D}\ \varphi]\!]^{\theta\delta} \triangleq \bigcap_{v \in [\![D]\!]} [\![\varphi]\!]^{\theta\delta[d \mapsto v]}$$

$$[\![\mu X(d_1 : D_1 = t_1,\ldots, d_n : D_n = t_n) \,.\, \varphi]\!]^{\theta\delta} \triangleq (\mathbf{lfp}\,\mathbf{T})([\![t_1]\!]^\delta,\ldots,[\![t_n]\!]^\delta)$$

with predicate transformer $\mathbf{T}$ defined as:

$$\mathbf{T}(F : [\![D_1]\!] \times \ldots \times [\![D_n]\!] \to 2^A) \triangleq \lambda v_1,\ldots,v_n \,.\, [\![\varphi]\!]^{\theta[X \mapsto F]\delta[d_1 \mapsto v_1,\ldots,d_n \mapsto v_n]}$$

Table 4.1: Semantics of FO-L$_\mu$ over a Kripke structure $\langle A, AP, \to, \ell\rangle$.

We note that this formula can be replaced by $\mu X \,.\, \nu Y \,.\, (q \wedge [\cdot]Y) \vee (\neg q \wedge [\cdot]X)$, which might seem stronger at first sight. The additional conjunct however does not change the meaning of the formula, because both fixpoints must identify the same set of nodes, and therefore in particular $[\cdot]X \Rightarrow [\cdot]Y$. We use a similar construction in this chapter to make the complexity analysis easier, even though we do not need this alternative formulation for our proof of correctness.

Notice that formula (4.1) expresses that eventually, any path will only pass through states that satisfy q. If q is taken to mean 'is not an accepting state', and if we view the Büchi automaton as a Kripke structure by ignoring the edge labels, then this formula expresses the absence of an accepting path in a Büchi automaton. Because a translation from LTL to Büchi automata is already known, it seems natural to use this formula as a framework for our translation. We will represent the Büchi automaton that is obtained from the translation in Section 4.1 using the notion of data in FO-L_μ.

Before we start our translation, we illustrate the use of data with some examples. The introduction of data allows us to formulate certain properties more concisely, by exploiting repetitive structures in the formula. Consider for instance the following formula.

$$\mu X \,.\, \langle\cdot\rangle X \vee (p(\mathbf{0}) \wedge \mu Y \,.\, \langle\cdot\rangle Y \vee (p(\mathbf{1}) \wedge \mu Z \,.\, \langle\cdot\rangle Z \vee p(\mathbf{2})))$$

This formula expresses that first a state in which $p(0)$ holds is reachable, then a state in which $p(1)$ holds and finally one in which $p(2)$ holds. This formula (and any extension thereof) can also be expressed as follows:

$$\mu X(i : N = \mathbf{0}) \,.\, \langle\cdot\rangle X(i) \vee (p(i) \wedge \langle\cdot\rangle X(i + \mathbf{1})) \vee i \approx \mathbf{3}$$

In the above, $[\![i \approx \mathbf{3}]\!]^\delta$ has the standard arithmetic meaning of '$[\![i]\!]^\delta = [\![3]\!]^\delta$' (we use a different equality symbol than usual to distinguish it from the syntactic equality relation on FO-L_μ formulas). Note that the formula above has collapsed the fixpoints into a single one, and that the number of boolean operators has also diminished.

Another example is the following formula, which expresses that out of the first $2k$ states visited, k states must be labelled with p:

$$\begin{aligned}\mu X(n : N = \mathbf{0}, m : N = \mathbf{0}) \,.\;\; & (n \approx k \wedge m \approx k) \vee \\ & ([\cdot]X(n + \mathbf{1}, m) \wedge p) \vee \\ & ([\cdot]X(n, m + \mathbf{1}) \wedge \neg p)\end{aligned}$$

In this formula, m and n count the number of p-states (resp. $\neg p$-states) that have been seen; initially they are set to zero, and after every step, $\mathbf{1}$ is added to n if the previous state was labelled with p, or to m if it was not. The size of this formula is $O(1)$, where a straightforward encoding in the normal L_μ would take $O(2^k)$ space (or $O(k^2)$ in the equational variant used in [BC96]).

We now return to the problem of translating LTL to the first-order L_μ. We base our translation on Büchi automaton representations of LTL formulas as described in Section 4.1. This representation is encoded into a data structure consisting of booleans and natural numbers. We express a FO-L_μ property that in a sense 'synchronizes' steps in the Büchi automaton (utilizing the data structure) with steps that are made in the

transition system against which the formula is checked. Keeping this synchrony intact, we can use the standard translation of $\mathsf{AFG}q$ to express that this Büchi automaton does not accept any path of the transition system.

Formally speaking, we assume that we are given an LTL automaton $\mathcal{B}_f$, and construct a FO-L$_\mu$ formula that is true in a state s_0 of the Kripke structure $\mathfrak{A}$ it is interpreted on if and only if $\mathcal{L}(\mathfrak{A}) \cap \mathcal{L}(\mathcal{B}_f) = \emptyset$, *i.e.*, the accepted language of the product of the Kripke structure and the Büchi automaton is empty. Using Theorem 4.1, we may then conclude that $\mathfrak{A} \models f$.

To aid the reader, we adopt the convention to use s (and s', s_0, etc.) for variables (which belong to the syntactic domain) denoting Kripke structure states, q, q', etc., for variables denoting Büchi automaton states, and a and b for the actual (semantic) states of Kripke structures and Büchi automata, respectively. For Büchi automaton labels we use variables p denoting subsets $P \subseteq AP$.

In the formula we construct, we use the sets and relations defined by our Büchi automaton. We do distinguish between syntax and semantics of these sets and relations in our notation; that is, we assume for a Büchi automaton $\langle B, L, \to, F, B_0 \rangle$ that there are data sorts B and L, a mapping $\to$ and sets F and B_0 that behave like their semantic counterparts. For instance, we assume they are defined such that $[\![q \xrightarrow{p} q']\!]^{\theta\delta} = \emptyset$ is equivalent to $\neg[\![q]\!]^{\delta} \xrightarrow{[\![p]\!]^{\delta}} [\![q']\!]^{\delta}$. In a similar way we take the liberty of using the Boolean connectives and the quantifiers both as syntactic elements of FO-L$_\mu$, and as semantic first-order operations.

Definition 4.1. *We define a translation function* tr *that generates a FO-L$_\mu$ formula from a Büchi automaton. Let $\mathcal{B} = \langle B, L, \to, F, B_0 \rangle$ be a Büchi automaton.*

$$\mathsf{tr}(\mathcal{B}) = \forall_{q_0 : B}\ q_0 \in B_0 \Rightarrow \mathsf{X}$$

where X *is a syntactic abbreviation for:*

$$\begin{aligned}
\mathsf{X} &= \mu X(q : B = q_0)\,.\,\mathsf{Y} \\
\mathsf{Y} &= \nu Y(q : B = q)\,.\,\mathsf{Z} \\
\mathsf{Z} &= \forall_{p:L}\ \forall_{q':B}\ (p \wedge q \xrightarrow{p} q') \Rightarrow [\cdot]\big((X(q') \wedge q \in F) \vee (Y(q') \wedge q \notin F)\big)
\end{aligned}$$

The variable q is bound twice in this formula, but because the variable declared second (as a parameter of Y) is always assigned the value of the first (the parameter of X), this can cause no confusion. In the remainder of this document, the variables q from this formula can therefore be thought of as 'the current state in the Büchi automaton'. This makes the presentation of the proofs that follow a little bit easier.

Note that Z quantifies over those q' and p that form a single step in the Büchi automaton from state q. The assertion p ensures that the required property is only checked along transitions in the Büchi automaton that have the label of the current state in the Kripke structure. In this manner, the quantifier and implication realise the aforementioned synchrony. In effect the formula $\mu X\ .\ \nu Y\ .\ [\cdot]X \vee (q \notin F \wedge [\cdot]Y)$ is checked on the paths of the Büchi automaton that have a corresponding path in the Kripke structure (*i.e.*, the runs of $\mathcal{B}$ on $\mathfrak{A}$).

If θ is a predicate environment and δ a data environment, then the semantics of X on $\mathfrak{A}$ under these environments is $[\![\mathsf{X}]\!]^{\theta\delta}$. The definition of semantics for FO-L$_\mu$ says this is equal to $(\mathbf{lfp}\,\mathbf{T})([\![q]\!]^{\delta})$, where $\mathbf{T}\colon (B \to 2^A) \to (B \to 2^A)$ is defined as:

$$\mathbf{T}(\hat{X}) = \lambda b\,.\,[\![\mathsf{Y}]\!]^{\theta[X\mapsto\hat{X}]\delta[q\mapsto b]}$$

As explained in Section 4.2, we can calculate this fixpoint by starting with an initial approximation $\hat{X}^0$ that is the minimal element of the lattice $\langle B \to 2^A, \sqsubseteq\rangle$, and then choosing the next approximation $\hat{X}^{m+1} = \Phi(\hat{X}^m)$. Concretely, the approximations for $\mathbf{lfp}\,\mathbf{T}$ are given as follows:

$$\hat{X}^0(b) = \emptyset$$
$$\hat{X}^{m+1}(b) = [\![\mathsf{Y}]\!]^{\theta[X\mapsto\hat{X}^m]\delta[q\mapsto b]}$$

Note that to compute an approximation for X, we must evaluate the formula Y, the solution of which is again a fixpoint. Therefore, every approximation of X gives rise to a fixpoint computation for Y under the current approximation for X. This fixpoint computation can again be done via approximation, leading to the following approximations for Y.

$$\hat{Y}_m^0(b) = S$$
$$\hat{Y}_m^{n+1}(b) = [\![\mathsf{Z}]\!]^{\theta[X\mapsto\hat{X}^m, Y\mapsto\hat{Y}_m^n]\delta[q\mapsto b]}$$

Note that $\hat{X}^m(b) = \hat{Y}_m^n(b)$ for large enough n. Using these definitions, we show the relationship between the L$_\mu$ formula of definition 4.1 and the Büchi automaton it was generated from.

For the rest of this section, fix a Büchi automaton $\mathcal{B} = \langle B, L, \to, B_0, F\rangle$. Remember that we already fixed some Kripke structure $\mathfrak{A} = \langle A, AP, \to, \ell\rangle$, and let $a_0 \in A$ and $b_0 \in B_0$.

Lemma 4.1 ($\Rightarrow$). *If there is an accepting run of $\mathcal{B}$ on a_0 that starts in b_0, then $a_0 \not\models_{\theta,\delta} \mathsf{X}$ for all θ and δ such that $\delta(q_0) = b_0$.*

Proof. Let θ and δ be such that $\delta(q_0) = b_0$. Suppose that there is such an accepting run, then this is witnessed by sequences $a_0, a_1, \ldots$ and $b_0, b_1, \ldots$ such that $a_i \to a_{i+1}$ and $b_i \xrightarrow{\ell(a_i)} b_{i+1}$ for all $i \geq 0$. We prove that $a_0 \not\models \mathsf{X}$ by showing that $\forall_{m,i\in\mathbb{N}}\ a_i \notin \hat{X}^m(b_i)$ by using induction on m. Because $[\![\mathsf{X}]\!]^{\theta\delta} = \hat{X}^m$ for large enough m, the required result then follows.

For $m = 0$, this holds trivially because $\hat{X}^0(b_i) = \emptyset$ for all i. For $m = l + 1$, assume:

$$\forall_{i\in\mathbb{N}}\ a_i \notin \hat{X}^l(b_i) \tag{IH}$$

We need to show that $\forall_{i\in\mathbb{N}}\ a_i \notin \hat{X}^m(b_i)$, which is equivalent to the following:

$$\forall_{i\in\mathbb{N}}\ a_i \notin [\![\mathsf{Y}]\!]^{\theta[X\mapsto\hat{X}^l]\delta[q\mapsto b_i]}$$

Let $i \in \mathbb{N}$. It is sufficient to prove that there is some n for which $a_i \notin \hat{Y}_l^n(b_i)$. We will do so by first showing that a_{i+k} (for some $k \geq 0$) is not an element of $\hat{Y}_l^1(b_{i+k})$, and then using an inductive argument to show that a_i cannot be in $\hat{Y}_l^n(b_i)$ for some $n \geq 1$. For the induction step of this argument, we need to derive that $a_j \notin \hat{Y}_l^{n+1}(b_j)$ from the fact that $a_{j+1} \notin \hat{Y}_l^n(b_{j+1})$. We therefore start by investigating the relationship between a_j and a_{j+1} by expanding the definition of $\hat{Y}_l^{n+1}$ in $a_j \in \hat{Y}_l^{n+1}(b_j)$.

Let θ' denote $\theta[X \mapsto \hat{X}^l, Y \mapsto \hat{Y}_l^n]$. By the definitions of approximation and the semantics of FO-L$_\mu$, $a_j \in \hat{Y}_l^{n+1}(b_j)$ is equivalent to the following:

$$
\begin{aligned}
& a_j \in \hat{Y}_l^{n+1}(b_j) \\
\iff & a_j \in [\![Z]\!]^{\theta'\delta[q \mapsto b_j]} \\
\iff & a_j \in [\![\forall_{p:L}\ \forall_{q':B}\ (p \wedge q \xrightarrow{p} q') \Rightarrow [\cdot]\big((X(q') \wedge q \in F) \vee (Y(q') \wedge q \notin F)\big)]\!]^{\theta'\delta[q \mapsto b_j]} \\
\iff & a_j \in \bigcap_{P \in L, b' \in B} [\![(p \wedge q \xrightarrow{p} q') \Rightarrow [\cdot]\big((X(q') \wedge q \in F) \vee (Y(q') \wedge q \notin F)\big)]\!]^{\theta'\delta[q \mapsto b_j, p \mapsto P, q' \mapsto b']}
\end{aligned}
$$

Expanding the definition of FO-L$_\mu$ semantics further, we find:

$$
\begin{aligned}
& \forall_{P \in L, b' \in B}\ \ell(a_j) = P \wedge b_j \xrightarrow{P} b' \Rightarrow \\
& \qquad a_j \in [\![[\cdot]\big((X(q') \wedge q \in F) \vee (Y(q') \wedge q \notin F)\big)]\!]^{\theta'\delta[q \mapsto b_j, p \mapsto P, q' \mapsto b']} \\
\iff & \forall_{b' \in B}\ b \xrightarrow{\ell(a_j)} b' \Rightarrow \\
& \qquad a_j \in [\![[\cdot]\big((X(q') \wedge q \in F) \vee (Y(q') \wedge q \notin F)\big)]\!]^{\theta'\delta[q \mapsto b_j, p \mapsto \ell(a_j), q' \mapsto b']} \\
\iff & \forall_{b' \in B}\ b \xrightarrow{\ell(a_j)} b' \Rightarrow \forall_{a' \in A}\ a_j \to a' \Rightarrow \\
& \qquad a' \in [\![(X(q') \wedge q \in F) \vee (Y(q') \wedge q \notin F)]\!]^{\theta'\delta[q \mapsto b_j, p \mapsto \ell(a_j), q' \mapsto b']}
\end{aligned}
$$

In particular, we can instantiate the universal quantifiers to derive that if $a_j \in \hat{Y}_l^{n+1}(b_j)$, for $\delta' = \delta[q \mapsto b_j, p \mapsto \ell(a_{j+1}), q' \mapsto b_{j+1}]$:

$$
a_{j+1} \in [\![(X(q') \wedge q \in F) \vee (Y(q') \wedge q \notin F)]\!]^{\theta'\delta'}
$$

Because $a_{j+1} \not\models_{\theta', \delta'} X(q')$ due to (IH), we have an even stronger implication:

$$
\text{for all } j, n \in \mathbb{N}, \text{ if } a_j \in \hat{Y}_l^{n+1}(b_j), \text{ then } a_{j+1} \in \hat{Y}_l^n(b_{j+1}) \text{ and } b_j \notin F. \tag{$*$}
$$

Because $b_0, b_1, \ldots$ is accepting, there must be some $k \in \mathbb{N}$ such that $b_{i+k} \in F$. Let k be such. Note that $b_{i+k} \in F$, so we may derive from ($*$) that $a_{i+k} \notin \hat{Y}_l^1(b_{i+k})$. Furthermore, ($*$) implies that $a_{i+j+1} \notin \hat{Y}_l^n(b_{i+j+1}) \Rightarrow a_{i+j} \notin \hat{Y}_l^{n+1}(b_{i+j})$ for all j, so by induction on k we find $a_i \notin \hat{Y}_l^{k+1}(b_i)$. This is sufficient to conclude our proof, as $k+1$ witnesses the existence of some n for which $a_i \notin \hat{Y}_l^n(b_i)$. □

Lemma 4.2 ($\Leftarrow$). *If $a_0 \not\models_{\theta,\delta} \mathsf{X}$ for all θ and δ such that $\delta(q_0) = b_0$, then there is an accepting run of $\mathcal{B}$ on a_0.*

Proof. In this proof, a and a' are always taken from A, b and b' from B and n, m and k from $\mathbb{N}$. Let θ be a predicate environment and δ a data environment such that $\delta(q_0) = b_0$. Assume that $a_0 \not\models_{\theta,\delta} \mathsf{X}$. Define $\rightsquigarrow \subseteq (A \times B)^2$ as $\langle a, b\rangle \rightsquigarrow \langle a', b'\rangle \triangleq a \to a' \wedge b \xrightarrow{\ell(a)} b'$. We first define a ternary predicate G over $A \times B \times \mathbb{N}$.

$$G(a,b,n) = \begin{cases} b \in F \wedge a \notin [\![\mathsf{X}]\!]^{\theta\delta[q_0 \mapsto b]}, & n = 0 \\ \exists_{a',b'}\ \langle a, b\rangle \rightsquigarrow \langle a', b'\rangle \wedge G(a', b', n-1), & n > 0 \end{cases}$$

This predicate is true for those a, b and n for which there is a sequence $\langle a, b\rangle \rightsquigarrow \ldots \rightsquigarrow \langle a', b'\rangle$ of length n such that $b \in F$ and $a \notin [\![\mathsf{X}]\!]^{\theta\delta[q_0 \mapsto b]}$. We will prove the following for all a and b.

$$a \notin [\![\mathsf{X}]\!]^{\theta\delta[q_0 \mapsto b]} \Rightarrow \exists_k\ G(a, b, k) \wedge k > 0. \tag{$*$}$$

Observe that if this implication holds, we can construct an accepting run of $\mathcal{B}$ on a starting in b, for any a, b for which $a \notin [\![\mathsf{X}]\!]^{\theta\delta[q_0 \mapsto b]}$. In particular we can then do so from $\langle a_0, b_0\rangle$, because $a_0 \not\models_{\theta,\delta} \mathsf{X}$ is equivalent to $a_0 \notin [\![\mathsf{X}]\!]^{\theta\delta[q_0 \mapsto b_0]}$ (the assignment to q_0 in the data environment has no effect). Therefore, proving ($*$) is sufficient to prove the lemma.

Tarski's theorem implies that $a \notin [\![\mathsf{X}]\!]^{\theta\delta[q_0 \mapsto b]}$ is equivalent to $\forall_m\ \exists_n\ a \notin \hat{Y}^n_m(b)$, because $[\![\mathsf{X}]\!]^{\theta\delta[q_0 \mapsto b]} = \hat{X}^m(b) = \hat{Y}^n_m(b)$ for sufficiently large m and n, and because $\hat{Y}^n_m(b) \supseteq \hat{Y}^{n+1}_m(b)$ and $\hat{X}^m(b) \subseteq \hat{X}^{m+1}(b)$ for all m and n.

Now take arbitrary m, n, a and b such that $a \notin \hat{Y}^{n+1}_m(b)$. Filling in the definition of $\hat{Y}^{n+1}_m(b)$ and unfolding the semantics of FO-L$_\mu$ like we did in Lemma 4.1, this is equal to

$$\neg\forall_{b' \in B}\ b \xrightarrow{\ell(a)} b' \Rightarrow \forall_{a' \in A}\ a \to a' \Rightarrow \\ a' \in [\![(X(q') \wedge q \in F) \vee (Y(q') \wedge q \notin F)]\!]^{\theta[X \mapsto \hat{X}^m, Y \mapsto \hat{Y}^n_m]\delta[q \mapsto b, p \mapsto \ell(a), q' \mapsto b']}$$

Using De Morgan's law, we find that there are a' and b' such that $\langle a, b\rangle \rightsquigarrow \langle a', b'\rangle$ and

$$a' \notin [\![(X(q') \wedge q \in F) \vee (Y(q') \wedge q \notin F)]\!]^{\theta'\delta'},$$

where $\theta' = \theta[X \mapsto \hat{X}^m, Y \mapsto \hat{Y}^n_m]$ and $\delta' = \delta[q \mapsto b, p \mapsto \ell(a'), q' \mapsto b']$. Expanding the semantics further, we derive:

$$\begin{aligned} & a' \notin [\![(X(q') \wedge q \in F) \vee (Y(q') \wedge q \notin F)]\!]^{\theta'\delta'} \\ \iff & a' \notin [\![X(q') \wedge q \in F]\!]^{\theta'\delta'} \wedge a' \notin [\![Y(q') \wedge q \notin F]\!]^{\theta'\delta'} \\ \iff & (a' \notin [\![X(q')]\!]^{\theta'\delta'} \vee a' \notin [\![q \in F]\!]^{\theta'\delta'}) \wedge (a' \notin [\![Y(q')]\!]^{\theta'\delta'} \vee a' \notin [\![q \notin F]\!]^{\theta'\delta'}) \\ \iff & (a' \notin \hat{X}^m(b') \vee b \notin F) \wedge (a' \notin \hat{Y}^n_m(b') \vee b \in F) \end{aligned}$$

Summarizing our calculation, we have found

$$\forall_{m,n,a,b}\quad a \notin \hat{Y}^{n+1}_m(b) \Rightarrow \big(\exists_{a',b'}\ \langle a, b\rangle \rightsquigarrow \langle a', b'\rangle \wedge \\ (a' \in \hat{Y}^n_m(b') \Rightarrow b \in F) \wedge (b \in F \Rightarrow a' \notin \hat{X}^m(b'))\big). \tag{$\dagger$}$$

Let m be a number so large that $\hat{X}^m(b) = [\![\mathsf{X}]\!]^{\theta\delta[q_0\mapsto b]}$ for all b. We now show that

$$\forall_{n,a,b}\ a \notin \hat{Y}^n_m(b) \Rightarrow \exists_k\ G(a,b,k) \wedge (k > 0 \vee b \in F). \tag{‡}$$

The proof proceeds by induction on n. Suppose $n = 0$, then $\hat{Y}^n_m(b) = \hat{Y}^0_m(b) = A$, and so the implication holds vacuously.

For $n = i+1$, let a, b be such that $a \notin \hat{Y}^n_m(b)$. Note that this implies that $a \notin \hat{X}^m(b)$. From (†) we can derive the existence of a' and b' such that $\langle a, b\rangle \rightsquigarrow \langle a', b'\rangle$ and

$$(a' \in \hat{Y}^i_m(b') \Rightarrow b \in F) \wedge (b \in F \Rightarrow a' \notin \hat{X}^m(b')).$$

In particular, either $b \in F$, in which case $G(a,b,0)$ is easily seen to hold because $a \notin \hat{X}^m(b)$ which is equivalent to $a \notin [\![\mathsf{X}]\!]^{\theta\delta[q_0\mapsto b]}$, or $a' \notin \hat{Y}^i_m(b')$. In the latter case, use the induction hypothesis to find a k such that $G(a', b', k)$ in which case $G(a, b, k+1)$ also holds.

We now strengthen this result a little by proving

$$\forall_{n,a,b}\ a \notin \hat{Y}^n_m(b) \Rightarrow \exists_k\ G(a,b,k) \wedge k > 0.$$

Note that we only have to consider the case that $b \in F$, because for $b \notin F$ the result follows directly from (‡). So assume $b \in F$. Again, the proof proceeds by induction on n, and again the base case $n = 0$ is trivial. For $n = i + 1$, assume $a \notin \hat{Y}^n_m(b)$ and from (†) obtain a' and b' such that $\langle a, b\rangle \rightsquigarrow \langle a', b'\rangle$ and $a' \notin \hat{X}^m(b')$. Because $a' \notin \hat{X}^m(b')$, also $a' \notin \hat{Y}^k_m(b')$ for large enough k. By (‡) then $G(a', b', k)$ for some $k > 0$. By the definition of G, then $G(a, b, k + 1)$.

Applying the result we just found to a_0, for which we know $a_0 \notin \hat{X}^m(b_0)$, and therefore $a_0 \notin \hat{Y}^k_m(b_0)$ for large enough k, we find that $\exists_k\ G(a_0, b_0, k)$, which concludes our proof. □

Theorem 4.2. *For any $a_0 \in A$ we have that $a_0 \models \mathrm{tr}(\mathcal{B})$ if and only if there is no accepting run of $\mathcal{B}$ on a_0.*

Proof. We start by expanding the semantics of FO-L$_\mu$ in $a_0 \models \mathrm{tr}(\mathcal{B})$.

$$\begin{aligned}
& a_0 \models \forall_{q_0:B}\ q_0 \in B_0 \Rightarrow \mathsf{X} \\
\iff & a_0 \in [\![\forall_{q_0:B}\ q_0 \in B_0 \Rightarrow \mathsf{X}]\!]^{\theta\delta} \\
\iff & a_0 \in \bigcap_{b_0 \in B} [\![q_0 \in B_0 \Rightarrow \mathsf{X}]\!]^{\theta\delta[q_0\mapsto b_0]} \\
\iff & \forall_{b_0\in B}\ a_0 \in [\![q_0 \in B_0]\!]^{\theta\delta[q_0\mapsto b_0]} \Rightarrow a_0 \in [\![\mathsf{X}]\!]^{\theta\delta[q_0\mapsto b_0]} \\
\iff & \forall_{b_0\in B}\ b_0 \in B_0 \Rightarrow a_0 \in [\![\mathsf{X}]\!]^{\theta\delta[q_0\mapsto b_0]} \\
\iff & \forall_{b_0\in B}\ b_0 \in B_0 \Rightarrow a_0 \models_{\theta,\delta[q_0\mapsto b_0]} \mathsf{X}
\end{aligned}$$

By Lemmas 4.1 and 4.2, this holds if and only if there is no accepting run of $\mathcal{B}$ on a_0. In other words, any word accepted by $\mathfrak{A}$ is not accepted by $\mathcal{B}$ and vice versa, which proves the theorem. □

The above proofs, together with Theorem 4.1, lead to the conclusion that we can translate any LTL formula into an equivalent L_μ formula:

Corollary 4.1. *For any LTL formula f and Kripke structure state a, $a \models f$ iff $a \models \mathsf{tr}(\mathcal{B}_f)$.*

4.3.1 Data specifications

We have formulated our translation in such a way that it uses a Büchi automaton directly (B, $\rightarrow$, B_0 and F occur in our formula). In order to use existing techniques [Mad97; DPW08] to be able to automatically check the L_μ formula against a Kripke structure, and also to exploit the structured manner in which we can build a Büchi automaton from an LTL formula, we encode B into a datatype which consists of only Booleans and natural numbers.

Let $\mathsf{A}f$ be an LTL formula and let $\Phi = \mathsf{cl}(f)$. The generalized Büchi automaton $\mathcal{B}_{\mathsf{A}f} = \langle B, \rightarrow, L, F, B_0 \rangle$ can be described using only Booleans (denoted by $\mathbb{B}$) and natural numbers (denoted by $\mathbb{N}$). Recall that a state in $\mathcal{B}_{\mathsf{A}f}$ is a tuple consisting of a subset of Φ and a counter $c \in \{0, \ldots, k-1\}$, with k the number of until operators in f.

We use the fact that, given some mapping $N : \Phi \rightarrow \{1, \ldots, |\Phi|\}$ that enumerates Φ, we can represent 2^Φ with the isomorphic domain $\mathbb{B}^{|\Phi|}$, by representing a subset $G \subseteq \Phi$ by the tuple $\langle b_1, \ldots, b_{|\Phi|} \rangle$ such that $b_{N(g)} = \mathbb{t}$ iff $g \in G$. The counter can be represented by a value from $\mathbb{N}$, and so we may represent states by an element from $\mathbb{B}^{|\Phi|} \times \mathbb{N}$. We assume that $N(f) = 1$.

Define a sort D with $[\![D]\!] = \mathbb{B}^{|\Phi|} \times \mathbb{N}$. Define a function $\mathsf{P} : D \rightarrow 2^{AP}$ that yields for an element $[\![D]\!]$ the set of atomic propositions in the set G that this element represents:

$$\mathsf{P}(\langle b_1, \ldots, b_{|\Phi|}, c \rangle) = \{g \in \Phi \cap AP \mid b_{N(g)} = \mathbb{t}\}.$$

Note that an application of P is a valid FO-L_μ formula, as its interpretation is a subset of AP. Introduce function symbols P, inB, inF, inB_0 and to:

$$\mathsf{sort}(\mathsf{inB}) = D \rightarrow B \qquad \mathsf{sort}(\mathsf{inB}_0) = D \rightarrow B$$
$$\mathsf{sort}(\mathsf{inF}) = D \rightarrow B \qquad \mathsf{sort}(\mathsf{to}) = D \times D \rightarrow B$$

Intuitively, these mappings represent predicates on states from the Büchi automaton. For instance, if a term d of sort D represents some state b in a $\mathcal{B}_{\mathsf{A}f}$, then $\mathsf{inF}(d)$ will have the same truth value as the predicate $b \in F$. The inB mapping is needed to identify terms that represent a valid state in the Büchi automaton (there are subsets of the closure of f that are not in B, see Section 4.1).

Let $\mathfrak{U}, \mathfrak{X}, \mathfrak{N}, \mathfrak{C}$ be sets of indices, and let L and R be mappings from indices to indices, such that for all i, j, k:

- $i \in \mathfrak{U}$, $L(i) = j$ and $R(i) = k$ if and only if $g_i = g_j \mathsf{U} g_k$,
- $i \in \mathfrak{X}$ and $R(i) = j$ if and only if $g_i = \mathsf{X} g_j$,
- $i \in \mathfrak{N}$ and $R(i) = j$ if and only if $g_i = \neg g_j$,
- $i \in \mathfrak{C}$, $L(i) = j$ and $R(i) = k$ if and only if $g_i = g_j \wedge g_k$.

Let $U : \mathbb{N} \to \mathfrak{U}$ enumerate $\mathfrak{U}$ in an arbitrary way. The definitions of the mappings inB, inF, inB_0 and to are given in Figure 4.1, and can easily be seen to correspond with the definitions in Section 4.1.

Clearly, this specification is linear in the number of subformulas of f. Note that our definition of size does not take into account the space needed to represent an identifier, as for all practical intents and purposes it can be seen as a constant.

We use the fact that $\langle b_0, \ldots, b_n, c\rangle \xrightarrow{p} q'$ implies $\mathsf{to}(\langle b_0, \ldots, b_n, c\rangle, q')$ and $p = \mathsf{P}(\langle b_0, \ldots, b_n, c\rangle) = \{a \in AP \mid b_{I(a)}\}$, where I maps a subformula ψ_i to its index i. The L$_\mu$ formula in definition 4.1 can be rewritten to the following formula using only quantifiers over D and using the previously defined mappings (*i.e.*, $q \in F$ is replaced by $\mathsf{inF}(q)$, $q \xrightarrow{p} q'$ by $\mathsf{to}(\langle b_0, \ldots, b_n, c\rangle, q')$ while replacing all occurrences of p by $\mathsf{P}(q)$, etc.).

$$
\begin{aligned}
&\forall_{q:D}\ \mathsf{inB}_0(q) \Rightarrow \\
&\quad \mu X(q : D = q)\,. \\
&\quad \nu Y(q : D = q)\,. \\
&\quad \forall_{q':D}\ (\mathsf{inB}(q') \wedge \mathsf{P}(q) \wedge \mathsf{to}(q, q')) \Rightarrow [\cdot]\big((X(q') \wedge \mathsf{inF}(q)) \vee (Y(q') \wedge \neg\mathsf{inF}(q))\big)
\end{aligned}
$$

Due to the expansion of $\mathsf{P}(q)$ to $\bigwedge_{a\in AP}(b_{I(a)} \Leftrightarrow a)$, the formula may grow to a size linear in $|f|$.

4.3.2 Complexity

We have given a translation from an LTL formula to a first-order L$_\mu$ formula over a data structure. We now show that model checking the resulting formula against a

$$
\begin{aligned}
\mathsf{inB}(\langle b_0, \ldots, b_n, c\rangle) = {} & \bigwedge_{i\in\mathfrak{N}} b_i \Leftrightarrow \neg b_{R(i)} \\
& \bigwedge_{i\in\mathfrak{C}} b_i \Leftrightarrow (b_{L(i)} \wedge b_{R(i)}) \wedge \\
& \bigwedge_{i\in\mathfrak{U}} (b_{R(i)} \Rightarrow b_i) \wedge (b_i \Rightarrow (b_{L(i)} \vee b_{R(i)})) \wedge \\
\mathsf{inB}_0(\langle b_0, \ldots, b_n, c\rangle) = {} & \mathsf{inB}(\langle b_0, \ldots, b_n, c\rangle) \wedge \neg b_0 \\
\mathsf{inF}(\langle b_0, \ldots, b_n, c\rangle) = {} & \mathsf{inB}(\langle b_0, \ldots, b_n, c\rangle) \wedge (\neg b_{U(c)} \vee b_{R(U(c))}) \\
\mathsf{to}(\langle b_0, \ldots, b_n, c\rangle, {} & \\
\quad \langle b'_0, \ldots, b'_n, c'\rangle) = {} & \bigwedge_{i\in\mathfrak{X}} (b_i \Leftrightarrow b'_{R(i)}) \wedge \\
& \bigwedge_{i\in\mathfrak{U}} (b_i \Leftrightarrow (b_{R(i)} \vee (b_{L(i)} \wedge b'_i))) \wedge \\
& \mathsf{inF}(\langle b_0, \ldots, b_n, c\rangle) \Leftrightarrow (c' = (c+1) \bmod |\mathfrak{U}|)
\end{aligned}
$$

Figure 4.1: Encoding of a generalized Büchi automaton in simple data types.

Kripke structure has the same time complexity as other LTL model checking methods. In particular, we establish the same worst-case time complexity as Bhat et al. [BC96].

Theorem 4.3. *Let ψ be the L_μ formula that is the result of the above translation for an LTL formula $\mathsf{A}f$. Verifying ψ on a Kripke structure $\mathfrak{A}$ can be done in $O(|\mathfrak{A}|)\cdot 2^{O(|f|)}$ time.*

Proof. Note that $[\![D]\!]$ is finite. Let $M = |[\![D]\!]|$ and let $N = |\mathfrak{A}|$. The predicate transformer $\mathbf{T}$, defined in the semantics of FO-L$_\mu$ as a function of type $([\![D]\!] \to 2^A) \to ([\![D]\!] \to 2^A)$, can also be interpreted as being of type $(2^A)^{[\![D]\!]} \to (2^A)^{[\![D]\!]}$. Using the correspondence between power sets and boolean vectors, we transform the type further to $(\mathbb{B}^{A\times[\![D]\!]}) \to (\mathbb{B}^{A\times[\![D]\!]})$. As such, the fixpoint of $\mathbf{T}$ can be computed as the simultaneous fixpoint of $M\cdot N$ separate functions of type $\mathbb{B}^{A\times[\![D]\!]} \to \mathbb{B}$ (see, *e.g.*, [AN01]).

We can adapt the semantics of FO-L$_\mu$ in the obvious way to deal with this change. For instance, assume that $A = \{a_1,\ldots,a_N\}$ and $[\![D]\!] = \{d_1,\ldots,d_M\}$, and let index function $\mathsf{i}\colon [\![D]\!]\cup A \to \mathbb{N}$ be such that $\mathsf{i}(d_i) = i$ and $\mathsf{i}(a_i) = i$ for all i. If $\bar{X} = \langle X_1,\ldots,X_{M\cdot N}\rangle$ is a vector of Booleans, let $\bar{X}^i$ denote the vector $\langle X_{(i-1)N+1},\ldots,X_{iN}\rangle$. We can interpret the vector $\bar{X}^i$ as a subset of A again by defining $a \in \bar{X}^i(d) = X_{(i-1)N+\mathsf{i}(a)}$. The definitions for the fixpoint operator and the predicate transformer then become:

$$
\begin{aligned}
[\![\mu X(d : D = e)\,.\,\varphi]\!]^{\theta\delta} &\triangleq \langle(\mathbf{lfp}\,\mathbf{T})_{N\cdot(\mathsf{i}([\![e]\!]^\delta)-1)},\ldots,(\mathbf{lfp}\,\mathbf{T})_{N\cdot\mathsf{i}([\![e]\!]^\delta)-1}\rangle\\
\mathbf{T}(\bar{X}:\mathbb{B}^{A\times[\![D]\!]}) &\triangleq \langle a_1 \in [\![\varphi]\!]^{\theta[X\mapsto\bar{X}^1]\delta[d\mapsto d_1]},\ldots,a_N \in [\![\varphi]\!]^{\theta[X\mapsto\bar{X}^1]\delta[d\mapsto d_1]},\\
&\qquad\vdots\\
&\quad a_1 \in [\![\varphi]\!]^{\theta[X\mapsto\bar{X}^M]\delta[d\mapsto d_M]},\ldots,a_N \in [\![\varphi]\!]^{\theta[X\mapsto\bar{X}^M]\delta[d\mapsto d_M]}\rangle
\end{aligned}
$$

Using this transformation, and using the techniques (and notation) from [AN01] to transform a vectorial fixpoint formula into a system of fixpoint equations, the interpretation of $\mathrm{tr}(\mathcal{B})$ on $\mathfrak{A}$ can be computed as the solutions of $\langle T_{a_1},\ldots,T_{a_N}\rangle$ in the following equation system:

$$
\begin{aligned}
T_{a_1} &\overset{\mu}{=} \bigwedge_{b\in B_0} X_{a_1,b}\\
&\vdots\\
T_{a_N} &\overset{\mu}{=} \bigwedge_{b\in B_0} X_{a_N,b}\\
X_{a_1,b_1} &\overset{\mu}{=} Y_{a_1,b_1}\\
&\vdots\\
X_{a_N,b_M} &\overset{\mu}{=} Y_{a_N,b_M}\\
Y_{a_1,b_1} &\overset{\nu}{=} a_1 \in [\![\mathsf{Z}]\!]^{\theta[X\mapsto\langle X_{a_1,b_1},\ldots,X_{a_1,b_M}\rangle,Y\mapsto\langle Y_{a_1,b_1},\ldots,Y_{a_1,b_M}\rangle]\delta[q\mapsto b_1]}\\
&\vdots\\
Y_{a_N,b_M} &\overset{\nu}{=} a_N \in [\![\mathsf{Z}]\!]^{\theta[X\mapsto\langle X_{a_N,b_1},\ldots,X_{a_N,b_M}\rangle,Y\mapsto\langle Y_{a_N,b_1},\ldots,Y_{a_N,b_M}\rangle]\delta[q\mapsto b_M]}
\end{aligned}
$$

If we inspect Z more closely, we see that every disjunction (either true disjunction or implication) in that formula has one disjunct that does not contain X or Y. In the expression $[\![\mathsf{Z}]\!]^{\theta[\ldots]\delta[\ldots]}$ above, those disjuncts can therefore be eliminated, as their truth value is constant.

Take the right hand side of Y_{a_i,b_j}, for example. By expanding the definition of semantics of FO-L$_\mu$, we find that it is equivalent to:

$$\begin{aligned} & a_i \in [\![\mathsf{Z}]\!]^{\theta[X \mapsto \langle X_{a_i,b_1},\ldots,X_{a_i,b_M}\rangle, Y \mapsto \langle Y_{a_i,b_1},\ldots,Y_{a_N,b_M}\rangle]\delta[q \mapsto b_j]} \\ \iff & \forall_{b'}\ b_j \xrightarrow{\ell(a_i)} b' \Rightarrow \forall_{a' \in A}\ a_i \to a' \Rightarrow ((X_{a',b'} \wedge b' \in F) \vee (Y_{a',b'} \wedge b' \notin F)) \end{aligned}$$

The latter can easily seen to be equivalent to $\bigwedge_{\langle a',b'\rangle \in S} X_{a',b'} \wedge \bigwedge_{\langle a',b'\rangle \in S'} Y_{a',b'}$ for some finite sets S and S' containing those tuples $\langle a', b'\rangle$ such that $a_i \to a'$ and $b_j \xrightarrow{\ell(a_i)} b'$, and such that S contains tuples for which $b' \in F$ and S' those for which $b' \notin F$. Moreover, the number of outgoing edges from a_i and b_j bounds both the size of these sets and the time needed to compute them (by using an appropriate data structure, like edge lists).

The size of this equation system is therefore $\mathcal{O}(|\mathfrak{A}| \cdot |\mathcal{B}|)$, and the time needed to compute the equation system is linear in its size. The time needed to solve the equation system is also linear, because the system is *disjunctive/conjunctive straight*, that is, the right hand sides of two mutually dependent variables are always both purely disjunctive, or purely conjunctive [KKV01; GK05]. Because $\mathcal{O}(|\mathcal{B}|) = 2^{\mathcal{O}(|f|)}$ because of the powerset construction, the theorem follows. □

4.4 Translation of CTL*

Assume that tr generates data structures with fresh names every time it is used. We define a translation tr$'$ that translates a CTL* formula into a modal L$_\mu$ formula. The intuition is that every CTL* formula can be seen as a CTL structure containing linear time fragments. Nested linear time fragments form a problem, because we cannot use the LTL translation on them directly. Instead, we take the innermost fragment (which must be LTL), and translate that using our translation function. We then replace this fragment in the original by a placeholder and repeat the procedure. In the translated fragments, the placeholders are again substituted for the translated counterparts of the linear time fragment they represent.

Definition 4.2. *Two formulae (either L$_\mu$ or CTL*) φ and ψ are equivalent, denoted $\varphi \approx \psi$, when for every Kripke structure $\mathfrak{A}$ we have $\mathfrak{A}, a \models \varphi$ if and only if $\mathfrak{A}, a \models \psi$ for all $a \in A$.*

If φ and ψ are both L$_\mu$ formula, then we define, given a predicate environment θ and a data environment δ, $\varphi \approx_{\theta\delta} \psi$ if and only if for every Kripke structure $\mathfrak{A}$ we have $\mathfrak{A}, a \models_{\theta\delta} \varphi$ iff $\mathfrak{A}, a \models_{\theta\delta} \psi$ for all $a \in A$.

We introduce a set AP' that is disjoint from AP, which contains for every CTL* formula f an atomic proposition a_f. A function R is defined that takes a CTL* path

formula, and returns a CTL* formula in which all top level $\mathsf{A}g$ subformulae are replaced by a_g:

$$\begin{array}{llll} R(\mathsf{A}f) & \triangleq a_f & R(\mathsf{X}f) & \triangleq \mathsf{X}R(f) \\ R(f \mathsf{U} g) & \triangleq R(f) \mathsf{U} R(g) & R(f \vee g) & \triangleq R(f) \vee R(g) \\ R(\neg f) & \triangleq \neg R(f) & R(a) & \triangleq a \end{array}$$

The $\bar{R}$ function takes a L$_\mu$ formula and syntactically replaces every $a_g \in AP'$ that occurs in it by $\mathsf{tr}'(g)$. What is left is to define tr', which takes a CTL* formula and yields a L$_\mu$ formula:

$$\begin{array}{lll} \mathsf{tr}'(\mathsf{A}f) & = \begin{cases} \mathsf{tr}(\mathcal{B}_f), & \neg\exists_g\, \mathsf{A}g \in f \\ \bar{R}(\mathsf{tr}'(\mathsf{A}R(f))), & \text{otherwise} \end{cases} \\ \mathsf{tr}'(f \vee g) & = \mathsf{tr}'(f) \vee \mathsf{tr}'(g) \\ \mathsf{tr}'(\neg f) & = \neg \mathsf{tr}'(f) \\ \mathsf{tr}'(a) & = a \end{array}$$

It is immediately apparent from this definition that the size of the resulting formula is again linear in the size of the original.

Lemma 4.3. *Let φ, χ and ψ be a first-order modal L$_\mu$ formulae, and let $\varphi[\chi/\psi]$ denote φ in which all occurrences of ψ are syntactically replaced by χ. If, for some predicate environment θ and data environment δ, $\chi \approx_{\theta\delta} \psi$, then $\varphi \approx_{\theta\delta} \varphi[\chi/\psi]$, provided that χ nor ψ contain free variable names (either fixpoint or data) that are bound in φ.*

Proof. Fix formulae φ, χ and ψ. The proof goes by structural induction on φ. The induction hypothesis states that if $\varphi' \in \varphi$ and $\varphi' \neq \varphi$, and also $\chi \approx_{\theta\delta} \psi$ for some θ and δ, then $\varphi'[\chi/\psi] \approx_{\theta\delta} \varphi'$.

The base cases are trivial: either ψ does not occur in φ, in which case $\varphi[\chi/\psi] = \varphi$, or $\varphi = \psi$.

The other cases are also very straightforward. We demonstrate the case that $\varphi = \mu X(d : D = e)\ .\ \varphi'$. Suppose $\chi \approx_{\theta\delta} \psi$ for some ρ and ε. Then $[\![\varphi[\chi/\psi]]\!]^{\theta\delta} = (\mu\mathbf{T})([\![e]\!]^\delta)$, where

$$\mathbf{T}(F : \mathbb{X}) \triangleq \lambda v_1, \ldots, v_n : \mathsf{sort}(X).[\![\varphi'[\chi/\psi]]\!]^{\theta[X\mapsto F]\delta[d_1\mapsto v_1,\ldots,d_n\mapsto v_n]}.$$

Because X and $d_1, \ldots, d_n$ do not occur freely in χ and ψ,

$$[\![\chi]\!]^{\theta[X\mapsto F]\delta[d_1\mapsto v_1,\ldots,d_n\mapsto v_n]} = [\![\chi]\!]^{\theta\delta} = [\![\psi]\!]^{\theta\delta} = [\![\psi]\!]^{\theta[X\mapsto F]\delta[d_1\mapsto v_1,\ldots,d_n\mapsto v_n]}.$$

We may substitute $[\![\varphi']\!]^{\theta[X\mapsto F]\delta[d_1\mapsto v_1,\ldots,d_n\mapsto v_n]}$ for $[\![\varphi'[\chi/\psi]]\!]^{\theta[X\mapsto F]\delta[d_1\mapsto v_1,\ldots,d_n\mapsto v_n]}$ in the above using the induction hypothesis. Then $(\mu\Phi_d)([\![e]\!]^\delta) = [\![\varphi]\!]^{\theta\delta}$, which implies that $\varphi \approx_{\theta\delta} \varphi[\chi/\psi]$. □

Lemma 4.4. *Let f be a CTL* formula. If $g \sqsubseteq f$, and $h \approx g$, then $f \approx f[h/g]$, where $f[h/g]$ is f in which all occurrences of g are syntactically replaced by h.*

Proof. The proof is again by induction on the structure of the formula. □

Theorem 4.4. *If f is a CTL* formula, then $f \approx \mathrm{tr}'(f)$.*

Proof. Fix a Kripke structure $\mathfrak{A} = \langle A, AP, \to, \ell \rangle$. Our goal is to prove that $\mathfrak{A} \models f$ if and only if $\mathfrak{A} \models \mathrm{tr}'(f)$. We introduce a Kripke structure $\mathfrak{A}' = \langle A, \to, AP \cup AP', \ell' \rangle$, where ℓ' is such that for all $a \in A$, $p \in AP$ and $p_f \in AP'$ we have $p \in \ell'(a) \Leftrightarrow p \in \ell(s)$ and $p_f \in \ell'(a) \Leftrightarrow p \models \mathsf{A}f$. In words, $\mathfrak{A}'$ is the extension of $\mathfrak{A}$ such that states in which a CTL* formula $\mathsf{A}f$ holds are labelled with p_f. Note that this extension is conservative in the sense that for all $a \in A$ and CTL* formulae f over AP we have $\mathfrak{A}, a \models f$ iff $\mathfrak{A}', a \models f$.

We prove for all $a \in A$ that $\mathfrak{A}', a \models f$ if and only if $\mathfrak{A}', a \models \mathrm{tr}'(f)$ by induction on the structure of f. There are two base cases:

Case $f \in AP$**.** It follows trivially from the semantics of CTL* and the modal L_μ that $f \approx \mathrm{tr}'(f)$ in this case.

Case $f = \mathsf{A}g$ **and** $\neg\exists_h\, \mathsf{A}h \sqsubset g$**.** In this case, f is an LTL formula and is translated using tr, which was proven to satisfy the desired property in the previous section.

The base of our induction is therefore sound. For the inductive step, we distinguish three cases.

Case $f = \neg g$**.** It follows directly from the semantics of CTL* and the semantics of FO-L_μ that if $g \approx \mathrm{tr}'(g)$, then also $\neg g \approx \neg\mathrm{tr}'(g)$.

Case $f = g \vee h$**.** Again $f \approx \mathrm{tr}'(f)$ follows directly from the semantics of CTL* and FO-L_μ.

Case $f = \mathsf{A}g$ **and** $\exists_h\, \mathsf{A}h \in g$**.** Note that $\mathsf{A}g \approx \mathsf{A}R(g)$, and $\mathsf{A}R(g)$ contains only a single A operator. Then $\mathrm{tr}'(\mathsf{A}R(g)) = \mathrm{tr}(\mathcal{B}_{R(g)})$, so we know from the previous section that $\mathrm{tr}'(\mathsf{A}R(g)) \approx \mathsf{A}R(g)$. But then also $\mathrm{tr}'(\mathsf{A}R(g)) \approx \mathsf{A}g$, by lemma 4.4 and transitivity of $\approx$. This translation, $\mathrm{tr}'(\mathsf{A}R(g))$, may contain some atomic predicate a_h that we introduced for a subformula $\mathsf{A}h$ of g. By the induction hypothesis, we have that $\mathrm{tr}'(\mathsf{A}h) \approx \mathsf{A}h$, and because by definition $a_h \approx \mathsf{A}h$, we also have $a_h \approx \mathrm{tr}'(\mathsf{A}h)$. We can by lemma 4.3 syntactically replace every a_h by $\mathrm{tr}'(\mathsf{A}h)$ by applying $\bar{R}$, and obtain an equivalent formula. It follows that $\bar{R}(\mathrm{tr}'(\mathsf{A}R(g))) \approx f$.

We now know that $\mathfrak{A}', a \models f$ if and only if $\mathfrak{A}', a \models \mathrm{tr}'(f)$. However, f nor $\mathrm{tr}'(f)$ contain atomic predicates from AP', and therefore both would be evaluated the same on $\mathfrak{A}$. Therefore also $\mathfrak{A}, a \models f$ if and only if $\mathfrak{A}, a \models \mathrm{tr}'(f)$, which concludes our proof. □

We note that the above translation may be inefficient in practice, as it does not take advantage of the fact that for CTL formulae there is a much more straightforward translation to the modal L_μ. Without any change, tr' may therefore generate translations for CTL formulae that require exponential time to solve. However, it is easy to adapt tr' to include case distinctions for CTL operators, following the translation in, *e.g.*, [CGP99]. The adapted translation procedure then yields translations for CTL formulae that can be solved in linear time, and will provide more efficient translations for certain types of CTL* formulae.

Theorem 4.5. *The complexity of checking a CTL* formula f through the first-order modal L_μ on a Kripke structure $\mathfrak{A}$ is $O(|\mathfrak{A}|) \cdot 2^{O(|f|)}$.*

Proof. This can be seen by looking at the structure of $\mathrm{tr}'(f)$. Because of the way it was constructed, every innermost subformula $\varphi \in \mathrm{tr}'(f)$ that was generated by tr, is again a closed L_μ formula. We can introduce a fresh fixpoint variable X, and evaluate $[\![\mathrm{tr}'(f)]\!]^{\theta[X \mapsto F]\delta}$ for some θ and δ, where $F = [\![\varphi]\!]^{\theta\delta}$. As f is closed, this yields the same result as evaluating the expression using any other predicate environment. Note that we can calculate $[\![\varphi]\!]^{\theta\delta}$ in $O(|M|) \cdot 2^{O(|g|)}$ time, where g is the subformula of f to which φ corresponds. By lemma 4.3, we may substitute X for φ and repeat this procedure. When no more substitutions can be made, the remainder can be solved in $O(|\mathfrak{A}|)$ time, as the formula contains no fixpoints. Since the sum of the lengths of all g that were substituted is less than or equal to $|f|$, the entire operation is $O(|\mathfrak{A}|) \cdot 2^{O(|f|)}$. □

4.5 Conclusion

We presented a translation from CTL* formulae to first-order modal L_μ formulae. By using this specific variant of the L_μ, we are able to give a translation that is succinct, but that does not introduce performance penalties when checking the formula against a Kripke structure. Indeed, the time complexity of CTL* model checking via the first-order modal L_μ is no worse than that of CTL* model checking using the best existing direct method.

Chapter 5

Games

Parity games [EJ91; McN93; Zie98] are played by two players (represented by $\Diamond$ and $\Box$) on a directed graph in which every vertex is owned by one of the players, and vertices are assigned a priority. From each vertex, a game can be played by moving a single token along the edges in the graph; the choice where to move next is dictated by the player owning the vertex on which the token currently resides. The (infinite) path through the graph that the token visits while this game is played determines who wins the game. An important property of parity games is that from any vertex, exactly one of the players can always win every game started from that vertex by playing according to a fixed strategy. This player is designated the winner of that vertex. Partitioning the graph in vertices that are won by player $\Diamond$ and those won by player $\Box$ is referred to as *solving* the parity game.

It is well known that the parity game framework is closely related to the notion of (modal) fixpoint logic [EJ91]. In fact, modal μ-calculus model checking is polynomially reducible to parity game solving, and *vice versa*. In practice this means that the parity game framework can be used to solve practical verification and synthesis problems, see [Grä02; AVW03]. We examine the relation between fixpoint logic and games in closer detail in Chapter 7, but for now we will only say that for various fixpoint logics, the solution to a formula in that logic can be established by looking at the winner of a single node in a parity game that corresponds to the formula.

Despite the apparent simplicity of the problem of solving parity games, its precise complexity is still open: the problem is known to be in NP$\cap$coNP, and more specifically in UP $\cap$ coUP [Jur98], suggesting there just might exist a polynomial time algorithm. Indeed, non-trivial classes of parity games have been identified that admit polynomial time solving algorithms, see *e.g.* [Ber+06; Obd07].

In the past two decades, several advanced algorithms for solving parity games have been designed. These include algorithms exponential in the number of priorites, such as Jurdziński's *small progress measures* algorithm [Jur00] and Schewe's *bigstep* algorithm [Sch07], as well as the sub-exponential algorithm due to Jurdziński *et al.* [JPZ06] Such algorithms have been implemented and are used in practice to solve verification problems [FL09; Cra+13; KP14]. An interesting observation here is that the simplest, exponential algorithm known as the *recursive algorithm*, performs best in most practical scenarios.

Orthogonally to the algorithmic improvements, heuristics have been devised that

may speed up solving parity games that occur in practice, see, *e.g.*, [PW08; FL09; GW13; Ver13]. Such heuristics work particularly well for verification problems, which give rise to games with only few different priorities.

In this chapter, we investigate equivalence relations that approximate the solution to a parity game, following, *e.g.*, Fritz and Wilke's study of *delayed simulation* [Fri05; FW06]. The idea is to recast the solving problem as the problem of deciding *winner equivalence* between vertices: two vertices in a parity game are equivalent whenever they are won by the same player.

One reason to investigate equivalences on parity games is that they provide a new heuristic to speed up solving parity games. Finding equivalence relations that refine winner equivalence and that are decidable in polynomial time yields a preprocessing step that can be used to reduce the size of a parity game prior to solving.

There are also other important reasons to look at equivalence relations on parity games. Janin notes that because parity games give semantics to many automata-based formalisms, proofs on those formalisms often consist of finding a winning strategy in one game, given a winning strategy in another game [Jan05]. Being able to prove the existence of such a strategy by relating nodes in those games using a relation that is known to refine winner equivalence can then simplify the proof. The same observation is made by Kissig and Venema [KV09], who define a notion of bisimulation on games to 'streamline', as they put it, the proof of the main theorem in that paper. Also in the analysis of the descriptive complexity of parity games, Dawar and Grädel use the notion of strong bisimulation on parity games [DG08].

Because parity games give semantics to a number of formalisms, including fixpoint logic, there is yet another good reason to investigate equivalence relations on parity games. Model checkers may generate a parity game from a fixpoint formula in order to calculate its solution. If equivalences on these parity games can be lifted to the syntactic level (of the fixpoint logic), then perhaps smaller parity games can be generated from these formulas by not generating more than one vertex per equivalence class in the parity game. The results in this chapter confirm that on the parity games themselves, great reductions can be achieved. If therefore these reductions can be approximated by syntactic manipulations of the fixpoint logic from which the games are generated, then we arrive at another heuristic that may simplify fixpoint logic formulas. This strategy is similar to that of the techniques described in, *e.g.*, [OWW09; KRW12; Cra+ed], in which fixpoint formulas are transformed syntactically to reduce the size of the corresponding parity games.

From a practical viewpoint, we are particularly interested in those simulation and equivalence relations that strike a favourable balance between their power to compress the game graph and their computational complexity. Stuttering bisimulation [BCG88] for Kripke Structures is among a select number of candidates worth considering, with an $\mathcal{O}(nm)$ time complexity (n being the number of vertices and m the number of edges).

In [CKW11] we showed that stuttering bisimilarity indeed refines winner equivalence, and that—at the time of writing—it could help speeding up solving parity games. A weak point of stuttering bisimilarity is that it is inept when faced with alternations

between players along the possible plays: it cannot relate vertices belonging to different players. Turn-based games, controller synthesis problems *e.g.* [AVW03], and constructs such as $\Box\Diamond\phi$ and $\Diamond\Box\phi$ in μ-calculus verification, all give rise to such parity games.

In this chapter we define a relation that weakens stuttering bisimulation, so that it is able to relate vertices that belong to different players. We dub this relation *governed stuttering bisimulation*, because it takes into account the governing power of the players during the play; rather than relating nodes based on graph reachability criterions, as is done in stuttering bisimulation, we relate nodes based on which parts of the graph each *player* can reach, regardless of the opponent's moves.

We present a small lattice of equivalence relations on parity games which all enjoy the property of refining winner equivalence. Governed stuttering bisimilarity is shown to be the coarsest of these relations, and for governed bisimilarity itself we give the proof that it is an equivalence relation, and that it refines winner equivalence. The chapter is concluded with a brief discussion of the experiments that have been done with stuttering bisimilarity and governed stuttering bisimilarity on parity games.

5.1 Parity games

A parity game is a two-player graph game, played by two players on a directed graph. The game is formally defined as follows.

Definition 5.1. *A* parity game *is a directed graph* $(V, \to, \Pi, \Omega)$*, where*

- V *is a finite set of vertices,*
- $\to \subseteq V \times V$ *is a total edge relation (*i.e.*, for each* $v \in V$ *there is at least one* $w \in V$ *such that* $(v, w) \in \to$*),*
- $\Pi : V \to \{\Diamond, \Box\}$ *is a function assigning vertices to players.*
- $\Omega : V \to \mathbb{N}$ *is a priority function that assigns priorities to vertices,*

We note that in parity game literature, it is more common to find definitions that have two disjunct sets of vertices—one for each player—instead of a single set of vertices and the function Π that partitions it. The two definitions are obviously equivalent, however, and for the purpose of this chapter it is more convenient to be able to treat a parity game like a regular Kripke structure, in which states are labelled with players and priorities.

If i is a player, then $\neg i$ denotes the opponent of i, *i.e.*, $\neg\Diamond = \Box$ and $\neg\Box = \Diamond$. We will use the same notation for paths in parity games as for paths in Kripke structures.

Winning A game starting in a vertex $v \in V$ is played by placing a token on v, and then moving the token along the edges in the graph. Moves are taken indefinitely according to the following simple rule: if the token is on some vertex v, player $\Pi(v)$ moves the token to some vertex w such that $v \to w$. The result is an infinite path p in the game graph, sometimes called a *play*. The *parity* of the lowest priority that occurs infinitely often on p defines the *winner* of the path. If this priority is even, then player $\Diamond$ wins, otherwise player $\Box$ wins.

Strategies A *strategy* for player i is a partial function $s : V^* \to V$, that is defined exactly for those paths ending in a vertex owned by player i and determines the next vertex to be played onto. The set of strategies for player i in a game G is denoted $\mathbb{S}^*_{G,i}$, or simply $\mathbb{S}^*_i$ if G is clear from the context. If a strategy yields the same vertex for all paths that end in the same vertex, then the strategy is said to be *memoryless*. The set of memoryless strategies for player i in a game G is denoted $\mathbb{S}_{G,i}$, abbreviated to $\mathbb{S}_i$ when G is clear from the context. A memoryless strategy is usually given as a partial function $s : V \to V$.

A path p of length n is *consistent* with a strategy $s \in \mathbb{S}^*_i$, denoted $s \Vdash p$, if and only if for all $1 \leq j < n$ it is the case that if s is defined for $p_1 \dots p_j$, then $p_{j+1} = s(p_1 \dots p_j)$. The definition of consistency is extended to infinite paths (which we will often call *plays*) in the obvious manner. A strategy $s \in \mathbb{S}^*_i$ is said to be a *winning strategy* from a vertex v if and only if i is the winner of every play from v that is consistent with s. A vertex is won by i if i has a winning strategy from that vertex. Parity games are memoryless determined [EJ91], *i.e.*, each vertex in the game is won by exactly one player, and it suffices to play a memoryless strategy.

Relations Let R be a binary relation over a set V. For $v, w \in V$ we write $v\ R\ w$ for $(v, w) \in R$. For an equivalence relation R, and vertex $v \in V$ we define $[v]_R$, the equivalence class of v under R, as $\{v' \in V \mid v\ R\ v'\}$. The set of equivalence classes of V under R is denoted $V_{/R}$.

Arrow notation Throughout this chapter we will be using notation that will simplify reasoning about parity games. Let $\langle V, \to, \Omega, \Pi \rangle$ be a parity game, let $U, T \subseteq V, \mathcal{U} \subseteq 2^V$ and let s be a strategy (for either player).

Given a memoryless strategy s, we introduce a single-step relation $\xrightarrow{s} \subseteq \to$ that contains only those edges allowed by s:

$$v \xrightarrow{s} u \triangleq \begin{cases} v \to u \wedge s(v) = u, & s(v) \text{ is defined} \\ v \to u, & \text{otherwise} \end{cases}$$

We introduce special notation to express which parts of the graph can be reached from a certain node. We use $v \xmapsto{U} T$ to denote that there is a finite path $v \to \dots \to t$ such that $t \in T$, and $u \in U$ for every u on the path unequal to v or t. Conversely, $v \xmapsto{U}$ denotes the existence of an infinite path starting in v on which $u \in U$ for every u unequal to v.

We extend this notation to restrict this reachability analysis to plays that can be enforced by a specific player. We say that strategy s forces the play from v to T via U, denoted $v \xmapsto{s,U} T$, if and only if for all plays p starting in v such that $s \Vdash p$, there exists $n > 0$ such that $p_n \in T$ and $p_i \in U$ for all $0 < i < n$. Similarly, strategy s forces the play to diverge in U from v, denoted $v \xmapsto{s,U}$, if and only if for all such plays p, $p_i \in U$ for all $i > 0$.

Finally, if we are not interested in a particular strategy, but only in the *existence* of a strategy for a player i via which certain parts of the graph are reachable from v, we replace s by i in our notation to denote an existential quantification over memoryless strategies (the requirement that the strategy be memoryless will simplify our

proofs, but it is not difficult to prove that quantifying over all strategies would yield an equivalent definition):

$$v \overset{i,U}{\longmapsto} T \triangleq \exists_{s \in \mathbb{S}_i}\ v \overset{s,U}{\longmapsto} T \qquad\qquad v \overset{i,U}{\longmapsto} \triangleq \exists_{s \in \mathbb{S}_i}\ v \overset{s,U}{\longmapsto}$$

If in the above, $U = V$, then U is omitted. If $U = [v]_R$ for some equivalence relation R, then we sometimes write R instead of U.

The complement of these relations is denoted by a slashed version of the corresponding arrow, *e.g.*, $\neg v \overset{i,R}{\longmapsto} u$ can be written $v \overset{i,R}{\not\longmapsto} u$. We extend the transition relation of the parity game to sets and to sets of sets in the usual way, *i.e.*, if T is a set of vertices, and $\mathcal{U}$ is a set of vertex sets, then

$$v \to T \triangleq \exists_{u \in T}\ v \to u \qquad\qquad v \to \mathcal{U} \triangleq v \to \bigcup \mathcal{U}$$

All other arrow notation is extended in the same way; if a set of sets $\mathcal{U}$ is given as a parameter, it is interpreted as the union of $\mathcal{U}$.

5.2 Properties of parity games

In this section, we present some general lemmas about parity games that will be used in subsequent proofs. In what follows, fix a parity game $\langle V, \to, \Pi, \Omega \rangle$, and let $v \in V$, i a player, and $U, T, T' \subseteq V$.

One of the most basic properties we expect to hold is that a player can force the play towards some given set of vertices, or otherwise her opponent can force the play to the complement of that set.

Lemma 5.1. $v \overset{i,U}{\longmapsto} T \vee v \overset{\neg i,U}{\longmapsto} V \setminus T$.

Proof. We prove the equivalent $v \overset{i,U}{\not\longmapsto} T \Rightarrow v \overset{\neg i,U}{\longmapsto} V \setminus T$. Assume that $v \overset{i,U}{\not\longmapsto} T$. We show that $v \overset{\neg i,U}{\longmapsto} V \setminus T$. We distinguish on the player of v.

- $\Pi(v) = i$. As $v \overset{i,U}{\not\longmapsto} T$, we know $\forall_u\ v \to u \Rightarrow u \notin T$, hence also $\forall_u\ v \to u \Rightarrow u \in V \setminus T$, so $v \overset{\neg i,U}{\longmapsto} V \setminus T$.
- $\Pi(v) \neq i$. As $v \overset{i,U}{\not\longmapsto} T$, and the parity game is total, we know $\exists_u\ v \to u \wedge u \notin T$. Let u be such, and define $s : \mathbb{S}_{\neg i}$ such that $s(v) = u$. Then s is a witness for $v \overset{\neg i,U}{\longmapsto} V \setminus T$. □

In a similar train of thought, we expect that if from a single vertex, each player can force play towards some target set, then the players' target sets must contain related vertices.

Lemma 5.2. $v \overset{i,U}{\longmapsto} T \wedge v \overset{\neg i,U}{\longmapsto} T' \Rightarrow \exists_{u \in T, u' \in T'}\ u = u' \vee u \in U \vee u' \in U$.

Proof. Assume $v \overset{i,U}{\longmapsto} T \wedge v \overset{\neg i,U}{\longmapsto} T'$. Then there must be strategies $s \in \mathbb{S}_i$ and $s' \in \mathbb{S}_{\neg i}$ such that $v \overset{s,U}{\longmapsto} T$ and $v \overset{s',U}{\longmapsto} T'$. Let s and s' be such, and consider the unique play p starting in v such that $s \Vdash p$ and $s' \Vdash p$. For this play, there must be m and n such that $p_m \in T \wedge \forall_{i<m}\ p_i \in U$, and $p_n \in T' \wedge \forall_{i<n}\ p_i \in U$. If $m = n$, then this witnesses $\exists_{u \in T, u' \in T'}\ u = u'$. If $m < n$, then $p_m \in T \wedge p_m \in U$, and if $n < m$, then $p_n \in T' \wedge p_n \in U$. □

The lemmas above reason about players being able to reach vertices. The following lemma is essentially about avoiding vertices: it states that if one player can force divergence, then this is the same as saying that the opponent cannot force the play outside the class of the current vertex.

Lemma 5.3. $v \overset{i,U}{\longmapsto} \iff v \overset{\neg i,U}{\not\longmapsto} V \setminus U$.

Proof. Note that the truth values of $v \overset{i,U}{\longmapsto}$ and $v \overset{\neg i,U}{\not\longmapsto} V \setminus U$ only depend on edges that originate in U, and that these truth values do not depend on priorities at all. Therefore, the truth value of these predicates will not change if we apply the following transformations to our graph:

- For all $u \in V \setminus U$, replace all outgoing edges by a single edge $u \to u$.
- Make the priorities of all vertices in U such that they are even iff $i = \Diamond$, and the priorities of all other vertices odd iff $i = \Diamond$.

In the resulting graph, player i wins if and only if $v \overset{i,U}{\longmapsto}$, and player $\neg i$ wins if and only if $v \overset{\neg i,U}{\longmapsto} V \setminus U$. Since v can only be won by one player, the desired result follows. □

Lastly, we want to formalise the idea that if a player can force the play to a first set of vertices, and from there he can force the play to a second set of vertices, then he must be able to force the play to that second set.

Lemma 5.4. $(v \overset{i,U}{\longmapsto} T \wedge (\forall_{u \in T \setminus T'} u \in U \wedge u \overset{i,U}{\longmapsto} T')) \Rightarrow v \overset{i,U}{\longmapsto} T'$.

Proof. Assume $v \overset{i,U}{\longmapsto} T \wedge (\forall_{u \in T \setminus T'} u \in U \wedge \overset{i,U}{\longmapsto} T')$. There must be a strategy $s \in \mathbb{S}_i$ such that $v \overset{s,U}{\longmapsto} T$ and for each $u \in T \setminus T'$ a strategy $s_u \in \mathbb{S}_i$ such that $u \overset{s_u,U}{\longmapsto} T'$. We define strategy $s' \in \mathbb{S}_i^*$ as follows for a path pw consisting of a prefix p and ending in vertex w:

$$s'(pw) = \begin{cases} s(w) & \text{if } \forall_{u \in T \setminus T'} u \notin pw \\ s_u(w) & \text{if } pw = p'up''w \wedge u \in T \setminus T' \wedge \forall_{u' \in T \setminus T'} u' \notin p' \end{cases}$$

Observe that a memoryless strategy $s'' \in \mathbb{S}_i$ can be found that has the same behaviour as s'. Furthermore $v \overset{s'',U}{\longmapsto} T'$, and hence $v \overset{i,U}{\longmapsto} T'$. □

These lemmas also appeared in almost the same form in [Kei13]. The formulation in this section is a bit different, because we have defined our arrow notation ($v \overset{i,U}{\longmapsto} T$ etc.) in terms of sets, rather than in terms of equivalence relations.

5.3 A lattice of equivalences

We already mentioned that parity games are *determined*, meaning that every node in the game is won by exactly one of the players [McN93; Zie98]. Determinacy of parity games effectively induces a partition on the set of vertices V in those vertices won by player $\Diamond$ and those vertices won by player $\Box$. This partition is the natural equivalence relation on V.

Definition 5.2. *Let $(V, \to, \Pi, \Omega)$ be a parity game. Vertices $v, w \in V$ are* winner equivalent*, denoted $v \sim w$ iff v and w are won by the same player.*

As explained in the introduction of this chapter, we are in search of equivalence relations that are relatively cheap to compute, and that are as coarse as possible while still refining winner equivalence.

We can use an equivalence relation to 'compress' parity games as follows. Given two parity games with vertices V_1 and V_2, respectively, and an equivalence relation $R \subseteq V_1 \times V_2$ such that every node in V_1 is related to exactly one node in V_2, we will first prove that R refines $\sim$. We now know that if we solve the second game, then we have also solved the first game, because within an equivalence class the solution for every vertex is the same. If this second game is significantly smaller than the first, which can be achieved by choosing R as coarse as possible, then the time gained by only having to solve the smaller game may be more than the time needed to find such a relation R, resulting in an overall faster solving time.

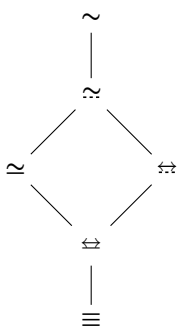


Figure 5.1: A lattice of relations on parity games.

The finest equivalence on parity games we can imagine in this setting is graph isomorphism, denoted $\equiv$. It has no reductive power, because in this case the 'smaller' game must have the same number of nodes and edges as the larger game.

The coarsest possible equivalence in this setting is winner equivalence itself. While this equivalence has the greatest reductive power, it is of course as difficult to compute as the solution to the larger game itself.

We will be considering four relations in this chapter that form a lattice that has winner equivalence as its greatest element, and graph isomorphism as its least element:

- strong bisimulation, denoted $\leftrightarrows$,
- stuttering bisimulation, denoted $\simeq$,
- governed bisimulation, denoted $\leftrightarrows$, and
- governed stuttering bisimulation, denoted $\simeq$.

The lattice formed by these relations is shown as a Hasse diagram in Figure 5.1. In the remainder of the chapter, we will give the definitions of these remaining four relations, and give proofs for the refinement relation between these parity game relations that is depicted by the Hasse diagram.

After isomorphism, strong bisimilarity is the finest relation on parity games that we consider. Its definition is essentially the same as the definition of strong bisimilarity for Kripke structures.

Definition 5.3. *Let $\langle V, \to, \Omega, \Pi \rangle$ be a parity game. Let $R \subseteq V \times V$ be a symmetric relation on vertices; R is a* strong bisimulation *if $v\, R\, v'$ implies*

- *$\Omega(v) = \Omega(v')$ and $\Pi(v) = \Pi(v')$;*
- *if $v \to u$ for some u, then $v' \to u'$ for some u' such that $u\, R\, u'$.*

Figure 5.2: Nodes with equal priorities are related by $\leftrightarrows$, but not by $\leftrightarrow$.

Two states v and v' are said to be strongly bisimilar*, denoted $v \leftrightarrow v'$, if and only if there is a strong bisimulation R such that $v \mathrel{R} v'$.*

It is straightforward to prove that strong bisimulation refines winner equivalence, and moreover, it is efficiently computable in $O(|{\rightarrow}|\log|V|)$ time [PT87]. Its ability to compress parity games is very limited however. One reason is that it will never relate two vertices if they are owned by different players. Not even when vertices have only one outgoing edge, in which case it really does not matter which player owns the vertex. Evidently, the nodes with equal priorities are won by the same player in Figure 5.2, but these nodes are not related by $\leftrightarrow$. This problem is solved by *governed bisimilarity*, a relation that is weaker than bisimilarity, but which still preserves winner equivalence.

Definition 5.4. *Let $\langle V, \rightarrow, \Omega, \Pi\rangle$ be a parity game. A symmetric relation $R \subseteq V \times V$ is a* governed bisimulation *if $v \mathrel{R} v'$ implies*

- $\Omega(v) = \Omega(v')$*;*
- $\Pi(v) \neq \Pi(v')$ *implies that $u \mathrel{R} u'$ for all u, u' such that $v \rightarrow u$ and $v' \rightarrow u'$;*
- *if $v \rightarrow u$ for some u, then $v' \rightarrow u'$ for some u' such that $u \mathrel{R} u'$.*

Vertices v and v' are said to be governed bisimilar*, denoted $v \leftrightarrows v'$, if and only if there is a governed bisimulation R such that $v \mathrel{R} v'$.*

The notion of governed bisimilarity corresponds to that of idempotence identifying bisimulation in Boolean equation systems [GW12]. Governed bisimilarity is again an equivalence relation, and is a governed bisimulation. It is easy to see from the definitions that bisimilarity refines governed bisimilarity; that the refinement is strict follows from the example in Figure 5.2.

Another reason that strong bisimulation does not compress games very well is that it is sensitive to repetition of priorities on paths. By the definition of a winning play, two plays are won by the same player if (but not only if) it traverses the same priorities in the same order. It does not matter how often a priority is seen before the next priority is encountered. This is illustrated by the two parity games in Figure 5.3.

We again try to relate these systems by finding a weaker relation that still refines winner equivalence. In this case, *stuttering bisimilarity* is our relation of choice. The traditional definition of stuttering bisimilarity (also known as stutter bisimilarity) is

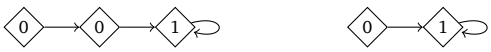


Figure 5.3: Nodes with equal priorities are related by $\simeq$, but not by $\leftrightarrow$.

Figure 5.4: The nodes with priority 0 are won by different players.

given by defining a *stuttering bisimulation* relation and defining stuttering bisimilarity to be the largest such relation, as is done in the following definition. The resulting relation relates nodes from which the players can visit the same priorities in the same order.

Definition 5.5. *Let $\langle V, \to, \Omega, \Pi\rangle$ be a parity game. Let $R \subseteq V \times V$ be a symmetric relation on vertices; R is a* stuttering bisimulation *if $v\ R\ v'$ implies*

- *$\Omega(v) = \Omega(v')$ and $\Pi(v) = \Pi(v')$;*
- *if $v \to u$, then either $v\ R\ u$, or there is a path $v' \to u_0 \to \ldots \to u_n$ such that $u\ R\ u_n$ and $v\ R\ u_i$ for all $i < n$;*
- *if there is an infinite path $v \to u_0 \to u_1 \to \ldots$ such that $v \mathcal{R} u_0$ and $u_i \mathcal{R} u_{i+1}$ for all i, then there is an infinite path $v' \to w_0 \to w_1 \to \ldots$ such that $v' \mathcal{R} w_0$ and $w_i \mathcal{R} w_{i+1}$ for all i.*

Two states v and v' are said to be stuttering bisimilar*, denoted $v \simeq v'$, if and only if there is a stuttering bisimulation relation R such that $v\ R\ v'$.*

The third clause in this definition makes the relation *divergence sensitive* [DV95]. It makes sure that we cannot relate the vertices with priority 0 in Figure 5.4.

It is a well-known result that $\simeq$ is an equivalence relation, and that it is again a stuttering bisimulation (see, *e.g.*, [BK08]). It was shown to refine winner equivalence in [CKW11], although this will also follow from the results in this chapter. We will be using a slightly different formulation of stuttering bisimilarity, which will make it easier to relate stuttering bisimulation to governed stuttering bisimulation. The following theorem gives our alternative definition of stuttering bisimulation.

Theorem 5.1. *Let $\langle V, \to, \Omega, \Pi\rangle$ be a parity game. Let $R \subseteq V \times V$ be an equivalence relation on vertices; R is an* eq-stuttering bisimulation *if $v\ R\ v'$ implies*

- *$\Omega(v) = \Omega(v')$ and $\Pi(v) = \Pi(v')$;*
- *$v \to \mathcal{C}$ implies $v' \overset{R}{\mapsto} \mathcal{C}$ for all $\mathcal{C} \in V_{/R} \setminus \{[v]_R\}$;*
- *$v \overset{R}{\mapsto}$ implies $v' \overset{R}{\mapsto}$.*

There is an eq-stuttering bisimulation relation R such that $v\ R\ v'$ if and only if $v \simeq v'$.

Proof. Suppose $v \simeq v'$, then there is an equivalence relation R that is a stuttering bisimulation such that $v\ R\ v'$: take $R{=}{\simeq}$. The theorem then follows from the fact that a stuttering bisimulation which is also an equivalence relation is an eq-stuttering bisimulation. This can be seen as follows.

Suppose R is an eq-stuttering bisimulation. Then it is an equivalence, and to prove that it is also a stuttering bisimulation, we need to show that the second condition from Definition 5.5 holds for R. So suppose $v \to u$ such that $\neg(v\ R\ u)$. Then $v \to \mathcal{C}$ for

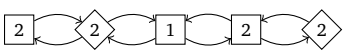
some $\mathcal{C} \in V_{/R} \setminus \{[v]_R\}$, and therefore $v' \overset{R}{\mapsto} \mathcal{C}$. In particular this means that there is a path $v' \to u_0 \to \ldots \to u_n$ for some $u_n \in \mathcal{C}$ (and $n \geq 0$), in which $v' \mathrel{R} u_i$ for all $i < n$ and $u \mathrel{R} u_n$. Because R is an equivalence relation, also $v \mathrel{R} u_i$ for all $i < n$.

Conversely, suppose that R is a stuttering bisimulation, and that it is also an equivalence relation. We prove that the second condition from Definition 5.1 holds for R. Suppose therefore that $v \to \mathcal{C}$ for some $\mathcal{C} \in V_{/R} \setminus \{[v]_R\}$, then $v \to u$ for some $u \in \mathcal{C}$, for which we know $\neg(v \mathrel{R} u)$. By Definition 5.5 there must be a path $v' \to u_0 \to \ldots \to u_n$ such that $v \mathrel{R} u_i$ for all $i < n$ and $u \mathrel{R} u_n$. Because R is an equivalence relation, $u_i \in [v']_R$ for all $i < n$, so $v' \overset{R}{\mapsto} u_n$. As $u \in \mathcal{C}$ and $u \mathrel{R} u'$, also $u_n \in \mathcal{C}$ and therefore $v' \overset{R}{\mapsto} \mathcal{C}$. □

So far, we have seen that there are winner-preserving relations that relate vertices owned by different players, and that there are winner-preserving relations that remove stuttering behaviour from games. The obvious question is whether we can combine the two.

Figure 5.5 shows an example of what we are trying to achieve. Vertices with equal priorities again are won by the same player, so we would like to design an equivalence relation that relates these vertices. To do so, we need to be able to relate vertices owned by different players, like governed bisimulation does, to relate vertices with priority 1. We must also be able to relate vertices that show the same behaviour modulo stuttering, to relate vertices with priority 2.

Our approach is to weaken stuttering bisimulation (while maintaining that it is a finer equivalence than winner equivalence) so that it will be able to relate vertices of different players. Note that we cannot simply weaken the definition of stuttering bisimulation by removing the requirement that $\Pi(v) = \Pi(v')$ without modifying the remaining clauses, as this would enable us to relate vertices won by different players, as the below parity game demonstrates:



The suggested weakening would allow us to relate all vertices with priority 2; the two left vertices, however are won by player ◇, whereas the other vertices are won by □.

The problem we see here is that it is no longer the case, as it was in stuttering bisimulation, that one player has total control over where the play goes from a certain equivalence class. In particular, in the two nodes on the right, it is player □ who controls whether the play stays in nodes with priority 2. Dually, in the two leftmost nodes it is player ◇ who decides.

If we drop the $\Pi(v) = \Pi(v')$ requirement from our definition, then apparently we need to strengthen the other requirements with information about who is in control

Figure 5.5: Governed stuttering bisimulation is weaker than stuttering bisimulation.

of the node. We change the $v \overset{R}{\mapsto} \mathcal{C}$ in Definition 5.1 to $v \overset{\Pi(v),R}{\longmapsto} \mathcal{C}$; for stuttering bisimilarity, the fact that all nodes in $[v]_R$ were owned by the same player, meant that the owner of v always had a strategy to reach $\mathcal{C}$ from v. Now multiple players are allowed in the same equivalence class, we have to exclude the possibility that somewhere along the way, the owner of v loses control to the opponent. This is exactly what $v \overset{\Pi(v),R}{\longmapsto} \mathcal{C}$ expresses: v can always force the play to reach $\mathcal{C}$, no matter which vertices in the equivalence class of v are visited first.

Something similar has to be done for the divergence criterion in the definition of stuttering bisimilarity. For stuttering bisimilarity, the existence of an infinite path of related vertices implies the existence of such a path in which the owner of v owned all the vertices along the path, which in turn implies the existence of a strategy for the owner of v to stay within the same equivalence class. If we drop the requirement that related vertices are owned by the same player, such a strategy is not guaranteed to exist anymore. If neither player has a strategy to stay in the current equivalence class, *i.e.*, staying in the current class requires cooperation from both players, then the infinite paths in this equivalence class have become irrelevant: if the priority of the current class is even, then player odd will always try to leave the current class, and vice versa. This is illustrated in Figure 5.6. The divergence requirement can therefore also be weakened: only if a player can force the play to stay in the current class, that player must be able to do so from any other vertex in the class.

Figure 5.6: Equal priorities are related by $\simeq$. Neither player can force the play to visit only vertices with priority 0.

We arrive at the following definition of a relation on parity games, which we have dubbed *governed stuttering bisimilarity*.

Definition 5.6. *Let $(V, \to, \Omega, \Pi)$ be a parity game. Let $R \subseteq V \times V$ be an equivalence relation. Then R is a* governed stuttering bisimulation *if $v \mathrel{R} v'$ implies*

- $\Omega(v) = \Omega(v')$;
- *$v \to \mathcal{C}$ implies $v' \overset{\Pi(v),R}{\longmapsto} \mathcal{C}$, for all $\mathcal{C} \in V_{/R} \setminus \{[v]_R\}$.*
- *$v \overset{i,R}{\longmapsto}$ implies $v' \overset{i,R}{\longmapsto}$ for $i \in \{\Diamond, \Box\}$.*

Vertices v and v' are governed stuttering bisimilar*, denoted $v \simeq v'$, iff a governed stuttering bisimulation R exists such that $v \mathrel{R} v'$.*

To complete our lattice of equivalence relations, we need to show that governed stuttering bisimilarity is weaker than both governed bisimilarity and stuttering bisimilarity, that it is indeed an equivalence relation, and finally, that governed stuttering bisimilarity refines winner equivalence. We treat these three issues separately in the following sections. After proving governed stuttering bisimilarity to be an equivalence

relation, we also define a quotienting operation that, given a parity game, yields a governed stuttering bisimilar parity game that is as small as possible. This quotienting operation is subsequently used in the proof for refinement of winner equivalence.

5.3.1 Governed stuttering bisimilarity is weaker

In this section we show that governed stuttering bisimilarity is a strictly coarser relation than both governed bisimilarity and stuttering bisimilarity. This is done in the following two theorems. In both cases, the proof of refinement is straightforward, and strictness can be shown with a small example.

Theorem 5.2. *Governed bisimilarity strictly refines governed stuttering bisimilarity.*

Proof. We show that $\leftrightarrows$ is a governed stuttering bisimulation. Let (1), (2) and (3) refer to the three requirements from Definition 5.4. The strictness of the refinement follows from the example in Figure 5.5. Let v and v' be such that $v \leftrightarrows v'$. We need to show three things.

- $\Omega(v) = \Omega(v')$. This follows immediately from (1).
- $v \to \mathcal{C}$ implies $v' \xrightarrow{\Pi(v),\leftrightarrows} \mathcal{C}$ for all $\mathcal{C} \in V_{/\leftrightarrows} \setminus \{[v]_{\leftrightarrows}\}$. Assume $v \to \mathcal{C}$ for such $\mathcal{C}$, *i.e.*, $v \to u$ for such u that $\neg v \leftrightarrows u$. Note that there exists some u' such that $v' \to u'$ and $u \leftrightarrows u'$. If $\Pi(v) = \Pi(v')$, then this suffices, because obviously the strategy for $\Pi(v)$ that sends the play to u' from v' is a witness for $v' \xrightarrow{\Pi(v),\leftrightarrows} \mathcal{C}$. If $\Pi(v) \neq \Pi(v')$, then $u \leftrightarrows u'$ for *all* u, u' such that $v' \to u$ and $v' \to u'$. Any strategy chosen for $\Pi(v)$ therefore witnesses $v' \xrightarrow{\Pi(v),\leftrightarrows} \mathcal{C}$.
- $v \xrightarrow{i,\leftrightarrows}$ implies $v' \xrightarrow{i,\leftrightarrows}$ for $i \in \{\Diamond, \Box\}$. Assume $v \xrightarrow{i,\leftrightarrows}$. Suppose u is a vertex such that $v \leftrightarrows u$ and $\Pi(u) = i$. We first argue that there is some u' such that $u \to u'$ and $v \leftrightarrows u'$. Because $v \xrightarrow{i,\leftrightarrows}$, there must be v' such that $v \to v'$ and $v \leftrightarrows v'$. By the third requirement in Definition 5.4 there is then some u' such that $u \to u'$ and $v' \leftrightarrows u'$. By transitivity of $\leftrightarrows$, then $v \leftrightarrows u'$.
  This means we can define a strategy $s \in \mathbb{S}_i$ such that $v \leftrightarrows s(u)$ if $v \leftrightarrows u$. Let s be such, and let u be such that $v \leftrightarrows u$. We show that if $u \xrightarrow{s} u'$, then $u \leftrightarrows u'$. If $\Pi(u) = i$, then it follows directly from our definition of s. If $\Pi(u) \neq i$, then we must show that all successors of u are related to u again. If $\Pi(u) = \Pi(v)$, then all successors of v are related to v, otherwise $v \xrightarrow{i,\leftrightarrows}$. But then by the third requirement in Definition 5.4, all successors of u must also be related to u. If $\Pi(u) \neq \Pi(v)$, then there must be v' such that $v \to v'$ and $v \leftrightarrows v'$, and then by the third requirement in Definition 5.4 also some u" such that $u \to u'$ and $v' \leftrightarrows u'$. Due to the second requirement in Definition 5.4, all successors of u are related to one another, and by transitivity, are in particular all related to u. Therefore, any play allowed by s stays in $[v]_{\leftrightarrows}$ indefinitely. □

Theorem 5.3. *Stuttering bisimilarity strictly refines governed stuttering bisimilarity.*

Proof. We show that any eq-stuttering bisimulation (as defined in Definition 5.1) is a governed stuttering bisimulation; the strictness of the refinement follows from the

example in Figure 5.5. Let R be an eq-stuttering bisimulation, and let $v\ R\ v'$ for some vertices v and v'. We need to show three things.

- $\Omega(v) = \Omega(v')$. This follows immediately from Definition 5.1.
- $v \to \mathcal{C}$ implies $v' \xmapsto{\Pi(v),R} \mathcal{C}$ for all $\mathcal{C} \in V_{/R} \setminus \{[v]_R\}$. So assume $v \to \mathcal{C}$ for some $\mathcal{C} \in V_{/R}$ such that $\mathcal{C} \neq [v]_R$. By Definition 5.1, $v' \xmapsto{R} \mathcal{C}$, so there is a path $p = v' \to \ldots \to u \to t$ such that $v', \ldots, u \in [v']_R$ and $t \in \mathcal{C}$. Because all those vertices are related by R, they must all be owned by the same player: $\Pi(v)$. Now consider a strategy s for player $\Pi(v)$ that chooses for each of these vertices the successor in p. Obviously, $v' \xmapsto{s,R} t$, and therefore $v' \xmapsto{\Pi(v),R} \mathcal{C}$.
- $v \xmapsto{i,R}$ implies $v' \xmapsto{i,R}$ for $i \in \{\Diamond, \Box\}$. The proof for this is similar to that of the second requirement. □

5.3.2 Governed stuttering bisimilarity is an equivalence

Proving that $\simeq$ is an equivalence relation on parity games is far from straightforward: transitivity no longer bows to the standard proof strategies that work for stuttering bisimilarity and branching bisimilarity [GW96; Bas96]. As a result of the asymmetry in the use of two different transition relations in the second requirement of Definition 5.6, proving that the equivalence closure of the union of two governed stuttering bisimulation relations is again a governed stuttering bisimulation relation is equally problematic.

Instead we pursue the following strategy. We characterise governed stuttering bisimulation, in three steps, by a set of symmetric requirements. The obtained alternative characterisation is then used in our equivalence proof. These alternative characterisations do not facilitate the reuse of standard proof strategies, but they are instrumental in the technically involved proof that the equivalence closure of two governed stuttering bisimulation relations is again a governed stuttering bisimulation relation. Apart from being convenient technically, the characterisations offer more insight into the nature of governed stuttering equivalence.

Our result below states that we can rephrase the second condition of governed stuttering bisimulation by requiring that either player must have the same power to force the play from any pair of related vertices to reach an arbitrary class. Thus, we abstract from the player that takes the initiative to leave its class in one step.

Theorem 5.4. *Let $R \subseteq V \times V$ and $v, v' \in V$. Then R is a governed stuttering bisimulation iff R is an equivalence relation and $v\ R\ v'$ implies:*

- $\Omega(v) = \Omega(v')$;
- $v \xmapsto{i,R} \mathcal{C}$ *iff* $v' \xmapsto{i,R} \mathcal{C}$ *for all* $i \in \{\Diamond, \Box\}, \mathcal{C} \in V_{/R} \setminus \{[v]_R\}$;
- $v \xmapsto{i,R}$ *iff* $v' \xmapsto{i,R}$ *for all* $i \in \{\Diamond, \Box\}$.

Proof. The proof for the implication from right to left is immediate. For the other direction, assume that R is a governed stuttering bisimulation. Observe that it suffices to prove the second condition; the other two are in full agreement with Definition 5.6.

Let i be an arbitrary player, and suppose that $v \overset{i,R}{\longmapsto} \mathcal{C}$ for some $v \in V$ and $\mathcal{C} \in V_{/R} \setminus \{[v]_R\}$. Obviously, there must be some $u \in [v]_R$ such that $u \to \mathcal{C}$. Let u be such, and distinguish the following two cases:

- Case $\Pi(u) = i$. It follows directly from Definition 5.6 that $v' \overset{i,R}{\longmapsto} \mathcal{C}$.
- Case $\Pi(u) \neq i$. By Definition 5.6, because $v \, R \, u$, $v \overset{\neg i,R}{\longmapsto} \mathcal{C}$. From Lemma 5.3 it then follows that $v \overset{\neg i,R}{\not\longmapsto}$ and $v \overset{i,R}{\not\longmapsto}$, because $v \overset{i,R}{\longmapsto} \mathcal{C}$ and $v \overset{\neg i,R}{\longmapsto} \mathcal{C}$. By Definition 5.6, it then also holds that $v' \overset{i,R}{\not\longmapsto}$ and $v' \overset{\neg i,R}{\not\longmapsto}$.
  Towards a contradiction, suppose that $v' \overset{i,R}{\longmapsto} \mathcal{C}$. By Lemma 5.1 it must then be the case that $v' \overset{\neg i,R}{\longmapsto} V \setminus \mathcal{C}$. Because of this, and $v \overset{\neg i,R}{\not\longmapsto}$ and $v \overset{i,R}{\not\longmapsto}$ it follows that there must be some $u' \in [v]_R$ such that $u' \to V \setminus \mathcal{C} \setminus [v]_R$. We again distinguish two cases:
  - Case $\Pi(u') = i$. Then $u' \overset{i}{\to} V \setminus \mathcal{C} \setminus [v]_R$, and by Definition 5.6, $u \overset{i,R}{\longmapsto} V \setminus \mathcal{C} \setminus [v]_R$, which contradicts $u \overset{\neg i,R}{\longmapsto} \mathcal{C}$ according to Lemma 5.3.
  - Case $\Pi(u') \neq i$. Then $u' \overset{\neg i}{\to} V \setminus \mathcal{C} \setminus [v]_R$, and by Definition 5.6, $v \overset{\neg i,R}{\longmapsto} V \setminus \mathcal{C} \setminus [v]_R$, which contradicts $v \overset{i,R}{\longmapsto} \mathcal{C}$ according to Lemma 5.3. □

While the above alternative characterisation of governed stuttering bisimulation is now fully symmetric, the restriction on the class $\mathcal{C}$ that is considered in the second condition turns out to be too strong to facilitate our proof that $\simeq$ is an equivalence relation. We generalise this condition once more to reason about sets of classes. A perhaps surprising side effect of this generalisation is that the third condition about divergence becomes superfluous. Note that this last generalisation is not trivial, as $v \overset{i,R}{\longmapsto} \{\mathcal{C}_1, \mathcal{C}_2\}$ is in general neither equivalent to saying that $v \overset{i,R}{\longmapsto} \mathcal{C}_1$ and $v \overset{i,R}{\longmapsto} \mathcal{C}_2$, nor to $v \overset{i,R}{\longmapsto} \mathcal{C}_1$ or $v \overset{i,R}{\longmapsto} \mathcal{C}_2$.

Theorem 5.5. *Let $R \subseteq V \times V$ and $v, v' \in V$. Then R is a governed stuttering bisimulation iff R is an equivalence relation and $v \, R \, v'$ implies:*

- $\Omega(v) = \Omega(v')$;
- $v \overset{i,R}{\longmapsto} \mathcal{U}$ *iff* $v' \overset{i,R}{\longmapsto} \mathcal{U}$ *for all* $i \in \{\Diamond, \Box\}, \mathcal{U} \subseteq V_{/R} \setminus \{[v]_R\}$.

Proof. We show that the second condition is equivalent to the conjunction of the last two conditions in Theorem 5.4. We split the proof into an *if*-part and an *only-if*-part.

⇐ The second condition from Theorem 5.4 is equivalent to the second condition above, if we let $\mathcal{U}$ range only over singleton sets (if $v \overset{i,R}{\longmapsto} \mathcal{C}$, take $\mathcal{U} = \{\mathcal{C}\}$). The third condition is equivalent to the second condition above, where $\mathcal{U} = V_{/R} \setminus \{[v]_R\}$. This can be seen by appealing to Lemma 5.3.

⇒ Let R be a governed stuttering bisimulation relation and let $v, v' \in V$ such that $v \, R \, v'$. Assume that $v \overset{i,R}{\longmapsto} \mathcal{U}$ for some $\mathcal{U} \subseteq V_{/R} \setminus \{[v]_R\}$. Let $S = \{u \in [v]_R \mid u \to \mathcal{U}\}$. We distinguish the following two cases:

Case $\exists_{u \in S} \, \Pi(u) = i$. Let u be such. There is a class $\mathcal{C} \in \mathcal{U}$ such that $u \to \mathcal{C}$ (in particular, $u \overset{i,R}{\longmapsto} \mathcal{C}$). By Theorem 5.4 then also $v' \overset{i,R}{\longmapsto} \mathcal{C}$, from which $v' \overset{i,R}{\longmapsto} \mathcal{U}$ follows immediately.

Case $\forall_{u\in S}\ \Pi(u)\neq i$**.** Towards a contradiction, suppose that $\exists_{w\in V\setminus\mathcal{U}}\ u\to w$ for all $u\in S$. Let $s\in\mathbb{S}_i$ be such that $v \overset{s,R}{\longmapsto} \mathcal{U}$, and define a strategy $s'\in\mathbb{S}_{\neg i}$ such that $s'(u)\notin\bigcup\mathcal{U}$ for all $u\in S$. Now consider the unique path p such that $s \Vdash p$ and $s' \Vdash p$. Because $v \overset{s,R}{\longmapsto} \mathcal{U}$, there must be some $i\in\mathbb{N}$ such that $p_i\in[v]_R$ and $p_{i+1}\in\bigcup\mathcal{U}$. Moreover, $p_i\in S$, so $\Pi(p_i)\neq i$, and therefore $p_{i+1}=s'(p_i)$. However, this contradicts the fact that $s'(p_i)\notin\bigcup\mathcal{U}$ by the definition of s'.

From the above it follows that there must be some vertex $u\in S$ such that $\forall_{w\in V}\ u\to w\Rightarrow u\in\bigcup\mathcal{U}$. In particular, $u \overset{\neg i,R}{\longmapsto\!\!\!\!\!\!\!\!\;/\;\;} \mathcal{C}$ for all $\mathcal{C}\in V_{/R}\setminus\mathcal{U}$.

From $v \overset{i,R}{\longmapsto} \mathcal{U}$ we derive, using Lemma 5.3, that $v \overset{\neg i,R}{\longmapsto\!\!\!\!\!\!\!\!\;/\;\;}$. By Theorem 5.4 it follows that $v' \overset{\neg i,R}{\longmapsto\!\!\!\!\!\!\!\!\;/\;\;}$, and by Lemma 5.3 again $v' \overset{i,R}{\longmapsto} V\setminus[v]_R$. Let $s\in\mathbb{S}_i$ be such that $v' \overset{s,R}{\longmapsto} V\setminus[v]_R$, and consider any play p starting in v' such that $s\Vdash p$. There must be some $n\in\mathbb{N}$ such that $\forall_{i\leq n}\ p_i\in[v]_R$ and $p_{n+1}\notin[v]_R$.

Note that $\Pi(p_n)\neq i$; if $\Pi(p_n)=i$, then $p_n\to\mathcal{C}$ for some $\mathcal{C}$ such that $\mathcal{C}\notin\mathcal{U}$ (because $p_n\notin S$ by our assumption that $\Pi(u)\neq i$ for all $u\in S$) and $\mathcal{C}\neq[v]_R$. This $\mathcal{C}$ witnesses $p_n \overset{i,R}{\longmapsto} V_{/R}\setminus\mathcal{U}\setminus\{[v]_R\}$. But because $p_n\ R\ u$, and $u \overset{\neg i,R}{\longmapsto} \mathcal{C}$ for some $\mathcal{C}\in\mathcal{U}$, also $p_n \overset{\neg i,R}{\longmapsto} \mathcal{U}$ by Theorem 5.4, which contradicts $p_n \overset{i,R}{\longmapsto} V_{/R}\setminus\mathcal{U}\setminus\{[v]_R\}$ by Lemma 5.2.

Towards a contradiction, suppose that $p_{n+1}\notin\bigcup\mathcal{U}$, then $p_n \overset{\neg i,R}{\longmapsto} \mathcal{C}$ for some $\mathcal{C}\in V_{/R}\setminus\mathcal{U}$. But then by Theorem 5.4, $u \overset{\neg i,R}{\longmapsto} \mathcal{C}$, which we had already established was not the case. So $p_{n+1}\in\bigcup\mathcal{U}$.

Apparently, for all plays p starting in v' allowed by s there is some n such that $\forall_{i<n}\ p_i\in[v]_R$ and $p_{n+1}\in\bigcup\mathcal{U}$. Therefore, s witnesses $v' \overset{i,R}{\longmapsto} \mathcal{U}$. □

Note that the divergence requirement $v \overset{i,R}{\longmapsto}$ iff $v' \overset{i,R}{\longmapsto}$ can be recovered by instantiating set $\mathcal{U}$ by $V_{/R}\setminus\{[v]_R\}$ for player $\neg i$ in the above theorem. We give one more alternative definition. In the previous definition, we lifted the notion of forcing play via the current equivalence class towards a target class, to the notion of forcing play via the current equivalence class towards a set of target classes. In the next definition, we perform a similar lifting; rather than forcing play towards a set of target classes via the current equivalence class, we now allow the play to be forced to that set via a set of equivalence classes.

Theorem 5.6. *Let $R\subseteq V\times V$ and $v,v'\in V$. Then R is a governed stuttering bisimulation iff R is an equivalence relation and $v\ R\ v'$ implies:*

- $\Omega(v)=\Omega(v')$
- $v \overset{i,\mathcal{U}}{\longmapsto} \mathcal{T}$ *iff* $v' \overset{i,\mathcal{U}}{\longmapsto} \mathcal{T}$ *for all* $i\in\{\Diamond,\Box\},\mathcal{U},\mathcal{T}\subseteq V_{/R}$ *such that* $[v]_R\in\mathcal{U}$ *and* $[v]_R\notin\mathcal{T}$.

Proof. We show that the second condition is equivalent to second condition in Theorem 5.5. We split the proof into an *if*-part and an *only-if*-part.

⇐ The second condition from Theorem 5.5 is equivalent to the second condition above if we fix $\mathcal{U}=\{[v]_R\}$.

⇒ Let R be a governed stuttering bisimulation and let $i,v,v',\mathcal{U}$ and $\mathcal{T}$ be as described. Assume that $v \overset{i,\mathcal{U}}{\longmapsto} \mathcal{T}$; under this assumption we will prove that $v' \overset{i,\mathcal{U}}{\longmapsto} \mathcal{T}$. The proof for the implication in the other direction is completely symmetric.

Let s be such that $v \overset{s,\mathcal{U}}{\longmapsto} \mathcal{T}$ and consider the set of paths originating in v that are allowed by s. All these paths must have a prefix $v \dots w, u$ such that $v, \dots, w \notin \bigcup \mathcal{T}$ but $u \in \bigcup \mathcal{T}$. Call these prefixes the s-prefixes of v.

We proceed by induction on the length of the longest such prefix. If the longest prefix has length 2, then all prefixes have length 2, implying that $v \xrightarrow{i} \mathcal{T}$. In particular then $v \overset{i,R}{\longmapsto} \mathcal{T}$ and by Theorem 5.5 also $v' \overset{i,R}{\longmapsto} \mathcal{T}$, which proves $v' \overset{i,\mathcal{U}}{\longmapsto} \mathcal{T}$.

As the induction hypothesis, assume that if $u \, R \, u'$, $u \overset{s,\mathcal{U}}{\longmapsto} \mathcal{T}$ and the longest s-prefix of u is shorter than the longest s-prefix of v, then $u' \overset{i,\mathcal{U}}{\longmapsto} \mathcal{T}$. Note that every s-prefix p of v must have a first position n such that $p_n \notin [v]_R$. Collect all these p_n in a set U, and notice that for all $u \in U$, $u \in \bigcup \mathcal{T}$, or $u \overset{s,\mathcal{U}}{\longmapsto} \mathcal{T}$. Furthermore, $v \overset{s,R}{\longmapsto} U$.

Let $[U]_R$ denote $\{[u]_R \mid u \in U\}$. By Theorem 5.5, $v' \overset{i,R}{\longmapsto} [U]_R$. Now consider an arbitrary $u' \in \bigcup([U]_R \setminus \mathcal{T})$. Because there is some $u \in U$ such that $u \, R \, u'$, and because $u \overset{s,\mathcal{U}}{\longmapsto} \mathcal{T}$ for such u, we can use the induction hypothesis to derive that $u' \overset{i,\mathcal{U}}{\longmapsto} \mathcal{T}$.

The above in particular implies two facts:

- $v' \overset{i,\mathcal{U}}{\longmapsto} [U]_R$
- $u' \overset{i,\mathcal{U}}{\longmapsto} \mathcal{T}$ for all $u' \in \bigcup([U]_R \setminus \mathcal{T})$

Using these, we can now apply Lemma 5.4 to conclude that $v' \overset{i,\mathcal{U}}{\longmapsto} \mathcal{T}$. □

With this last characterization, it is now straightforward to prove that governed stuttering bisimilarity is an equivalence relation. We do so by showing that the transitive closure of the union of two governed stuttering bisimulations R and S is again a governed stuttering bisimulation. The generalizations from classes to sets of classes allows us to view equivalence classes in $(R \cup S)^*$ as the union of sets of equivalence classes from R (or S), giving us an easy way to compare the effect of the second requirement of Theorem 5.6 on $(R \cup S)^*$ with its effect on R and S.

Theorem 5.7. *Governed stuttering bisimilarity is an equivalence relation.*

Proof. We show that $(R \cup S)^*$ is a governed stuttering bisimulation if R and S are, by showing that $(R \cup S)^*$ satisfies the conditions of Theorem 5.6 if R and S do. If $v, v' \in V$ are related under $(R \cup S)^*$, then there exists a sequence of vertices $u_0, \dots u_n$ such that $v \, R \, u_0 \, S \, \dots \, R \, u_n \, S \, v'$ (the strict alternation between the two relations can always be achieved because R and S are reflexive). By transitivity of $=$ we then have $\Omega(v) = \Omega(v')$, so the first property is satisfied.

For the second property, assume that $v \overset{i,\mathcal{U}}{\longmapsto} \mathcal{T}$ for some $i \in \{\Diamond, \Box\}$ and some $\mathcal{U}, \mathcal{T} \subseteq V_{/(R \cup S)^*}$ such that $[v]_{(R \cup S)^*} \in \mathcal{U}$ and $[v]_{(R \cup S)^*} \notin \mathcal{T}$. We need to prove that $v' \overset{i,\mathcal{U}}{\longmapsto} \mathcal{T}$. Note that R and S both refine $(R \cup S)^*$, so we can find sets $\mathcal{U}_R \subseteq V_{/R}$ and $\mathcal{U}_S \subseteq V_{/S}$ such that $\bigcup \mathcal{U}_R = \bigcup \mathcal{U}_S = \bigcup \mathcal{U}$. Because $v \overset{i,\mathcal{U}}{\longmapsto} \mathcal{T}$, also $v \overset{i,\mathcal{U}_R}{\longmapsto} \mathcal{T}$, and by Theorem 5.6 then $u_0 \overset{i,\mathcal{U}_R}{\longmapsto} \mathcal{T}$, which is equivalent to $u_0 \overset{i,\mathcal{U}_S}{\longmapsto} \mathcal{T}$. By a simple inductive argument we now arrive at $v' \overset{i,\mathcal{U}_S}{\longmapsto} \mathcal{T}$, which is equivalent to $v' \overset{i,\mathcal{U}}{\longmapsto} \mathcal{T}$. □

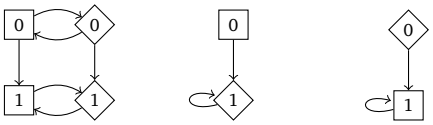


Figure 5.7: Both (b) and (c) are minimal representations of (a).

Quotienting

One reason for studying equivalence relations for parity games is that they may offer the prospect of minimising the parity game by merging vertices that are considered equivalent. The resulting minimised structure is referred to as the quotient. However, not all equivalence relations admit such a quotienting operation; in particular, the delayed simulation relation [FW06] for parity games does not seem to have a natural quotienting operation.

Quotienting for governed stuttering bisimulation can be done efficiently. Due to the nature of governed stuttering bisimulation, we have some freedom in the definition of the quotient, in particular when assigning players to the vertices in the quotient. We therefore first define a notion of minimality, and we subsequently define the quotient in terms of that notion.

Definition 5.7. *A* $\simeq$*-*minimal *representation of a parity game* $(V, \to, \Omega, \Pi)$ *is defined as a game* $(V_{\mathrm{m}}, \to_{\mathrm{m}}, \Omega_{\mathrm{m}}, \Pi_{\mathrm{m}})$*, that satisfies the following conditions (where* $c, c', c'' \in V_{\mathrm{m}}$*):*

$$
\begin{aligned}
V_{\mathrm{m}} &= \{[v]_{\simeq} \mid v \in V\} \\
\Omega_{\mathrm{m}}(c) &= \Omega(v) \text{ for all } v \in c \\
\Pi_{\mathrm{m}}(c) &= i, \text{ if for all } v \in c, \text{ and some } c' \neq c \text{ we have } v \xrightarrow{i,\simeq} c' \text{ and } v \xrightarrow{\neg i,\simeq} V \setminus c' \\
c \to_{\mathrm{m}} c &\text{ iff } v \xrightarrow{i,\simeq} \text{ for all } v \in c \text{ for some player } i \\
c \to_{\mathrm{m}} c' &\text{ iff } v \xrightarrow{i,\simeq} c' \text{ for all } v \in c \text{ for some player } i \text{ and } c' \neq c
\end{aligned}
$$

Observe that for the third clause above, if from some vertex v the play could be forced to c' by i without $\neg i$ having the opportunity to diverge, player i is in charge of the game when the play arrives in c. This requires the representative in the quotient to be owned by player i.

Note that a parity game may have multiple $\simeq$-minimal representations. It is not hard to verify that every parity game contains at least as many vertices and edges as its $\simeq$-minimal representations. Moreover, any parity game is governed stuttering bisimulation equivalent to all its $\simeq$-minimal representations, in the sense that every vertex in the original game has a governed stuttering bisimilar vertex in the $\simeq$-minimal representation. As a result, the governed stuttering bisimulation quotient of a graph can be defined as its least $\simeq$-minimal representation, given some arbitrary ordering on parity games. A natural ordering would be one that is induced by an ordering on players, *e.g.*, $\Box < \Diamond$.

Example. Consider the parity game in Figure 5.7a. Two of its four minimal representations are in Figure 5.7b and 5.7c. Observe that the particular player chosen for the 0 and 1 vertices is arbitrary and does not impact the solution to the games. The cycle formed by the 0 vertices does not give rise to a self-loop in the minimal representations, because neither player can force to stay on that cycle. This is not the case for the cycle of 1 vertices, hence we see a self-loop in the minimal representation. ■

5.3.3 Governed stuttering bisimilarity refines winner equivalence

In this section, we prove that governed stuttering bisimilarity is strictly finer than winner equivalence. That is, vertices that are won by different players are never related by governed stuttering bisimilarity. In order to prove this result, we must first lift the concept of governed stuttering bisimilarity to paths.

Paths of length 1 are equivalent if the vertices they consist of are equivalent. If paths p and q are equivalent, then $pv \simeq q$ iff v is equivalent to the last vertex in q, and $pv \simeq qw$ iff $v \simeq w$. An infinite path p is equivalent to a path q if for all finite prefixes of p there is an equivalent prefix of q and *vice versa*.

We define $\mathbb{P}^n_s(v)$ to be the set of paths of length n that start in v and that are allowed by some strategy s. $\mathbb{P}^\omega_s(v)$ is then the set of all infinite paths allowed by s, starting in v. By $\mathbb{P}_s(v)$ we denote $\bigcup_{n\in\mathbb{N}} \mathbb{P}^n_s(v)$, *i.e.*, the set of all finite paths starting in v and allowed by s.

Lemma 5.5. *Let $(V, \to, \Pi, \Omega)$ be a parity game, and let $(Q, \to, \Pi, \Omega)$ be its quotient. Let $v \in V$, and $w \in Q$. For all $s \in \mathbb{S}_i$ there is some $s' \in \mathbb{S}^*_i$ such that for all $q \in \mathbb{P}^\omega_{s'}(w)$ there is a $p \in \mathbb{P}^\omega_s(v)$ such that $p \simeq q$.*

Proof. For finite paths p, let $\bar{p}$ denote p without its last vertex. Define an arbitrary complete ordering $\prec$ on vertices, and define the following for finite paths q, where $\min_\prec \emptyset$ is defined to be $\bot$:

$$\mathsf{next}(q) = \min_\prec \{v' \in V \mid \exists_{p\in\mathbb{P}_s(v)}\ p \simeq q \wedge p \xrightarrow{s} v' \wedge pv' \not\simeq q\}$$
$$\mathsf{div}(q) = \exists_{p\in\mathbb{P}^\omega_s(v)}\ p \simeq q$$

Define a strategy $s' \in \mathbb{S}^*_i$ for finite paths $q = w \ldots w'$ such that if $q \simeq p$ for some $p \in \mathbb{P}_s(v)$, then:

$$\begin{cases} s'(q) = w' & \text{if } \mathsf{div}(w') \text{ and } w' \to w' \\ s'(q) \simeq \mathsf{next}(q) & \text{otherwise.} \end{cases}$$

We first establish that it is always possible to find such s'. Let $w' \in V$ such that $\Pi(w') = i$. If $\mathsf{div}(q)$ and $w \to w'$, then obviously $s'(w')$ can be defined to be w'. If $\neg\mathsf{div}(q)$ or $w \not\to w'$, then first show that $\mathsf{next}(q) \neq \bot$, by distinguishing two cases:

Case $\mathsf{div}(q)$, then $w' \not\to w'$. Because w' is a vertex in a quotient graph, $w' \xrightarrow{i,\simeq}\!\!\!\!\!\!\not\;\;$. Because $q \simeq p$ for some $p \in \mathbb{P}_s(v)$, there must also be v' such that $p \xrightarrow{s} v'$, so if $\mathsf{next}(q) = \bot$, this must be because for such p and v', $pv' \simeq q$. This can however not be the case, because this would imply that if $p \simeq q$ and $p \xrightarrow{s} v'$, then $pv' \simeq p$;

in other words, all paths allowed by s are governed stuttering equivalent to p. This would however mean that $p \xmapsto{s,\simeq}$, which contradicts $p \xmapsto{i,\simeq}$.

Case $\neg\mathsf{div}(q)$, then it follows straightforwardly that $\mathsf{next}(q) \neq \bot$.

We now know that if $\neg\mathsf{div}(q)$ or $w' \not\to w'$, then $\mathsf{next}(q) \neq \bot$. Therefore, there must be some $p = v \dots v' v'' \in \mathbb{P}_s(v)$ such that $v'' = \mathsf{next}(q)$, $v \dots v' \simeq q$ and $v' \not\simeq v''$. We now show that we can define $s'(q)$ such that $s'(q) \simeq \mathsf{next}(q)$. Distinguish cases on $\Pi(v')$:

Case $\Pi(v') = i$, then $v' \xmapsto{i,\simeq} [v'']_\simeq$, and because $v' \simeq w'$, also $w' \xmapsto{i,\simeq} [v'']_\simeq$. Because w' is a vertex in a quotient graph, this must mean that $w' \to [v'']_\simeq$, so we can choose $s'(q) \in [v'']_\simeq$.

Case $\Pi(v') \neq i$, then $v' \xmapsto{\neg i,\simeq} [v'']_\simeq$, and therefore $w' \xmapsto{\neg i,\simeq} [v'']_\simeq$. But then $w' \to w''$ implies $w'' \simeq v''$ for all w'', which can be seen as follows. If $w' \to w''$ such that $w' \simeq w''$ then $w' = w''$ because w' is a vertex in a quotient graph, and hence $w' \xmapsto{\Pi(,)\simeq}$, which would contradict $w' \xmapsto{\neg i,\simeq} [v'']_\simeq$ by Lemma 5.3. If on the other hand $w' \to w''$ such that $w' \not\simeq w''$ and $w'' \not\simeq v''$, then $w' \xmapsto{\Pi(,)\simeq} [w'']_\simeq$, which contradicts $w' \xmapsto{\neg i,\simeq} [v'']_\simeq$ by Lemma 5.2. So there must be at least one successor (because the transition relation is total) w'' such that $w' \to w''$ and $w'' \simeq v''$. We can therefore choose $s'(q) \in [v'']_\simeq$.

Now we have shown that it is always possible to define a strategy adhering to the restrictions above, let s' be such a strategy. We show with induction on n that for all n,

$$\forall_{q \in \mathbb{P}^n_{s'}(w)} \exists_{p \in \mathbb{P}_s(v)} p \simeq q.$$

For $n = 0$, this is trivial, because $v \simeq w$. For $n = m + 1$, assume as the induction hypothesis that $\forall_{\bar{q} \in \mathbb{P}^m_{s'}(w)} \exists_{\bar{p} \in \mathbb{P}_s(v)} \bar{p} \simeq \bar{q}$. Let $q \in \mathbb{P}^n_{s'}(w)$ and let $w', w'' \in V$ and $\bar{q} \in \mathbb{P}^m_{s'}(w)$ such that $\bar{q} = w \dots w'$ and $q = \bar{q} w''$. Distinguish cases on the player who owns w'.

Case $\Pi(w') = i$. Then $v' = s'(\bar{q})$. The induction hypothesis yields some $\bar{p} \in \mathbb{P}_s(v)$ such that $\bar{p} \simeq \bar{q}$, therefore the restrictions on $s'(\bar{q})$ formulated above apply.

If $w' = w''$, then $\mathsf{div}(\bar{q})$, so there must be some $p \in \mathbb{P}^\omega_s(v)$ such that $p \simeq q$.

If $w' \neq w''$, then $w'' \simeq \mathsf{next}(q)$, so there is some $p \in \mathbb{P}_s(v)$ such that $p = p' v'$ and $p' \simeq \bar{q}$ and $v' \simeq s'(\bar{q})$. By definition, $p \simeq q$ for such p.

Case $\Pi(w') \neq i$. From the induction hypothesis, obtain a $\bar{p} \in \mathbb{P}_s(v)$ such that $\bar{p} \simeq \bar{q}$. Let v' be the last vertex in $\bar{p}$. Note that $w' \xmapsto{\neg i,\simeq} [w'']_\simeq$, and because $\bar{p} \simeq \bar{q}$, also $v' \xmapsto{\neg i,\simeq} [w'']_\simeq$. So let $s' \in \mathbb{S}_{\neg i}$ be such that $v' \xmapsto{s',\simeq} [w'']_\simeq$. Now consider an infinite path $\bar{p}p$ such that $s \Vdash \bar{p}p$ and $s' \Vdash \bar{p}p$. For some index $k \geq 0$, it must be the case that $p_k \simeq w''$ and $p_l \simeq w'$ for all $l < k$. So $\bar{p} p_0 \dots p_k \simeq q$. □

Theorem 5.8. *Governed stuttering bisimilarity strictly refines winner equivalence.*

Proof. Let $G = (V, \to, \Pi, \Omega)$ be a parity game, and let $v, w \in V$ such that $v \simeq w$. Let $(Q, \to, \Pi, \Omega)$ be the quotient of G, and let $c \in Q$ be such that $w \simeq c$. By transitivity of $\simeq$, also $v \simeq c$. Now suppose that player i has a winning strategy s from v. Then by

Lemma 5.5, i has a strategy s' from c such that for every play $q \in \mathbb{P}^{\omega}_{s'}(c)$ there is a play $p \in \mathbb{P}^{\omega}_{s}(v)$ such that $p \simeq q$. Because the priorities occurring infinitely often on such p and q are the same, s' is also winning for i. If $\neg i$ had a winning strategy s' from w, then we could repeat this argument to construct a winning strategy for $\neg i$ from c, but this would be contrary to the fact that parity games are determined. Therefore, w must also be won by player i. □

5.4 Performance

In [CKW12b] we presented an algorithm that calculates the governed stuttering bisimulation quotient of a parity game in $\mathcal{O}(n^2m)$ time, where n is the number of nodes and m the number of edges in the parity game. This complexity is a factor n worse than the $\mathcal{O}(nm)$ worst case time complexity for calculating the stuttering bisimulation quotient.

While the extra complexity may seem demotivating at first, because we were explicitly looking for relations that were efficiently computable, there are reasons to believe that in practice, governed stuttering equivalence may perform better than stuttering equivalence. The extra complexity is caused by a rather intriguing special case, described as Problem 4.59 in [Kei13]. Because this case is so specific, it may be expected that the extra complexity does not manifest itself in the average case. Another reason why the extra complexity may not be as severe an impediment as it seems, is that one of the factors n is in fact an upper bound on the number of partitions that are generated. If in practice, governed stuttering equivalence yields a smaller number of equivalence classes than stuttering equivalence, then the partition refinement algorithm will terminate after fewer refinement steps. Indeed, the experiments carried out in [CKW12b] show that calculating the governed stuttering equivalence quotient of a game is not significantly slower than calculating the stuttering equivalence quotient.

A very elaborate discussion of the reductive power of the presented equivalences, and of the speedup that can be achieved by solving the quotient of a game rather than the original, can be found in [Kei13]. We give a brief summary here. The results pertain to a set of parity games that is categorized as follows.

- The 'modelchecking' category contains parity games that encode a model checking problem. In particular, reachability, liveness and fairness properties are checked on communication protocols, a cache coherence protocol, two-player board games and models of industrial systems. Characteristic is that the number of priorities in these games is low (usually 3 or less).
- The 'equivalence' category contains parity games that encode an equivalence checking problem. A number of communication protocols that make use of a buffer is instantiated for different buffer sizes. It is then checked whether these buffers are bisimilar, weakly bisimilar, branching bisimilar and branching simulation equivalent. Again the number of priorities in these games is low (1 or 2).
- The 'mlsolver' category consists of parity games that encode the problem of deciding whether a modal formula is satisfiable (*i.e.*, if there is a model in which it holds),

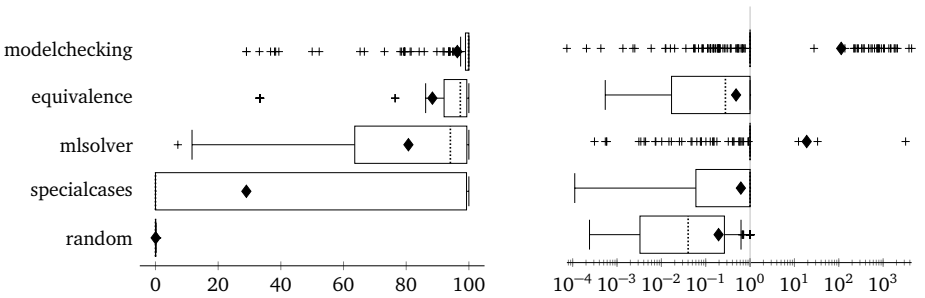
and games that encode the problem of deciding whether a modal formula is valid (*i.e.*, if it holds in all models). The games are generated by the MLSolver tool [FL10], and contain a larger number of priorities than those in the previous categories.

– The 'specialcases' category is a collection of parity games that were designed to be difficult to solve with a specific parity game solver. In particular, it includes games that demonstrate the lower bound for the small progress measures algorithm [Jur00], strategy improvement algorithms [VJ00; Sch07] and the local algorithm from [SS98]. These games typically have a larger number of priorities, but have a regular structure because they are composed of gimmicks that are difficult to deal with for the particular algorithm they were designed for.
– Finally, a 'random' category of randomly generated parity games is included. As the name suggests, these games contain arbitrary amounts of priorities and have an irregular structure.

Figure 5.8a shows the size reduction that is achieved by reducing the parity games modulo governed stuttering bisimulation. We see that for the real-world problems (model checking, equivalence checking and satisfiability / validity checking), the reductions are quite promising at over 80%. The special cases either reduce to two nodes, or can practically not be reduced at all. Given that these graphs have a very regular structure, this is to be expected. The random games can hardly be reduced at all, which is unsurprising, given that governed stuttering relates nodes with identical potential, and the potential of every node is randomized.

The size reductions for the other relations are very similar in their distribution, but worse on average; strong bisimulation reduces the least, followed by governed bisimulation, then stuttering bisimulation, and governed stuttering bisimulation reducing the most. Apart from a few outliers, there is no difference between the size reductions for stuttering bisimulation and governed stuttering bisimulation in categories other than 'mlsolver'. In that category, governed stuttering bisimulation reduces on average 40% better than stuttering bisimulation.

In Figure 5.8b we can see that in general, it does not pay off to reduce modulo governed stuttering bisimulation before solving, although there is a small number of



(a) Size reduction in %

(b) Speedup factor

Figure 5.8: Box plot of the effects of governed stuttering bisimulation reduction (the diamonds indicate averages, dotted lines are medians). Illustration from [Kei13].

cases in which a dramatic speedup is achieved. The image is slightly distorted because in many cases, the solving times are very small, in which case overhead caused by serialization and disk access may play a significant role, and the inaccuracy of the time measurements may be also influence the results. The analysis in [Kei13] shows that in cases where solving a parity game takes over 5 seconds (either with or without the reduction), the model checking problems are in general solved more quickly by applying governed stuttering bisimulation reduction first, with the median and average both above the 100 mark. For the other categories, however, directly solving the parity game is still faster.

5.5 Conclusion

In this chapter, we presented a lattice of equivalence relations on parity games, with winner equivalence, the coarsest relation, as the top element. In theory, such relations can speed up solving parity games by minimizing the games first using a polynomial quotienting algorithm for one of these relations, and then solving the minimized game. Because current solving algorithms have exponential worst-case complexity, this approach could speed up the solving process.

In practice, solving the original game using the fastest solving algorithms tends to be quicker than performing the reduction first and then solving the result. Only in a few cases a significant speedup can be achieved. The reduction in size however is rather good for most practical cases, with reductions of over 80% for the parity games in categories that represent real-world problems.

For model checking via fixpoint logic, this suggests that it can be interesting to research syntactic transformations on fixpoint logic formulas that approximate the effect of governed stuttering bisimilarity. Currently, fast solvers for fixpoint formulas first generate a parity game, and then solve this game. If an inexpensive syntactic transformation can be found for the fixpoint formula that yields a smaller, but equivalent, parity game, then not only the solving time can be shortened, but also the time needed for generation. As the generation of large parity games is costly both in terms of time and in terms of memory usage, it is interesting to find out whether a syntactic transformation can be found that comes close to the 80% reduction that governed stuttering bisimilarity achieves on parity games.

Chapter 6

Equational Fixpoint Logic

Modal logics such as the first-order modal μ-calculus presented in Chapter 4 reason about very specific models; in this case about Kripke structures. A similar logic that reasons about labelled transition systems was used in Chapter 3 to describe properties about the FlexRay protocol. Both logics closely resemble first-order logic, but with a domain of discourse that is partitioned into two parts; the model under scrutiny (the Kripke structure or labelled transition system) forms one part, for which we only have a restricted form of quantification (the box operator), and the 'data domain' forms the other part (and has no such restriction). The only other difference with first-order logic is the presence of a fixpoint operator.

To evaluate a formula in such a modal logic, a translation can be made to a fixpoint logic that is no longer modal: it does not have the strict separation between model and data, and the restriction on quantification is lifted. Most commonly used as a translation target are the equational fixpoint logic called *Boolean equation systems* (BES) [And92; Mad97] and its first-order extensions, *parameterised Boolean equation systems* (PBES) [GW04] and *predicate equation systems* (PES) [ZC05]. Formulas of these logics consist of sequences of equations that equate propositional (resp. predicate) variables with propositional (predicate) formulas, in which every equation is annotated with a least or greatest fixpoint symbol. The equational syntax allows for compact notation, because identical subformulas can be represented by a single equation.

In fixpoint logic literature not related to model checking, it is more common to find a non-equational fixpoint logic called *least fixpoint logic* (LFP) [Lib04], which is a straightforward extension of first-order logic with a single fixpoint operator. By adding the notion of fixpoint to first-order logic, LFP essentially extends first-order logic in the same way PBES does.

Despite the apparent similarities between the PBES and LFP formalisms, there are two notable differences that make it difficult to relate results about LFP to results about PBES and *vice versa*. Firstly, the PBES formalism explicitly introduces notions of least and greatest fixpoints, and the semantics of PBES is given in terms of these notions. LFP only employs a least fixpoint operator. Secondly, the PBES formalism is equational, LFP is not.

In this chapter we make an effort to overcome these differences in presentation, by defining a logic that is a generalisation of both PBES and LFP. We will look at a logic that we shall call *equational fixpoint logic* (EFL), which has the PBES and LFP

logics as normal forms. EFL can then serve as a unified framework in which results about both sublogics can be compared more easily. Moreover, developing new theories in the setting of EFL will yield results that are applicable in both domains. We will use this in the next chapter, where we will investigate the problem of diagnostics generation for model checking via fixpoint logic. Lastly, the fact that PBES and LFP are equally expressive follows directly from the fact that both are normal forms of EFL. This result could have been obtained from the results in Section 1.4.4 of [AN01], which are presented in the more general setting of vectorial fixpoints on arbitrary lattices. Translating back and forth to this general setting can however be cumbersome (we have done this in a very coarse manner in the complexity proof in Chapter 4), so restating it in our more specific setting may prove to be more insightful to some. Moreover, our proofs are constructive, and therefore suggest ways to translate between the formalisms by syntactic manipulations.

The chapter is structured as follows. In Section 6.1, we present the syntax and semantics of EFL. We then dedicate a section to a number of general lemmas about the semantics of the fixpoint operator in EFL in Section 6.2. We then present a notion of syntactic monotonicity for EFL in Section 6.3, similar to the syntactic restrictions that are usually imposed on LFP and PBES formulas. In Section 6.4, we show that PBES and LFP are indeed fragments of EFL, and that translation from any EFL formula to either of these fragments is always possible, so that PBES and LFP may be seen as normal forms of EFL. We conclude with some closing remarks in Section 6.5.

6.1 Syntax and semantics

Equational fixpoint logic (EFL) is defined as first-order logic extended with a fixpoint equation system operator. This operator works as a binder for second-order variable names taken from some set $\mathcal{X}$. Second-order variable names are usually denoted X, Y, Z, and every second-order variable X has an associated arity. Slightly abusing our own notational conventions, we denote this arity by $\mathsf{ar}(X)$, overloading the arity functions from signatures that we usually also call ar.

Syntax The formulas of EFL over signature $\Sigma = \langle \mathcal{R}, \mathcal{F}, \mathsf{ar} \rangle$ are generated by φ in the grammar below. Let non-terminals X and x_i (for $i \in \mathbb{N}$) generate elements from $\mathcal{X}$ and $\mathcal{V}$, respectively, and let R and f generate elements from $\mathcal{R}$ and $\mathcal{F}$, respectively.

$$\begin{aligned} \varphi, \psi &::= R\, t_0 \dots t_{\mathsf{ar}(R)-1} \mid [X\, t_0 \dots t_{\mathsf{ar}(X)-1} : \mathcal{E}] \mid \neg\varphi \mid \varphi \vee \psi \mid \exists_x\ \varphi \\ t_0, t_1, \dots &::= x \mid f\, t_0 \dots t_{\mathsf{ar}(f)-1} \\ \mathcal{E} &::= \varepsilon \mid (\sigma X x_0 \dots x_{\mathsf{ar}(X)} = \varphi)\mathcal{E} \\ \sigma &::= \mathbf{lfp} \mid \mathbf{gfp} \end{aligned}$$

We also allow the use of some commonly used derived operators:

$$\varphi \wedge \psi \triangleq \neg(\neg\varphi \vee \neg\psi) \qquad \varphi \Rightarrow \psi \triangleq \neg\varphi \vee \psi$$
$$\varphi \Leftrightarrow \psi \triangleq (\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi) \qquad \forall_x\ \varphi \triangleq \neg\exists_x\ \neg\varphi$$

We will call the syntactic structures generated by non-terminal $\mathcal{E}$ *equation systems*. Before we proceed to the semantics of EFL formulas, we will introduce some notational conventions. To increase readability, we will usually omit the trailing ε in equation systems. Sometimes, we will write an equation system in a tabular form, in which every equation is set on a new line, and in which we omit the surrounding parentheses. Parameter lists for relations and second-order variables are often written in vectorial notation: $\bar{t}$ is used rather than $t_0 \dots t_n$, $|\bar{t}|$ denotes the number of elements of such a vector, and if $\bar{t} = t_0, \dots, t_n$, then $\bar{t}_i$ denotes the element t_i.

For $i \in \mathbb{N}$ we denote by $\mathcal{E}^i$ the equation system $\mathcal{E}$ without its first i equations (so in particular, $\mathcal{E}^0 = \mathcal{E}$, and if $\mathcal{E}$ has i equations or less, then $\mathcal{E}^i = \varepsilon$). The formula $[X\bar{t} : \varepsilon]$ may be abbreviated to $X\bar{t}$. The subformula relation $\sqsubseteq$ is extended to EFL in the usual way.

We define the *subsystem ordering* as a partial ordering on equation systems, denoted by $\sqsubseteq$, as it is very similar to the notion of subformula. We recursively define that $\mathcal{E}' \sqsubseteq \mathcal{E}$ holds if and only if there is some i such that:

– $\mathcal{E}' = \mathcal{E}^i$, or
– $\mathcal{E}^i = (\sigma X \bar{x} = \varphi)\mathcal{E}^{i+1}$ and $[Y\bar{t} : \mathcal{E}''] \sqsubseteq \varphi$ such that $\mathcal{E}' \sqsubseteq \mathcal{E}''$, for some $X, Y, \bar{x}, \varphi$ and $\bar{t}$.

A strict variant is defined as $\mathcal{E} \sqsubset \mathcal{E}'$ if and only if $\mathcal{E} \sqsubseteq \mathcal{E}'$ and $\mathcal{E} \neq \mathcal{E}'$.

The set $\mathsf{bnd}(\mathcal{E})$ of *locally bound variables* of an equation system $\mathcal{E}$ is defined by:

$$\mathsf{bnd}(\varepsilon) = \emptyset$$
$$\mathsf{bnd}((\sigma X \bar{x} = \varphi)\mathcal{E}) = \{X\} \cup \mathsf{bnd}(\mathcal{E}).$$

For EFL formulas and equation systems we also define sets $\mathsf{bnd}^*(\varphi)$ and $\mathsf{bnd}^*(\mathcal{E})$ that contain their *bound variables*: $X \in \mathsf{bnd}^*(\varphi)$ if and only if there is some $[Y\bar{t} : \mathcal{E}] \sqsubseteq \varphi$ such that $X \in \mathsf{bnd}(\mathcal{E})$, and $X \in \mathsf{bnd}^*(\mathcal{E})$ if and only if there is some $\mathcal{E}' \sqsubseteq \mathcal{E}$ such that $X \in \mathsf{bnd}(\mathcal{E}')$.

In a formula $\forall_x \varphi$ or $\exists_x \varphi$ we say that x is bound in φ. An equation system $\mathcal{E}$ is also a binder: the variables in $\mathsf{bnd}(\mathcal{E})$ are bound in the right-hand sides of $\mathcal{E}$. If a variable is bound in a formula φ, then it is also bound in all subformulas of φ. The variables occurring in a formula φ that are not bound in φ are called *free*. The set of free variables in a formula φ is denoted $\mathsf{fv}(\varphi)$. The set of free variables $\mathsf{fv}(\mathcal{E})$ in a system $\mathcal{E}$ is defined as the union of the sets of free variables in the right-hand sides in $\mathcal{E}$ (this excludes the variables from $\mathsf{bnd}(\mathcal{E})$).

The *size* $|\mathcal{E}|$ of an equation system is defined as the number of $\mathcal{E}'$ such that $\mathcal{E}' \sqsubset \mathcal{E}$.

An equation system $\mathcal{E}$ induces a strict ordering $<_{\mathcal{E}}$ on $\mathsf{bnd}^*(\mathcal{E})$: if $X, Y \in \mathsf{bnd}^*(\mathcal{E})$, then $X <_{\mathcal{E}} Y$ if and only if there is some $\mathcal{E}' \sqsubset \mathcal{E}$ such that $X \notin \mathsf{bnd}^*(\mathcal{E}') \wedge Y \in \mathsf{bnd}^*(\mathcal{E}')$. The reflexive closure of $<_{\mathcal{E}}$ is denoted $\leq_{\mathcal{E}}$ and is used as a partial order on $\mathcal{X}$.

To increase readability in the rest of the section[1], we impose two syntactic restrictions on EFL formulas. First, we require that for every equation $(\sigma X \bar{x} = \varphi)$ in an equation system, $\mathsf{fv}(\varphi) \subseteq \mathcal{X} \cup \{x \mid x \in \bar{x}\}$. Any formula can be altered to meet this restriction by increasing the arity of the bound second-order variables in the formula,

[1] As best we can; the notation will get rather hairy even with these restrictions.

without changing the semantics of the formula (the semantics of EFL is described below, and the proof for this claim is left as an exercise to the reader).

The second restriction is that we require bound variables to be *uniquely named*. That is, in an EFL formula φ,

- $\mathsf{bnd}^*(\varphi) \cap \mathsf{fv}(\varphi) = \emptyset$,
- for every equation $(\sigma X \bar{x} = \psi)$ in φ, $X \notin \mathsf{bnd}^*(\psi)$, and
- for every subformula of the form $\chi \vee \psi$, $\mathsf{bnd}^*(\chi) \cap \mathsf{bnd}^*(\psi) = \emptyset$ (and similarly for the derived binary operators).

This restriction can always be satisfied by renaming bound variables in a formula. With this restriction in place, we do not have to worry about name clashes in our proofs. In particular, we use this restriction so that we may write $\sigma_X, \bar{x}_X$ and φ_X to denote the fixpoint operator, variable vector and right-hand side of the equation that binds X in $\mathcal{E}$, if $\mathcal{E}$ is clear from the context.

Semantics A formula is evaluated on a structure $\mathfrak{A} = \langle \Sigma, A, \mathcal{I} \rangle$ and an environment θ which maps every $x \in \mathcal{V}$ to an element of A, and which maps every $X \in \mathcal{X}$ to a subset of $A^{\mathsf{ar}(X)}$. The semantics of EFL formulas is now very similar to that of formulas of first-order logic; the only differences are that θ also includes second-order variables in its domain, and that there is an additional way to construct an 'atomic' formula by means of the fixpoint equation system operator.

$$
\begin{array}{ll}
\mathfrak{A}, \theta \models R\bar{t} & \text{iff } \bar{t}^{\mathfrak{A},\theta} \in \mathcal{I}(R) \\
\mathfrak{A}, \theta \models [X\bar{t} : \mathcal{E}] & \text{iff } \bar{t}^{\mathfrak{A},\theta} \in \mathbf{S}^{\mathfrak{A}}(\theta, \mathcal{E})(X), \text{ with } \mathbf{S}^{\mathfrak{A}} \text{ as defined below} \\
\mathfrak{A}, \theta \models \neg\varphi & \text{iff } \mathfrak{A}, \theta \models \varphi \text{ does not hold} \\
\mathfrak{A}, \theta \models \varphi \vee \psi & \text{iff } \mathfrak{A}, \theta \models \varphi \text{ or } \mathfrak{A}, \theta \models \psi \\
\mathfrak{A}, \theta \models \exists_x \varphi & \text{iff } \mathfrak{A}, \theta[x \mapsto a] \models \varphi \text{ for some } a \in A
\end{array}
$$

The function $\mathbf{S}^{\mathfrak{A}}$ calculates the semantic relation associated with the second-order variable X in a system $\mathcal{E}$, in the context of a structure $\mathfrak{A}$ and an environment θ.

Definition 6.1. *The* solution *of an equation system $\mathcal{E}$ under environment θ in structure $\mathfrak{A}$, denoted $\mathbf{S}^{\mathfrak{A}}(\theta, \mathcal{E})$ is defined inductively on the structure of $\mathcal{E}$ as follows:*

$$
\mathbf{S}^{\mathfrak{A}}(\theta, \mathcal{E}) = \begin{cases} \theta, & \mathcal{E} = \varepsilon, \\ \mathbf{S}^{\mathfrak{A}}(\theta[X \mapsto \sigma\, \mathbf{T}_{\mathcal{E}}^{\mathfrak{A},\theta}], \mathcal{E}^1), & \mathcal{E} = (\sigma X \bar{x} = \varphi)\mathcal{E}^1, \end{cases}
$$

in which the mapping $\mathbf{T}_{\mathcal{E}}^{\mathfrak{A},\theta}$ is called a predicate transformer *and is defined as follows:*

$$
\mathbf{T}^{\mathfrak{A},\theta}_{(\sigma X \bar{x} = \varphi)\mathcal{E}^1}(\mathcal{R}) = \{\bar{a} \mid \mathfrak{A}, \mathbf{S}^{\mathfrak{A}}(\theta[X \mapsto \mathcal{R}], \mathcal{E}^1)[\bar{x} \mapsto \bar{a}] \models \varphi\}.
$$

For $\mathcal{E} \neq \varepsilon$, $\mathbf{S}^{\mathfrak{A}}(\theta, \mathcal{E})$ is not defined if $\mathbf{T}_{\mathcal{E}}^{\mathfrak{A},\theta}$ is not monotone, or if $\mathbf{S}^{\mathfrak{A}}(\theta[X \mapsto \sigma\, \mathbf{T}_{\mathcal{E}}^{\mathfrak{A},\theta}], \mathcal{E}^1)$ is not defined.

If it is clear from the context which structure is used, then the $\mathfrak{A}$ superscripts may be left out to increase readability.

We introduce shorthand notation to say that two environments are identical for all variables in some set S: by $\theta \equiv_S \theta'$ we denote that $\forall_{x \in S}\ \theta(x) = \theta'(x)$.

Our definition of semantics is only defined if all predicate transformers used in the definition are monotone, because for non-monotone functions it is not guaranteed that the required fixpoint exists. As monotonicity of this predicate transformer is undecidable in general, we would like a syntactic restriction of EFL so that the resulting predicate transformers are always monotone (this is a common practice that we also find in the setting of the LFP and PBES fixpoint logics). Because our notion of syntactic monotonicity for EFL is easier to prove correct after establishing some basic facts about the solutions of equation systems, we postpone its treatment until Section 6.3.

Example. In Chapter 5 we introduced the notation $v \xmapsto{U} T$ to denote that there is a path from v to a node in T, such that only nodes from U are visited before T is reached. We gave a textual, inductive definition of this predicate. Using EFL, this definition can be expressed as:

$$v \xmapsto{U} T \triangleq [Xv : (\mathbf{lfp}\, Xu = \exists_{u'}\ u \to u' \wedge (u' \in T \vee (u' \in U \wedge Xu')))].$$

Likewise, divergence can be expressed using a single greatest fixpoint, corresponding to the coinductive definition of $v \xmapsto{U}$:

$$v \xmapsto{U} \triangleq [Xv : (\mathbf{gfp}\, Xu = \exists_{u'}\ u \to u' \wedge u' \in U \wedge Xu')].$$

■

6.2 Solution

The solution of an equation system is by far the most interesting part of the EFL semantics. Although the mutual recursion of $\mathbf{S}(\theta, \mathcal{E})$ and $\mathbf{T}^{\theta}_{\mathcal{E}}$ makes it difficult to read and understand the definition, we need a good understanding of the notion of solution to establish the results in the sections to come. This section is therefore devoted to a number of lemmas that describe some interesting properties of the **S** function that can be used in subsequent proofs.

The following lemmas all quantify universally over models $\mathfrak{A}$, environments θ and equation systems $\mathcal{E}$. The first lemma is almost trivial, but is used quite often in the proofs that follow. It shows why the validity of $[X\bar{t} : \mathcal{E}]$ in $\mathfrak{A}$ and θ does not depend on $\mathcal{E}$ (but only on θ) if $\mathcal{E}$ does not bind X locally. This is also the reason to allow abbreviating $[X\bar{t} : \varepsilon]$ to $X\bar{t}$: the empty equation system has no effect on the truth value of this formula.

Lemma 6.1. $\mathbf{S}(\theta, \mathcal{E})(X) = \theta(X)$ *for all* $X \notin \mathsf{bnd}(\mathcal{E})$.

Proof. The lemma follows directly from the fact that **S** only updates θ for second-order variables that are bound by $\mathcal{E}$. □

The second lemma says that only the values in θ for variables that are free in $\mathcal{E}$ can affect the solution of $\mathcal{E}$ under θ.

Lemma 6.2. *For all θ', if $\theta \equiv_{\mathsf{fv}(\mathcal{E})} \theta'$, then $\mathbf{S}(\theta, \mathcal{E}) \equiv_{\mathsf{bnd}(\mathcal{E})} \mathbf{S}(\theta', \mathcal{E})$.*

Proof. We proceed by induction on the structure of $\mathcal{E}$. For $\mathcal{E} = \varepsilon$, the lemma holds trivially, because $\eta \equiv_\emptyset \eta'$ for any η and η'. So let $\mathcal{E} = (\sigma X \bar{x} = \varphi)\mathcal{E}'$ and assume now as the induction hypothesis that for all $Y \in \mathsf{bnd}(\mathcal{E}')$, η and η' such that $\eta \equiv_{\mathsf{fv}(\mathcal{E}')} \eta'$, we have that $\mathbf{S}(\eta, \mathcal{E}')(Y) = \mathbf{S}(\eta', \mathcal{E}')(Y)$. We need to prove for all $Y \in \mathsf{bnd}(\mathcal{E})$ that $\mathbf{S}(\theta, \mathcal{E})(Y) = \mathbf{S}(\theta', \mathcal{E})(Y)$, which, expanding the definition of S, is equivalent to showing that

$$\mathbf{S}(\theta[X \mapsto \sigma(\mathbf{T}^{\theta}_{\mathcal{E}})], \mathcal{E}')(Y) = \mathbf{S}(\theta'[X \mapsto \sigma(\mathbf{T}^{\theta'}_{\mathcal{E}})], \mathcal{E}')(Y). \tag{$*$}$$

So fix $Y \in \mathsf{bnd}(\mathcal{E})$ and distinguish two cases.

Case $Y = X$. Fix some θ and θ' such that $\theta \equiv_{\mathsf{fv}(\mathcal{E})} \theta'$. Filling in X for Y in ($*$), and using Lemma 6.1 to simplify, our proof obligation becomes:

$$\theta[X \mapsto \sigma(\mathbf{T}^{\theta}_{\mathcal{E}})](X) = \theta'[X \mapsto \sigma(\mathbf{T}^{\theta'}_{\mathcal{E}})](X).$$

We therefore need to prove that $\sigma(\mathbf{T}^{\theta}_{\mathcal{E}}) = \sigma(\mathbf{T}^{\theta'}_{\mathcal{E}})$. We set out to prove that $\mathbf{T}^{\theta}_{\mathcal{E}} = \mathbf{T}^{\theta'}_{\mathcal{E}}$, by showing that, given some R and $\bar{a}$ matching the arity of X,

$$\mathfrak{A}, \mathbf{S}(\theta[X \mapsto R], \mathcal{E}')[\bar{x} \mapsto \bar{a}] \models \varphi \text{ iff } \mathfrak{A}, \mathbf{S}(\theta'[X \mapsto R], \mathcal{E}')[\bar{x} \mapsto \bar{a}] \models \varphi.$$

This must be the case because $\mathbf{S}(\theta[X \mapsto R], \mathcal{E}') = \mathbf{S}(\theta'[X \mapsto R], \mathcal{E}')$, which follows directly from the induction hypothesis because $\theta'[X \mapsto R] \equiv_{\mathsf{fv}(\mathcal{E}')} \theta[X \mapsto R]$.

Case $Y \neq X$. We have just shown that $\mathbf{T}^{\theta}_{\mathcal{E}} = \mathbf{T}^{\theta'}_{\mathcal{E}}$, and so it follows trivially that $\theta'[X \mapsto \sigma(\mathbf{T}^{\theta'}_{\mathcal{E}})] \equiv_{\mathsf{fv}(\mathcal{E}')} \theta[X \mapsto \sigma(\mathbf{T}^{\theta}_{\mathcal{E}})]$. The induction hypothesis therefore gives us the desired result. □

We often apply the lemma above when we know that θ and θ' are equal on more than just $\mathsf{fv}(\mathcal{E})$, and in situations where we would like to prove that $\mathbf{S}(\theta, \mathcal{E})$ and $\mathbf{S}(\theta', \mathcal{E})$ are identical for *every* variable, not just for $\mathsf{bnd}(\mathcal{E})$. For these situations we state a slightly more specific lemma.

Lemma 6.3. *$\mathbf{S}(\theta, \mathcal{E}) = \mathbf{S}(\theta', \mathcal{E})$ if $\theta \equiv_{\mathcal{V} \cup \mathcal{X} \setminus \mathsf{bnd}(\mathcal{E})} \theta'$.*

Proof. This follows immediately from Lemmas 6.1 and 6.2. □

Using these results, we can prove the next lemma, which says that evaluating a subsystem of $\mathcal{E}$ in the environment $\mathbf{S}(\theta, \mathcal{E})$ (the solution of $\mathcal{E}$ under θ) yields that same environment. This was already shown by Mader in the setting of fixpoint equation systems as Corollary 3.7 in [Mad97].

Lemma 6.4. *$\mathbf{S}(\mathbf{S}(\theta, \mathcal{E}), \mathcal{E}^i) = \mathbf{S}(\theta, \mathcal{E})$ for all i.*

Proof. The proof is by induction on i. First consider the base case $i = 0$. Note that by Lemma 6.1, $\mathbf{S}(\theta, \mathcal{E})$ and θ agree on all $X \notin \mathsf{bnd}(\mathcal{E}^0)$. By Lemma 6.3, we then have that $\mathbf{S}(\mathbf{S}(\theta, \mathcal{E}), \mathcal{E}^0) = \mathbf{S}(\theta, \mathcal{E})$.

Now let $i > 0$, let $\mathcal{E}^{i-1} = (\sigma X \bar{x} = \varphi)\mathcal{E}^i$ and as our induction hypothesis, assume that we have $\mathbf{S}(\mathbf{S}(\theta, \mathcal{E}), \mathcal{E}^{i-1}) = \mathbf{S}(\theta, \mathcal{E})$. Filling in the definition of $\mathbf{S}$, we get (for $F = \sigma\, \mathbf{T}^{\mathbf{S}(\theta,\mathcal{E})}_{\mathcal{E}^{i-1}}$):

$$\mathbf{S}(\theta, \mathcal{E}) = \mathbf{S}(\mathbf{S}(\theta, \mathcal{E}), \mathcal{E}^{i-1}) = \mathbf{S}(\mathbf{S}(\theta, \mathcal{E})[X \mapsto F], \mathcal{E}^i), \text{ and by Lemma 6.1,}$$

$$\mathbf{S}(\mathbf{S}(\theta, \mathcal{E})[X \mapsto F], \mathcal{E}^i)(X) = \mathbf{S}(\theta, \mathcal{E})[X \mapsto F](X) = F.$$

Therefore also $\mathbf{S}(\theta, \mathcal{E})(X) = F$, so $\mathbf{S}(\theta, \mathcal{E})[X \mapsto F] = \mathbf{S}(\theta, \mathcal{E})$. In the above we saw that $\mathbf{S}(\mathbf{S}(\theta, \mathcal{E})[X \mapsto F], \mathcal{E}^i) = \mathbf{S}(\theta, \mathcal{E})$, so it follows that $\mathbf{S}(\mathbf{S}(\theta, \mathcal{E}), \mathcal{E}^i) = \mathbf{S}(\theta, \mathcal{E})$. □

The following two lemmas say something about the solution of a particular variable in an equation system. The first lemma shows that it is indeed still a fixpoint of the associated predicate transformer when evaluated under the environment $\mathbf{S}(\theta, \mathcal{E})$. This should come as no surprise, as the solution, say R, for a variable X is computed as a fixpoint of the associated predicate transformer, under an environment θ for which $\theta(X)$ approaches R during the fixpoint approximation.

Lemma 6.5. *Let $\mathcal{E}^i = (\sigma X \bar{x} = \varphi)\mathcal{E}'$ for some i, then $\mathbf{S}(\theta, \mathcal{E})(X) = \sigma\, \mathbf{T}^{\mathbf{S}(\theta,\mathcal{E})}_{\mathcal{E}^i}$.*

Proof. By Lemma 6.4, $\mathbf{S}(\theta, \mathcal{E})(X) = \mathbf{S}(\mathbf{S}(\theta, \mathcal{E}), \mathcal{E}^i)(X)$. Expanding the definition of $\mathbf{S}$, $\mathbf{S}(\theta, \mathcal{E})(X) = \mathbf{S}(\mathbf{S}(\theta, \mathcal{E})[X \mapsto \sigma\, \mathbf{T}^{\mathbf{S}(\theta,\mathcal{E})}_{\mathcal{E}^i}], \mathcal{E}')(X)$. By Lemma 6.1, we have $\mathbf{S}(\theta, \mathcal{E})(X) = \mathbf{S}(\theta, \mathcal{E})[X \mapsto \sigma\, \mathbf{T}^{\mathbf{S}(\theta,\mathcal{E})}_{\mathcal{E}^i}](X)$, and therefore $\mathbf{S}(\theta, \mathcal{E})(X) = \sigma\, \mathbf{T}^{\mathbf{S}(\theta,\mathcal{E})}_{\mathcal{E}^i}$. □

The second lemma says that checking whether $\bar{a}$ is an element of R is equivalent to checking that the right-hand side of the equation for X holds under the solution of the system, if $\bar{x}$ is mapped to $\bar{a}$. One would expect this to be true, because R is computed in every step of the fixpoint approximation as the set of vectors $\bar{b}$ such that the right-hand side holds under the current approximation

Lemma 6.6. *Let $\mathcal{E}^i = (\sigma X \bar{x} = \varphi)\mathcal{E}'$ for some i, then*

$$\bar{a} \in \mathbf{S}^{\mathfrak{A}}(\theta, \mathcal{E})(X) \text{ if and only if } \mathfrak{A}, \mathbf{S}(\theta, \mathcal{E})[\bar{x} \mapsto \bar{a}] \models \varphi.$$

Proof. By Lemma 6.5, $\mathbf{S}(\theta, \mathcal{E})(X) = \sigma\, \mathbf{T}^{\theta}_{\mathcal{E}}$, which, because it is a fixpoint, is equal to $\mathbf{T}^{\theta}_{\mathcal{E}}(\sigma\, \mathbf{T}^{\theta}_{\mathcal{E}})$, which by Definition 6.1 equals $\{\bar{c} \mid \mathfrak{A}, \mathbf{S}(\mathbf{S}(\theta, \mathcal{E}), \mathcal{E}')[\bar{x} \mapsto \bar{c}] \models \varphi\}$. Therefore $\bar{a} \in \mathbf{S}^{\mathfrak{A}}(\theta, \mathcal{E})(X)$ is equivalent to $\mathfrak{A}, \mathbf{S}(\mathbf{S}(\theta, \mathcal{E}), \mathcal{E}')[\bar{x} \mapsto \bar{a}] \models \varphi$, which by Lemma 6.4 is equivalent to $\mathfrak{A}, \mathbf{S}(\theta, \mathcal{E})[\bar{x} \mapsto \bar{a}] \models \varphi$. □

The following lemma shows that if a system $\mathcal{E}$ can be split into two, *i.e.*, $\mathcal{E} = \mathcal{E}_1\mathcal{E}_2$, such that $\mathcal{E}_1$ is in some sense independent of the other, then the solution of $\mathcal{E}_1$ can be calculated independently of $\mathcal{E}_2$, after which it suffices to calculate the solution of $\mathcal{E}_2$ in the environment that is the solution of $\mathcal{E}_1$ to obtain the solution of $\mathcal{E}$. Note that this modularity gives rise to a polynomial solving algorithm if $\mathcal{E}_1$ and $\mathcal{E}_2$ are solvable in polynomial time. Given that systems in which all equations have the same fixpoint symbol are computable in polynomial time, this lemma therefore also suggests a straightforward proof for the fact that model checking L_μ formulas of alternation depth 1 (using Niwiński's definition, see *e.g.*, [BS06] for a comprehensive treatment) can be done in polynomial time.

Lemma 6.7. *If* $\mathcal{E} = \mathcal{E}_1\mathcal{E}_2$ *and* $\mathsf{fv}(\mathcal{E}_1) \cap \mathsf{bnd}(\mathcal{E}_2) = \emptyset$*, then* $\mathbf{S}(\theta, \mathcal{E}) = \mathbf{S}(\mathbf{S}(\theta, \mathcal{E}_1), \mathcal{E}_2)$.

Proof. The proof goes by induction on the size of $\mathcal{E}_1$. If $\mathcal{E}_1 = \varepsilon$, the lemma follows trivially. So assume as the induction hypothesis that the lemma holds for all $\mathcal{E}' = \mathcal{E}_1'\mathcal{E}_2'$ with $\mathcal{E}_1'$ smaller than $\mathcal{E}_1$.

Let $\mathcal{E}_1 = (\sigma X \bar{x} = \varphi)\mathcal{E}_1'$. We first show that $\mathbf{T}^{\mathfrak{A},\theta}_{\mathcal{E}_1} = \mathbf{T}^{\mathfrak{A},\theta}_{\mathcal{E}_1\mathcal{E}_2}$:

$$\begin{aligned}
\mathbf{T}^{\mathfrak{A},\theta}_{\mathcal{E}_1}(R) &= \{\bar{a} \mid \mathfrak{A}, \mathbf{S}(\theta[X \mapsto R], \mathcal{E}_1')[\bar{x} \mapsto \bar{a}] \models \varphi\} && \text{(Definition 6.1)}\\
&= \{\bar{a} \mid \mathfrak{A}, \mathbf{S}(\mathbf{S}(\theta[X \mapsto R], \mathcal{E}_1'), \mathcal{E}_2)[\bar{x} \mapsto \bar{a}] \models \varphi\} && (*)\\
&= \{\bar{a} \mid \mathfrak{A}, \mathbf{S}(\theta[X \mapsto R], \mathcal{E}_1'\mathcal{E}_2)[\bar{x} \mapsto \bar{a}] \models \varphi\} && \text{(IH)}\\
&= \mathbf{T}^{\mathfrak{A},\theta}_{\mathcal{E}_1\mathcal{E}_2}(R) && \text{(Definition 6.1)}
\end{aligned}$$

The step at $(*)$ is valid because $\mathsf{fv}(\varphi) \cap \mathsf{bnd}(\mathcal{E}_2) = \emptyset$, and Lemma 6.1. We can now finish our proof.

$$\begin{aligned}
\mathbf{S}(\mathbf{S}(\theta, \mathcal{E}_1), \mathcal{E}_2) &= \mathbf{S}(\mathbf{S}(\theta[X \mapsto \sigma\, \mathbf{T}^{\mathfrak{A},\theta}_{\mathcal{E}_1}], \mathcal{E}_1'), \mathcal{E}_2) && \text{(Definition 6.1)}\\
&= \mathbf{S}(\theta[X \mapsto \sigma\, \mathbf{T}^{\mathfrak{A},\theta}_{\mathcal{E}_1}], \mathcal{E}_1'\mathcal{E}_2) && \text{(IH)}\\
&= \mathbf{S}(\theta[X \mapsto \sigma\, \mathbf{T}^{\mathfrak{A},\theta}_{\mathcal{E}_1\mathcal{E}_2}], \mathcal{E}_1'\mathcal{E}_2)\\
&= \mathbf{S}(\theta, \mathcal{E}_1\mathcal{E}_2) && \text{(Definition 6.1)}
\end{aligned}$$

□

Something similar can be shown when $\mathcal{E}_1$ only depends on $\mathcal{E}_2$, but not the other way around.

Lemma 6.8. *If* $\mathcal{E} = \mathcal{E}_1\mathcal{E}_2$ *and* $\mathsf{fv}(\mathcal{E}_2) \cap \mathsf{bnd}(\mathcal{E}_1) = \emptyset$*, then* $\mathbf{S}(\theta, \mathcal{E}) = \mathbf{S}(\mathbf{S}(\theta, \mathcal{E}_2), \mathcal{E}_1)$.

Proof. The proof for this lemma is similar to that of the previous lemma, and again goes by induction on the size of $\mathcal{E}_1$. If $\mathcal{E}_1 = \varepsilon$, the lemma holds trivially. Assume as the induction hypothesis that the lemma holds for $\mathcal{E}_1'$ and $\mathcal{E}_2'$ with $\mathcal{E}_1'$ smaller than $\mathcal{E}_1$.

Let $\mathcal{E}_1 = (\sigma X \bar{x} = \varphi)\mathcal{E}_1'$. We first show that $\mathbf{T}^{\mathfrak{A},\mathbf{S}(\theta,\mathcal{E}_2)}_{\mathcal{E}_1} = \mathbf{T}^{\mathfrak{A},\theta}_{\mathcal{E}_1\mathcal{E}_2}$:

$$\begin{aligned}
\mathbf{T}^{\mathfrak{A},\mathbf{S}(\theta,\mathcal{E}_2)}_{\mathcal{E}_1}(R) &= \{\bar{a} \mid \mathfrak{A}, \mathbf{S}(\mathbf{S}(\theta, \mathcal{E}_2)[X \mapsto R], \mathcal{E}_1')[\bar{x} \mapsto \bar{a}] \models \varphi\} && \text{(Definition 6.1)}\\
&= \{\bar{a} \mid \mathfrak{A}, \mathbf{S}(\mathbf{S}(\theta[X \mapsto R], \mathcal{E}_2), \mathcal{E}_1')[\bar{x} \mapsto \bar{a}] \models \varphi\} && (*)\\
&= \{\bar{a} \mid \mathfrak{A}, \mathbf{S}(\theta[X \mapsto R], \mathcal{E}_1'\mathcal{E}_2)[\bar{x} \mapsto \bar{a}] \models \varphi\} && \text{(IH)}\\
&= \mathbf{T}^{\mathfrak{A},\theta}_{\mathcal{E}_1\mathcal{E}_2}(R) && \text{(Definition 6.1)}
\end{aligned}$$

The step at $(*)$ is valid because by Lemma 6.2, $\mathbf{S}(\theta, \mathcal{E}_2)$ and $\mathbf{S}(\theta[X \mapsto R], \mathcal{E}_2)$ are equal on all variables in $\mathsf{bnd}(\mathcal{E}_2)$, and by Lemma 6.1, they are equal on all variables outside $\mathsf{bnd}(\mathcal{E}_2)$, except X. Because $\mathbf{S}(\theta[X \mapsto R], \mathcal{E}_2)(X) = R$ by Lemma 6.1, we may conclude that $\mathbf{S}(\theta[X \mapsto R], \mathcal{E}_2) = \mathbf{S}(\theta, \mathcal{E}_2)[X \mapsto R]$. We can now finish the proof. Below, the

same reasoning is used at the $(*)$ mark as above.

$$\begin{aligned}
\mathbf{S}(\mathbf{S}(\theta,\mathcal{E}_2),\mathcal{E}_1) &= \mathbf{S}(\mathbf{S}(\theta,\mathcal{E}_2)[X \mapsto \sigma\,\mathbf{T}^{\mathfrak{A},\mathbf{S}(\theta,\mathcal{E}_2)}_{\mathcal{E}_1}],\mathcal{E}_1') && \text{(Definition 6.1)}\\
&= \mathbf{S}(\mathbf{S}(\theta[X \mapsto \sigma\,\mathbf{T}^{\mathfrak{A},\mathbf{S}(\theta,\mathcal{E}_2)}_{\mathcal{E}_1}],\mathcal{E}_2),\mathcal{E}_1') && (*)\\
&= \mathbf{S}(\theta[X \mapsto \sigma\,\mathbf{T}^{\mathfrak{A},\mathbf{S}(\theta,\mathcal{E}_2)}_{\mathcal{E}_1}],\mathcal{E}_1'\mathcal{E}_2) && \text{(IH)}\\
&= \mathbf{S}(\theta[X \mapsto \sigma\,\mathbf{T}^{\mathfrak{A},\theta}_{\mathcal{E}_1\mathcal{E}_2}],\mathcal{E}_1'\mathcal{E}_2)\\
&= \mathbf{S}(\theta,\mathcal{E}_1\mathcal{E}_2) && \text{(Definition 6.1)}
\end{aligned}$$ □

The following two lemmas show that if in a system $\mathcal{E} = \mathcal{E}_1\mathcal{E}_2$, we are interested in the solution of a variable in one of the two subsystems, and that subsystem is independent of the other, then we only need to calculate the solution for the subsystem in which the variable is bound.

Lemma 6.9. *If $\mathcal{E} = \mathcal{E}_1\mathcal{E}_2$ such that $\mathsf{fv}(\mathcal{E}_1) \cap \mathsf{bnd}(\mathcal{E}_2) = \emptyset$, then $\mathbf{S}(\theta,\mathcal{E})(X) = \mathbf{S}(\theta,\mathcal{E}_1)(X)$ for all $X \in \mathsf{bnd}(\mathcal{E}_1)$ and all θ.*

Proof. By Lemma 6.7, $\mathbf{S}(\theta,\mathcal{E}) = \mathbf{S}(\mathbf{S}(\theta,\mathcal{E}_1),\mathcal{E}_2)$, and then by Lemma 6.1 we can derive $\mathbf{S}(\mathbf{S}(\theta,\mathcal{E}_1),\mathcal{E}_2)(X) = \mathbf{S}(\theta,\mathcal{E}_1)(X)$ for all $X \in \mathsf{bnd}(\mathcal{E}_1)$, since $\mathsf{bnd}(\mathcal{E}_1) \cap \mathsf{bnd}(\mathcal{E}_2) = \emptyset$ by the unique naming convention. □

Lemma 6.10. *If $\mathcal{E} = \mathcal{E}_1\mathcal{E}_2$ such that $\mathsf{fv}(\mathcal{E}_2) \cap \mathsf{bnd}(\mathcal{E}_1) = \emptyset$, then $\mathbf{S}(\theta,\mathcal{E})(X) = \mathbf{S}(\theta,\mathcal{E}_2)(X)$ for all $X \in \mathsf{bnd}(\mathcal{E}_2)$ and all θ.*

Proof. This can be shown by a straightforward induction on the number of equations in $\mathcal{E}_1$. If $\mathcal{E}_1 = \varepsilon$, then the lemma holds vacuously, because then $\mathcal{E} = \mathcal{E}_2$. Assume as the induction hypothesis that $\mathbf{S}(\theta,\mathcal{E}_1'\mathcal{E}_2)(X) = \mathbf{S}(\theta,\mathcal{E}_2)(X)$ for all $X \in \mathsf{bnd}(\mathcal{E}_2)$ and all $\mathcal{E}_1'$ smaller than $\mathcal{E}_1$ such that $\mathsf{fv}(\mathcal{E}_2) \cap \mathsf{bnd}(\mathcal{E}_1) = \emptyset$. Now suppose that $\mathcal{E}_1 = (\sigma\, Y\, \bar{y} = \varphi)\mathcal{E}_1'$. Then $\mathbf{S}(\theta,\mathcal{E})(X) = \mathbf{S}(\theta[Y \mapsto \sigma\,\mathbf{T}^{\mathfrak{A},\theta}_{\mathcal{E}}],\mathcal{E}_1'\mathcal{E}_2)(X)$. By the induction hypothesis this is equal to $\mathbf{S}(\theta[Y \mapsto \sigma\,\mathbf{T}^{\mathfrak{A},\theta}_{\mathcal{E}}],\mathcal{E}_2)(X)$, which in turn is equal to $\mathbf{S}(\theta,\mathcal{E}_2)(X)$ by Lemma 6.2. □

For completeness, we also give the generalisation of Lemma 3.10 in [Mad97], which can now easily be proven using the previous lemmas.

Lemma 6.11. *If $\mathcal{E} = \mathcal{E}_1\mathcal{E}_2$ such that $\mathsf{fv}(\mathcal{E}_1) \cap \mathsf{bnd}(\mathcal{E}_2) = \emptyset$ and $\mathsf{fv}(\mathcal{E}_2) \cap \mathsf{bnd}(\mathcal{E}_1) = \emptyset$, then $\mathbf{S}(\theta,\mathcal{E}) = \mathbf{S}(\mathbf{S}(\theta,\mathcal{E}_2),\mathcal{E}_1)$.*

Proof. By Lemma 6.1, $\mathbf{S}(\theta,\mathcal{E})(X) = \theta(X) = \mathbf{S}(\theta,\mathcal{E}_2)(X) = \mathbf{S}(\mathbf{S}(\theta,\mathcal{E}_2),\mathcal{E}_1)(X)$ for all $X \notin \mathsf{bnd}(\mathcal{E})$. For $X \in \mathsf{bnd}(\mathcal{E}_1)$ note that by Lemma 6.2, $\mathbf{S}(\mathbf{S}(\theta,\mathcal{E}_2),\mathcal{E}_1)(X) = \mathbf{S}(\theta,\mathcal{E}_1)(X)$, and then by Lemma 6.9 this is equal to $\mathbf{S}(\theta,\mathcal{E})(X)$. For $X \in \mathsf{bnd}(\mathcal{E}_2)$, by Lemma 6.1, $\mathbf{S}(\mathbf{S}(\theta,\mathcal{E}_2),\mathcal{E}_1)(X) = \mathbf{S}(\theta,\mathcal{E}_2)(X)$, and then by Lemma 6.10, this can now be seen to be equal to $\mathbf{S}(\theta,\mathcal{E})(X)$. □

6.3 Monotonicity

Definition 6.1 computes fixpoints over predicate transformers. Such fixpoints are only guaranteed to exist when the predicate transformers are monotone. We will call a system $\mathcal{E}$ *semantically monotone* if $\mathbf{S}(\theta, \mathcal{E})$ is defined for all θ, which is the same as saying that all predicate transformers that are used in the computation of $\mathbf{S}(\theta, \mathcal{E})$ are monotone.

In general, the monotonicity property of the predicate transformers is undecidable, so we would like to define a syntactic restriction of our EFL formulas that is guaranteed to only give rise to monotone predicate transformers. Of course, determining whether a given formula adheres to the restriction should be computable in a reasonable time; we give a definition that is easily seen to be computable in linear time with respect to the size of the formula.

The following definition of syntactic monotonicity for EFL could be informally stated as 'any bound second-order variable only occurs positively in the right-hand side of its defining equation, and in the equations reachable via its defining equation'.

Definition 6.2. *An equation system $\mathcal{E}$ is* syntactically monotone *if and only if for all $X \in \mathsf{bnd}(\mathcal{E})$, there exist functions $\mathsf{pos}_{\mathcal{E},X}$ and $\mathsf{neg}_{\mathcal{E},X}$ satisfying the equalities denoted in the table below, such that $\mathsf{pos}_{\mathcal{E},X}(\varphi_X)$. An EFL formula φ is syntactically monotone if and only if for every subformula $[Y\,\bar{t} : \mathcal{F}] \sqsubseteq \varphi$, $\mathcal{F}$ is syntactically monotone.*

χ	$\mathsf{pos}_{\mathcal{E},X}(\chi)$	$\mathsf{neg}_{\mathcal{E},X}(\chi)$	
$[Y\,\bar{t} : \mathcal{F}]$	$\mathbb{t}$,	$\mathbb{t}$,	$Y <_{\mathcal{E}} X$
	$\mathbb{t}$,	$\mathbb{f}$,	$Y = X$
	$\mathsf{pos}_{\mathcal{E},X}(\varphi_Y)$,	$\mathsf{neg}_{\mathcal{E},X}(\varphi_Y)$,	$Y \not\leq_{\mathcal{E}} X$
$R\bar{t}$	$\mathbb{t}$	$\mathbb{t}$	
$\neg\psi$	$\mathsf{neg}_{\mathcal{E},X}(\psi)$	$\mathsf{pos}_{\mathcal{E},X}(\psi)$	
$\exists_x\, \psi$	$\mathsf{pos}_{\mathcal{E},X}(\psi)$	$\mathsf{neg}_{\mathcal{E},X}(\psi)$	
$\psi_0 \vee \psi_1$	$\mathsf{pos}_{\mathcal{E},X}(\psi_0) \wedge \mathsf{pos}_{\mathcal{E},X}(\psi_1)$	$\mathsf{neg}_{\mathcal{E},X}(\psi_0) \wedge \mathsf{neg}_{\mathcal{E},X}(\psi_1)$	

We assume that for each monotone $\mathcal{E}$ and $X \in \mathsf{bnd}(\mathcal{E})$, $\mathsf{pos}_{\mathcal{E},X}$ and $\mathsf{neg}_{\mathcal{E},X}$ are uniquely determined (for instance by defining them to be the least functions satisfying the restrictions above, with respect to some arbitrary ordering on functions).

Although the definition is somewhat unwieldy, syntactic monotonicity is conceptually quite simple. Imagine an edge-labelled graph with the bound variables of a system as vertices, with an edge labelled 'positive' from X to Y if Y occurs in φ_X in the scope of an even number of negations, and with an edge labelled 'negative' from X to Y if Y occurs in φ_X in the scope of an odd number of negations. The system is syntactically monotone if in this graph, every cycle contains an even number of edges labelled 'negative'.

Note that we need not uniquely define $\mathsf{pos}_{\mathcal{E},X}$ and $\mathsf{neg}_{\mathcal{E},X}$; the existence of such predicates is a sufficient condition to prove semantic monotonicity.

Example. The following formula is syntactically monotone.

$$[X : \mathbf{lfp}\, X = \neg Y \vee [Z : \mathbf{lfp}\, Z = Z] \\ \mathbf{gfp}\, Y = \neg X \wedge Y \wedge \neg[W : \mathbf{gfp}\, W = W \wedge \neg Y\,]]$$

Removing any single negation symbol from this formula will cause it not to be syntactically monotone anymore. ■

Our syntactic restriction is more complicated than the restrictions that are usually imposed on LFP and PBES. This is due to the fact that for PBES, it is usually required that *all* variables occur only in the scope of an even number of negations. This restriction is obviously sufficient, but more stringent than the restriction imposed on LFP, in which a bound variable is only required not to occur in the scope of an odd number of negations in its own right-hand side. Because we wish to be able to treat any valid LFP formula as an EFL formula, we have chosen to adapt this less strict requirement to the setting of EFL, resulting in a slightly more complicated notion of syntactic monotonicity due to our use of equation systems.

We wish to show that the predicate transformers induced by a system $\mathcal{E}$ are indeed monotone when $\mathcal{E}$ is syntactically monotone. Because the calculation of one predicate transformer may involve calculating the solution of another equation system, and therefore calculating the fixpoints of a number of other predicate transformers, we adopt a proof strategy in which we characterize how the solution $\mathbf{S}(\theta, \mathcal{E})$ of an equation system relates to the environment θ that it was given as input. This is done in terms of a relation under_X, defined as follows.

Definition 6.3. *Given a syntactically monotone system $\mathcal{E}$ and environments θ_1 and θ_2, we say that θ_1 is* under *θ_2 with respect to X, denoted $\theta_1\ \mathsf{under}_X\ \theta_2$, if and only if*

$$\theta_1 \equiv_{\mathcal{V}} \theta_2 \wedge \forall_{Y \in \mathsf{bnd}(\mathcal{E})}\ (\mathsf{pos}_{\mathcal{E},X}(\varphi_Y) \Rightarrow \theta_1(Y) \subseteq \theta_2(Y)) \wedge \\ (\mathsf{neg}_{\mathcal{E},X}(\varphi_Y) \Rightarrow \theta_1(Y) \supseteq \theta_2(Y)).$$

We can now show that this relationship is preserved by $\mathbf{S}$. This is stated in the following lemma.

Lemma 6.12. *For all syntactically and semantically monotone $\mathcal{E}$, $X \in \mathcal{X}$, θ_1 and θ_2,*

$$\theta_1\ \mathsf{under}_X\ \theta_2 \Rightarrow \mathbf{S}(\theta_1, \mathcal{E})\ \mathsf{under}_X\ \mathbf{S}(\theta_2, \mathcal{E}).$$

The proof of this lemma is postponed, but will involve two inductions: one on the structure of $\mathcal{E}$, and one on the structure of right-hand sides within $\mathcal{E}$. The proof used in that second induction will be re-used later, and is therefore stated in the next lemma. It sets some sufficient conditions under which monotonicity of a predicate transformer can be proven.

Lemma 6.13. *Let $\mathcal{E}$, φ, X, θ_1, θ_2 and $\bar{y}$ and $\bar{a}$ of corresponding size be such that:*

- *$X \in \mathsf{bnd}(\mathcal{E})$*
- *$\varphi = \varphi_Y$ for some $Y \in \mathsf{bnd}(\mathcal{E})$*

– $\mathbf{S}(\eta_1, \mathcal{E}')$ under_X $\mathbf{S}(\eta_2, \mathcal{E}')$ *for all* $\mathcal{E}' \sqsubseteq \mathcal{E}$ *and all* η_1, η_2 *such that* η_1 under_X η_2
– θ_1 under_X θ_2

Then the following implications hold:

$$\mathsf{pos}_{\mathcal{E},X}(\varphi) \Rightarrow (\mathfrak{A}, \mathbf{S}(\theta_1, \mathcal{E})[\bar{y} \mapsto \bar{a}] \models \varphi \Rightarrow \mathfrak{A}, \mathbf{S}(\theta_2, \mathcal{E})[\bar{y} \mapsto \bar{a}] \models \varphi), \text{ and}$$
$$\mathsf{neg}_{\mathcal{E},X}(\varphi) \Rightarrow (\mathfrak{A}, \mathbf{S}(\theta_2, \mathcal{E})[\bar{y} \mapsto \bar{a}] \models \varphi \Rightarrow \mathfrak{A}, \mathbf{S}(\theta_1, \mathcal{E})[\bar{y} \mapsto \bar{a}] \models \varphi).$$

Proof. We give the proof for the first implication; the proof for the second implication is similar. The proof is by structural induction on φ. Assume as the induction hypothesis that the lemma holds for all formulas smaller than φ, and assume $\mathsf{pos}_{\mathcal{E},X}(\varphi)$. Now distinguish cases based on the shape of φ.

Case $\varphi = S\bar{t}$, with S a first-order relation symbol. Then the semantics of φ depends only on the interpretation of S in $\mathfrak{A}$ and on the values assigned to first-order variables in $\mathbf{S}(\theta_1, \mathcal{E})[\bar{y} \mapsto \bar{a}]$ and $\mathbf{S}(\theta_2, \mathcal{E})[\bar{y} \mapsto \bar{a}]$, which are the same in both environments by the assumption that θ_1 under_X θ_2 (so $\theta_1 \equiv_{\mathcal{V}} \theta_2$) and Lemma 6.1.

Case $\varphi = [Z\bar{t} : \mathcal{E}']$, then we need to show that

$$\bar{t}^{\mathfrak{A}, \mathbf{S}(\theta_1, \mathcal{E})[\bar{y} \mapsto \bar{a}]} \in \mathbf{S}(\mathbf{S}(\theta_1, \mathcal{E})[\bar{y} \mapsto \bar{a}], \mathcal{E}')(Z)$$
$$\Rightarrow \bar{t}^{\mathfrak{A}, \mathbf{S}(\theta_2, \mathcal{E})[\bar{y} \mapsto \bar{a}]} \in \mathbf{S}(\mathbf{S}(\theta_2, \mathcal{E})[\bar{y} \mapsto \bar{a}], \mathcal{E}')(Z)$$

By assumption, $\mathbf{S}(\theta_1, \mathcal{E})$ under_X $\mathbf{S}(\theta_2, \mathcal{E})$, and therefore, by the same assumption (because $\mathcal{E}' \sqsubseteq \mathcal{E}$), also

$$\mathbf{S}(\mathbf{S}(\theta_1, \mathcal{E})[\bar{y} \mapsto \bar{a}], \mathcal{E}') \text{ } \mathsf{under}_X \text{ } \mathbf{S}(\mathbf{S}(\theta_2, \mathcal{E})[\bar{y} \mapsto \bar{a}], \mathcal{E}').$$

Because $\mathsf{pos}_{\mathcal{E},X}(\varphi)$, also $\mathsf{pos}_{\mathcal{E},X}(\varphi_Z)$ by Definition 6.2, and then by the definition of under_X,

$$\mathbf{S}(\mathbf{S}(\theta_1, \mathcal{E})[\bar{y} \mapsto \bar{a}], \mathcal{E}')(Z) \subseteq \mathbf{S}(\mathbf{S}(\theta_2, \mathcal{E})[\bar{y} \mapsto \bar{a}], \mathcal{E}')(Z),$$

which implies what we needed to prove.

Case $\varphi = \neg\psi$. Because $\mathsf{pos}_{\mathcal{E},X}(\varphi)$, $\mathsf{neg}_{\mathcal{E},X}(\psi)$. By the semantics of EFL,

$$\mathfrak{A}, \mathbf{S}(\theta_1, \mathcal{E})[\bar{y} \mapsto \bar{a}] \models \varphi \Leftrightarrow \mathfrak{A}, \mathbf{S}(\theta_1, \mathcal{E})[\bar{y} \mapsto \bar{a}] \not\models \psi.$$

Using the induction hypothesis, we obtain (using contraposition):

$$\mathfrak{A}, \mathbf{S}(\theta_1, \mathcal{E})[\bar{y} \mapsto \bar{a}] \not\models \psi \Rightarrow \mathfrak{A}, \mathbf{S}(\theta_2, \mathcal{E})[\bar{y} \mapsto \bar{a}] \not\models \psi.$$

Again using the semantics of EFL, we can now obtain our goal, because:

$$\mathfrak{A}, \mathbf{S}(\theta_2, \mathcal{E})[\bar{y} \mapsto \bar{a}] \not\models \psi \Leftrightarrow \mathfrak{A}, \mathbf{S}(\theta_2, \mathcal{E})[\bar{y} \mapsto \bar{a}] \models \varphi.$$

The remaining two cases ($\varphi = \psi_1 \vee \psi_2$ and $\varphi = \exists_x \psi$) are straightforward with a single appeal to the induction hypothesis, and are left as an exercise for the reader. □

We now return to the proof of the previously stated lemma.

Lemma 6.12. *For all syntactically and semantically monotone $\mathcal{E}$, $X \in \mathcal{X}$, θ_1 and θ_2,*

$$\theta_1 \operatorname{under}_X \theta_2 \Rightarrow \mathbf{S}(\theta_1, \mathcal{E}) \operatorname{under}_X \mathbf{S}(\theta_2, \mathcal{E}).$$

Proof. The proof is by structural induction on $\mathcal{E}$. For $\mathcal{E} = \varepsilon$, the lemma holds vacuously. Now assume as the induction hypothesis that the lemma holds for all systems smaller than $\mathcal{E}$, and let $\mathcal{E} = (\sigma\, Y\, \bar{y} = \varphi_Y)\mathcal{E}^1$. Assume $\theta_1 \operatorname{under}_X \theta_2$.

Let $\mathbf{T}_1 = \mathbf{T}_{\mathcal{E}}^{\mathfrak{A},\theta_1}$ and $\mathbf{T}_2 = \mathbf{T}_{\mathcal{E}}^{\mathfrak{A},\theta_2}$. Expanding the definition of $\mathbf{S}$, we find:

$$\mathbf{S}(\theta_1, \mathcal{E}) = \mathbf{S}(\theta_1[Y \mapsto \sigma\, \mathbf{T}_1], \mathcal{E}^1) \text{ and } \mathbf{S}(\theta_2, \mathcal{E}) = \mathbf{S}(\theta_2[Y \mapsto \sigma\, \mathbf{T}_2], \mathcal{E}^1).$$

If we can show that $\mathsf{pos}_{\mathcal{E},X}(\varphi_Z) \Rightarrow \sigma\, \mathbf{T}_1 \subseteq \sigma\, \mathbf{T}_2$ and $\mathsf{neg}_{\mathcal{E},X}(\varphi_Y) \Rightarrow \sigma\, \mathbf{T}_1 \supseteq \sigma\, \mathbf{T}_2$, then $\theta_1[Y \mapsto \sigma\, \mathbf{T}_1] \operatorname{under}_X \theta_2[Y \mapsto \sigma\, \mathbf{T}_2]$, and then by the induction hypothesis and the equalities above, $\mathbf{S}(\theta_1, \mathcal{E}) \operatorname{under}_X \mathbf{S}(\theta_2, \mathcal{E})$.

For the first implication, assume $\mathsf{pos}_{\mathcal{E},X}(\varphi_Y)$. We may prove that $\sigma\, \mathbf{T}_1 \subseteq \sigma\, \mathbf{T}_2$ by showing that $\mathbf{T}_1(R) \subseteq \mathbf{T}_2(R)$ for all $R \subseteq A^{\mathsf{ar}(Y)}$. Let $R \subseteq A^{\mathsf{ar}(Y)}$ and let $\bar{a} \in \mathbf{T}_1(R)$. We need to show that $\bar{a} \in \mathbf{T}_2(R)$. Let $\theta_1' = \theta_1[Y \mapsto R]$ and $\theta_2' = \theta_2[Y \mapsto R]$. Using the definitions of $\mathbf{T}_1$ and $\mathbf{T}_2$, and using the same argument for the second implication, our proof obligation becomes, for all $R \subseteq A^{\mathsf{ar}}(Y)$:

$$\mathsf{pos}_{\mathcal{E},X}(\varphi_Y) \Rightarrow (\mathfrak{A}, \mathbf{S}(\theta_1', \mathcal{E}^1)[\bar{y} \mapsto \bar{a}] \models \varphi_Y \Rightarrow \mathfrak{A}, \mathbf{S}(\theta_2', \mathcal{E}^1)[\bar{y} \mapsto \bar{a}] \models \varphi_Y), \text{ and}$$
$$\mathsf{neg}_{\mathcal{E},X}(\varphi_Y) \Rightarrow (\mathfrak{A}, \mathbf{S}(\theta_2', \mathcal{E}^1)[\bar{y} \mapsto \bar{a}] \models \varphi_Y \Rightarrow \mathfrak{A}, \mathbf{S}(\theta_1', \mathcal{E}^1)[\bar{y} \mapsto \bar{a}] \models \varphi_Y).$$

The induction hypothesis lets us apply Lemma 6.13 to directly obtain this result. □

The main result of this section is that syntactic monotonicity indeed implies semantic monotonicity, giving us the guarantee that the solution of an equation system is properly defined if that equation system is syntactically monotone. With the help of Lemmas 6.13 and 6.12, what remains is a straightforward induction on the structure of the equation system.

Theorem 6.1. *If $\mathcal{E}$ is syntactically monotone, then $\mathcal{E}$ is semantically monotone.*

Proof. The proof is by well-founded induction on $\mathcal{E}$ and $\sqsubseteq$. If $\mathcal{E} = \varepsilon$, then $\mathbf{S}(\theta, \mathcal{E})$ is defined to be equal to θ. As the induction hypothesis, assume that $\mathbf{S}(\theta, \mathcal{E}')$ is defined for all $\mathcal{E}' \sqsubset \mathcal{E}$.

If $\mathcal{E} = (\sigma\, X\, \bar{x} := \varphi)\mathcal{E}^1$, then we need to show that $\mathbf{S}(\theta, \mathcal{E}) = \mathbf{S}(\theta[X \mapsto \sigma\, \mathbf{T}_{\mathcal{E}}^{\mathfrak{A},\theta}], \mathcal{E}^1)$ is defined. We therefore need to show that $\sigma\, \mathbf{T}_{\mathcal{E}}^{\mathfrak{A},\theta}$ exists, which we do by proving that $\mathbf{T}_{\mathcal{E}}^{\mathfrak{A},\theta}$ is monotone. Once we have the proof for monotonicity of this predicate transformer, the induction hypothesis gives us the desired result.

To prove monotonicity of the predicate transformer, we need to show that for all $R, R' \subseteq A^{\mathsf{ar}(X)}$ such that $R \subseteq R'$, and for all $\bar{a} \in A^{\mathsf{ar}(X)}$,

$$\mathfrak{A}, \mathbf{S}(\theta[X \mapsto R], \mathcal{E}^1)[\bar{x} \mapsto \bar{a}] \models \varphi \Rightarrow \mathfrak{A}, \mathbf{S}(\theta[X \mapsto R'], \mathcal{E}^1)[\bar{x} \mapsto \bar{a}] \models \varphi. \qquad (*)$$

Note that all $\mathcal{E}' \sqsubseteq \mathcal{E}^1$ are syntactically monotone, and by the induction hypothesis therefore also semantically monotone. By Lemma 6.12 we also have $\eta_1 \operatorname{under}_X \eta_2 \Rightarrow \mathbf{S}(\eta_1, \mathcal{E}') \operatorname{under}_X \mathbf{S}(\eta_2, \mathcal{E}')$ for such $\mathcal{E}'$ and for all η_1 and η_2. This is sufficient to derive $(*)$ by applying Lemma 6.13. □

6.4 The PBES and LFP sublogics

We defined EFL in such a way that first-order logic can be seen as a fragment of EFL. We now investigate two fixpoint logics, PBES and LFP, and show that they too can be seen as syntactic fragments of EFL, both equally expressive as EFL itself. Of course the PBES and LFP logics traditionally use notation that differs from the EFL notation. We do not discuss the syntactic differences in detail, but Table 6.1 gives an example of the same statement, expressed in all three formalisms, using the traditional notation for each formalism.

LFP:	$[\mathbf{gfp}\, X\, x.[\mathbf{lfp}\, Y\, y.Y(y+1) \vee (X(y+1) \wedge \forall_{1<z<y}\ y \bmod z \neq 0)]x]1$
EFL:	$[X\,1 : \mathbf{gfp}\, X\, x = [Y\, x : \mathbf{lfp}\, Y\, y = Y(y+1) \vee (X(y+1) \wedge \forall_{1<z<y}\ y \bmod z \neq 0)]]$
PBES:	$\nu X(x:\mathbb{N}) = Y(x)$
	$\mu Y(y:\mathbb{N}) = Y(y+1) \vee (X(y+1) \wedge \forall_{1<z<y}\ y \bmod z \neq 0)$
	$\mathbf{init}\ X(1)$
EFL:	$[X\,1 : (\mathbf{gfp}\, X\, x = Y\, x)(\mathbf{lfp}\, Y\, y = Y(y+1) \vee (X(y+1) \wedge \forall_{1<z<y}\ y \bmod z \neq 0))]$

Table 6.1: The statement 'there are infinitely many prime numbers' in EFL, LFP and PBES notation.

Parameterised Boolean equation systems (PBES) The propositional variant of this formalism, called Boolean equation systems (BES), underlies model checking tool set CADP [Gar+11]. The first-order extension of BES results in an equational fixpoint logic which is the basis for model checking in the mCRL2 tool set [Cra+13]. It is used to encode the L_μ model checking problem and various equivalence checking problems on labelled transition systems.

Definition 6.1 corresponds exactly to the semantics of parameterised Boolean equation systems presented in [GW04], in the sense that for an equation system $\mathcal{E}$ with only first-order logic formulas as right-hand sides, $\mathbf{S}(\theta, \mathcal{E})$ corresponds to the definition of solution of a PBES. We can therefore characterise the PBES formalism as a fragment of EFL.

Definition 6.4. *An EFL formula φ is called a* parameterised Boolean equation system (PBES) *if and only if $\varphi = [X\,\bar{t} : \mathcal{E}]$ for some X, $\mathcal{E}$ and $\bar{t}$, such that all subformulas $[Y\,\bar{u} : \mathcal{E}']$ of right-hand sides of $\mathcal{E}$ have $\mathcal{E}' = \varepsilon$.*

The following lemma shows that we can always transform an equation system into an equation system with only first-order right-hand sides, in such a way that the solution of the original system corresponds with the solution of the transformed system. We first define a function that removes nested fixpoint operators from a formula. This function will be used again in Chapter 7.

Definition 6.5. *If φ is an EFL formula, then* $\mathsf{fo}(\varphi)$ *is equal to φ, in which every $[X\,\bar{t} : \mathcal{E}]$ is replaced by $[X\,\bar{t} : \varepsilon]$.*

Using this function, we can define a transformation that converts an EFL formula to a PBES formula.

Lemma 6.14. *Let $\mathcal{E}$ be an equation system with $\mathsf{bnd}^*(\mathcal{E}) = \{X_0, \ldots, X_n\}$, and let*

$$\mathcal{E}' = (\sigma_{X_0} X_0 \bar{x}_{X_0} = \mathsf{fo}(\varphi_{X_0})) \ldots (\sigma_{X_n} X_n \bar{x}_{X_n} = \mathsf{fo}(\varphi_{X_n})),$$

such that $X_i <_{\mathcal{E}} X_j$ for all $0 \leq i < j \leq n$. Then $\mathbf{S}^{\mathfrak{A}}(\theta, \mathcal{E})(X) = \mathbf{S}^{\mathfrak{A}}(\theta, \mathcal{E}')(X)$ for all $\mathfrak{A}, \theta$ and $X \in \mathsf{bnd}(\mathcal{E})$.

We postpone the proof of this lemma to Appendix A, because it uses some of the concepts that are presented in Chapter 7. It now follows that any EFL formula can be translated to an equivalent PBES formula.

Theorem 6.2. *Every EFL formula has an equivalent PBES formula.*

Proof. Let φ be an EFL formula, and let $X \in \mathcal{X}$ be a variable that does not occur in φ. Then φ is equivalent to $[X : (\mathbf{lfp} X = \varphi)]$. By Lemma 6.14, there then is a system $\mathcal{E}$ such that $[X : \mathcal{E}]$ is a PBES formula, and is equivalent to $[X : (\mathbf{lfp} X = \varphi)]$. □

Least fixpoint logic (LFP) Least fixpoint logic (LFP, see, *e.g.*, [Lib04] for an in-depth treatment) is a well known formalism that is usually defined as first-order logic, extended with a fixpoint operator. EFL captures LFP as a syntactic subset.

Definition 6.6. *An EFL formula φ is a formula of LFP if and only if for every $[X\,\bar{t} : \mathcal{E}] \sqsubseteq \varphi$, $\mathcal{E}$ is empty or consists of a single equation that defines X.*

The definition of EFL semantics with its reference to Definition 6.1 then simplifies to the following: $\mathfrak{A}, \theta \models [X\,\bar{t} : \varepsilon]$ if and only if $\bar{t}^{\mathfrak{A},\theta} \in \theta(X)$, and for $\mathcal{E} \neq \varepsilon$, $\mathfrak{A}, \theta \models [X\,\bar{t} : \mathcal{E}]$ if and only if $\bar{t}^{\mathfrak{A},\theta} \in \sigma(T^{\mathfrak{A},\theta})$, where:

$$T^{\mathfrak{A},\eta}(R) = \{\bar{a} \mid \mathfrak{A}, \eta[X \mapsto R, \bar{x} \mapsto \bar{a}] \models \varphi\}$$

This corresponds exactly to the semantics of the fixpoint operator in LFP.

We will now show that any EFL formula can be transformed to an equivalent LFP formula, through the use of the two transformation rules laid out in the following two lemmas. The first transformation rule says that if in a formula $[X\,\bar{t} : \mathcal{E}]$, X is the first bound variable of $\mathcal{E}$, then $\mathcal{E}^1$ can be 'pushed inwards' into φ_X, leaving a new formula $[X\,\bar{t} : \mathcal{E}']$ in which $\mathcal{E}'$ only has one equation. This shows the correspondence between the nesting order of variables in LFP and the order of equations in PBES.

Lemma 6.15. *Given $\mathfrak{A}$, θ and an equation system $\mathcal{E} = (\sigma X \bar{x} = \varphi)\mathcal{E}'$,*

$$\mathbf{S}(\theta, \mathcal{E})(X) = \mathbf{S}(\theta, (\sigma X \bar{x} = \varphi'))(X),$$

where φ' is φ in which all $[Y\,\bar{t} : \mathcal{F}]$ are replaced by $[Y\,\bar{t} : \mathcal{E}'\mathcal{F}]$.

Proof. We prove the lemma by showing that $\mathbf{T}_{\mathcal{E}}^{\mathfrak{A},\theta} = \mathbf{T}_{(\sigma X \bar{x} = \varphi')}^{\mathfrak{A},\theta}$. We show for all R

$$\begin{aligned}\mathbf{T}_{\mathcal{E}}^{\mathfrak{A},\theta}(R) &= \{\bar{a} \mid \mathfrak{A}, \mathbf{S}(\theta[X \mapsto R], \mathcal{E}')[\bar{x} \mapsto \bar{a}] \models \varphi\} \\ &= \{\bar{a} \mid \mathfrak{A}, \mathbf{S}(\theta[X \mapsto R], \varepsilon)[\bar{x} \mapsto \bar{a}] \models \varphi'\} = \mathbf{T}_{(\sigma X \bar{x} = \varphi')}^{\mathfrak{A},\theta}(R),\end{aligned}$$

by showing that $\mathfrak{A}, \mathbf{S}(\theta[X \mapsto R], \mathcal{E}')[\bar{x} \mapsto \bar{a}] \models \varphi$ and $\mathfrak{A}, \mathbf{S}(\theta[X \mapsto R], \varepsilon)[\bar{x} \mapsto \bar{a}] \models \varphi'$ are equivalent. We do this by structural induction on φ. We give the proof for base case $\varphi = [Y\bar{t} : \mathcal{F}]$. Note that by our unique-namedness assumption, $\mathsf{fv}(\mathcal{E}') \cap \mathsf{bnd}(\mathcal{F}) = \emptyset$.

$$\begin{aligned}&\mathfrak{A}, \mathbf{S}(\theta[X \mapsto R], \mathcal{E}')[\bar{x} \mapsto \bar{a}] \models \varphi \\ \Leftrightarrow\ & \bar{t}^{\mathfrak{A}, \mathbf{S}(\theta[X \mapsto R], \mathcal{E}')[\bar{x} \mapsto \bar{a}]} \in \mathbf{S}(\mathbf{S}(\theta[X \mapsto R], \mathcal{E}')[\bar{x} \mapsto \bar{a}], \mathcal{F})(Y) \\ \Leftrightarrow\ & \bar{t}^{\mathfrak{A}, \mathbf{S}(\theta[X \mapsto R], \mathcal{E}')[\bar{x} \mapsto \bar{a}]} \in \mathbf{S}(\mathbf{S}(\theta[X \mapsto R], \mathcal{E}'), \mathcal{F})(Y) && \text{(Lemma 6.2)} \\ \Leftrightarrow\ & \bar{t}^{\mathfrak{A}, \mathbf{S}(\theta[X \mapsto R], \mathcal{E}')[\bar{x} \mapsto \bar{a}]} \in \mathbf{S}(\mathbf{S}(\mathbf{S}(\theta[X \mapsto R], \mathcal{E}'), \mathcal{E}'), \mathcal{F})(Y) && \text{(Lemma 6.4)} \\ \Leftrightarrow\ & \bar{t}^{\mathfrak{A}, \mathbf{S}(\theta[X \mapsto R], \mathcal{E}')[\bar{x} \mapsto \bar{a}]} \in \mathbf{S}(\mathbf{S}(\theta[X \mapsto R], \mathcal{E}')[\bar{x} \mapsto \bar{a}], \mathcal{E}'\mathcal{F})(Y) && \text{(Lemma 6.7)} \\ &\mathfrak{A}, \mathbf{S}(\theta[X \mapsto R], \mathcal{E}')[\bar{x} \mapsto \bar{a}] \models \varphi'\end{aligned}$$

The inductive cases are now straightforward. □

Note that in the previous lemma, the resulting formula does not necessarily satisfy the unique-name assumption anymore. This can be easily resolved by renaming the variables in the subsystems appropriately.

Of course, we may need to translate a formula $[X\bar{t} : \mathcal{E}]$ where X is not defined by the first equation in $\mathcal{E}$. If X is bound by the i^{th} equation in $\mathcal{E}$, we will copy $\mathcal{E}^i$, rename all bound variables in the copy, and prepend it to $\mathcal{E}$, thus creating a system in which a copy of X is defined by the first equation in the system. This will be formalized later; the following lemma first states that this transformation of the system is sound.

Lemma 6.16. *Given $\mathfrak{A}$, θ, a mapping $f : \mathsf{bnd}^*(\mathcal{E}_2) \to \mathcal{X} \setminus (\mathsf{bnd}^*(\mathcal{E}) \cup \mathsf{fv}(\mathcal{E}))$, and an equation system $\mathcal{E} = \mathcal{E}_1\mathcal{E}_2$,*

$$\mathbf{S}(\theta, \mathcal{E})(X) = \mathbf{S}(\theta, f(\mathcal{E}_2)\mathcal{E})(f(X)) \text{ for all } X \in \mathsf{bnd}(\mathcal{E}_2),$$

where f is extended to range over $\mathcal{X}$ and over equation systems in the obvious way.

The proof of this lemma again uses concepts that are presented in Chapter 7, and is therefore moved to Appendix A. Using the two previous lemmas, and the standard lemmas at the beginning of the chapter, we can now show that any EFL formula can be translated to LFP. The next lemma forms the main part of the proof.

Lemma 6.17. *For all $\mathcal{E}_1$, X, $\mathcal{E}_2$ and $\bar{t}$ such that $[X\bar{t} : \mathcal{E}_1\mathcal{E}_2]$ is a PBES formula and:*

- *$X \in \mathsf{bnd}(\mathcal{E}_1)$,*
- *$\mathsf{bnd}(\mathcal{E}_1) \cap \mathsf{fv}(\mathcal{E}_2) = \emptyset$,*
- *for all $Y \in \mathsf{fv}(\mathcal{E}_1) \cap \mathsf{bnd}(\mathcal{E}_2)$ and $\bar{u}$ of the appropriate size, $[Y\bar{u} : \mathcal{E}_2]$ has an equivalent LFP formula,*

$[X\bar{t} : \mathcal{E}_1\mathcal{E}_2]$ *has an equivalent LFP formula.*

Proof. We prove the lemma with two inductions, first on the size of $\mathcal{E}_1$. Fix $\mathcal{E}_1$, and assume as the *outer induction hypothesis* that the lemma holds for all $\mathcal{E}_1'$ with fewer equations. Now fix some $X \in \mathsf{bnd}(\mathcal{E}_1)$ and assume as the *inner induction hypothesis* that the lemma holds for $\mathcal{E}_1$ and all $X' <_{\mathcal{E}_1} X$. Finally, fix some $\bar{t}$ and some $\mathcal{E}_2$ that meets the requirements; our proof obligation is now to show that $[X\bar{t} : \mathcal{E}_1\mathcal{E}_2]$ has an equivalent LFP formula.

Let i be such that $\mathcal{E}_1^i = (\sigma X\bar{x} = \varphi)\mathcal{E}'$ for some σ, $\bar{x}$, φ and $\mathcal{E}'$. Define a bijection $f : \mathcal{X} \to \mathcal{X}$ that maps variables bound by $\mathcal{E}_1^i\mathcal{E}_2$ to variables not in $\mathsf{bnd}(\mathcal{E}_1\mathcal{E}_2) \cup \mathsf{fv}(\mathcal{E}_1\mathcal{E}_2)$, and that maps every variable bound by $\mathcal{E}_1\mathcal{E}_2$ but not by $\mathcal{E}_1^i\mathcal{E}_2$ to itself. For an equation system $\mathcal{F}$, let $f(\mathcal{F})$ denote the syntactic replacement of every $Y \in \mathcal{X}$ in $\mathcal{F}$ by $f(Y)$ (*i.e.*, it replaces variable names on both sides of the equations in $\mathcal{F}$).

By Lemma 6.16, $[X\bar{t} : \mathcal{E}_1\mathcal{E}_2]$ is equivalent to $[f(X)\bar{t} : f(\mathcal{E}_1^i\mathcal{E}_2)\mathcal{E}_1\mathcal{E}_2]$. By Lemma 6.15 then, it is equivalent to $[f(X)\bar{t} : (\sigma f(X)\bar{x} = \varphi')]$, where φ' is equal to $f(\varphi)$ in which every $[Y\bar{u} : \mathcal{F}]$ is replaced by $[Y\bar{u} : f(\mathcal{E}^i\mathcal{E}_2)^1\mathcal{E}_1\mathcal{E}_2\mathcal{F}]$.

Because we assumed that $[X\bar{t} : \mathcal{E}_1\mathcal{E}_2]$ is a PBES formula, $\mathcal{F} = \varepsilon$, making the proof a little easier. To meet our proof obligation, we show that we can replace every such $[Y\bar{u} : f(\mathcal{E}_1^i\mathcal{E}_2)^1\mathcal{E}_1\mathcal{E}_2]$ in φ' by an equivalent LFP formula, yielding a formula φ'' equivalent to φ'. Obviously, $[f(X)\bar{t} : (\sigma f(X)\bar{x} = \varphi'')]$ is then an LFP formula (we may need to rename some variables from $\mathcal{X}$ to meet the unique naming property).

Note that we may write $[Y\bar{u} : f(\mathcal{E}_1^i\mathcal{E}_2)^1\mathcal{E}_1\mathcal{E}_2]$ as $[Y\bar{u} : f(\mathcal{E}_1^{i+1})f(\mathcal{E}_2)\mathcal{E}_1\mathcal{E}_2]$. Our new proof obligation is to show that this formula has an equivalent LFP formula. We do this by distinguishing cases on where Y is bound.

Case $Y \notin \mathsf{bnd}(f(\mathcal{E}_1^{i+1})f(\mathcal{E}_2)\mathcal{E}_1\mathcal{E}_2)$, then $[Y\bar{u} : \varepsilon]$ is equivalent.

Case $Y \in \mathsf{bnd}(\mathcal{E}_1\mathcal{E}_2)$, note that Y cannot be bound in $\mathcal{E}_1^i\mathcal{E}_2$, because Y occurs in φ', in which all bound variables from $\mathcal{E}_1^i\mathcal{E}_2$ have been renamed by f. Therefore, $Y <_{\mathcal{E}_1} X$. Furthermore, $\mathsf{fv}(\mathcal{E}_1\mathcal{E}_2) \cap \mathsf{bnd}(f(\mathcal{E}_1^{i+1})f(\mathcal{E}_2)) = \emptyset$, so by Lemma 6.10, $[Y\bar{u} : f(\mathcal{E}_1^{i+1})f(\mathcal{E}_2)\mathcal{E}_1\mathcal{E}_2]$ is equivalent to $[Y\bar{u} : \mathcal{E}_1\mathcal{E}_2]$. This formula in turn has an equivalent LFP formula by the inner induction hypothesis.

Case $Y \in \mathsf{bnd}(f(\mathcal{E}_2))$, then because $\mathsf{fv}(\mathcal{E}_2) \cap \mathsf{bnd}(\mathcal{E}_1) = \emptyset$, and f preserves this property, $\mathsf{fv}(f(\mathcal{E}_2)) \cap \mathsf{bnd}(f(\mathcal{E}_1^{i+1})) = \emptyset$.

Furthermore, also $\mathsf{fv}(f(\mathcal{E}_2)) \cap \mathsf{bnd}(\mathcal{E}_1\mathcal{E}_2) = \emptyset$, because of the way f was defined, and because $\mathsf{fv}(\mathcal{E}_2) \cap \mathsf{bnd}(\mathcal{E}_1) = \emptyset$, so the variables not renamed by f can not be in $\mathsf{bnd}(\mathcal{E}_1)$.

We may therefore apply Lemmas 6.9 and 6.10 to see that $[Y\bar{u} : f(\mathcal{E}_1^{i+1})f(\mathcal{E}_2)\mathcal{E}_1\mathcal{E}_2]$ is equivalent to $[Y\bar{u} : f(\mathcal{E}_2)]$. Note that because $Y \in \mathsf{bnd}(f(\mathcal{E}_2))$, there must be some Y' such that $Y = f(Y')$ and $Y' \in \mathsf{bnd}(\mathcal{E}_2)$. Furthermore, note that $[Y\bar{u} : f(\mathcal{E}_2)]$ is equivalent to $[Y'\bar{u} : \mathcal{E}_2]$. By our assumptions about Y' and $\mathcal{E}_2$, $[Y'\bar{u} : \mathcal{E}_2]$ in turn has an equivalent LFP formula.

Case $Y \in \mathsf{bnd}(f(\mathcal{E}_1^{i+1}))$, observe that $f(\mathcal{E}_1^{i+1})$ has fewer equations than $\mathcal{E}_1$. We also know that $\mathsf{bnd}(f(\mathcal{E}_1^{i+1})) \cap \mathsf{fv}(f(\mathcal{E}_2)\mathcal{E}_1\mathcal{E}_2) = \emptyset$, because $\mathsf{bnd}(\mathcal{E}_1) \cap \mathsf{fv}(\mathcal{E}_2) = \emptyset$, and

because f creates fresh variables that did not occur in $\mathcal{E}_1\mathcal{E}_2$. The reasoning for the two previous cases also applies to prove that for every $Z \in \mathsf{fv}(f(\mathcal{E}_1^{i+1})) \cap \mathsf{bnd}(f(\mathcal{E}_2)\mathcal{E}_1\mathcal{E}_2)$ and all $\bar{v}$ of the appropriate size, $[Z\bar{v} : f(\mathcal{E}_2)\mathcal{E}_1\mathcal{E}_2]$ has an equivalent LFP formula. We can therefore derive from the outer induction hypothesis that also $[Y\bar{u} : f(\mathcal{E}_1^{i+1})f(\mathcal{E}_2)\mathcal{E}_1\mathcal{E}_2]$ has an equivalent LFP formula. □

Note that it would be possible to formulate Lemma 6.17 for EFL formulas rather than for PBES formulas. In that case, an extra induction is needed on the nesting depth of the formula (where in our proof we use that $\mathcal{F} = \varepsilon$, this becomes a base case of the induction). We do not need this, however, to prove the following theorem.

Theorem 6.3. *Any EFL formula has an equivalent LFP formula.*

Proof. Given an EFL formula, first convert it to an equivalent PBES formula $[X\bar{t} : \mathcal{E}]$ using Theorem 6.2. Then apply Lemma 6.17 to obtain an equivalent LFP formula, by choosing $\mathcal{E}_1 = \mathcal{E}$ and $\mathcal{E}_2 = \varepsilon$. □

Note that the constructive nature of the proof immediately suggests an algorithm to implement a translation procedure from EFL formulas to LFP formulas. We could have chosen an approach that focuses more on this algorithmic aspect, for instance along the lines of the translation from modal equation systems to modal μ-calculus in [BFL13], but this approach has the advantage that it yields two independent transformations on EFL formulas that preserve the solution, in the form of Lemmas 6.15 and 6.16.

Corollary 6.1. *The EFL, LFP and PBES formalisms are equivalent.*

6.5 Closing remarks

We have defined the logic EFL, which is a syntactic generalization of the LFP and PBES logics, and we have proven it to be equally expressive as LFP and PBES by showing that both these logics are normal forms of EFL, in the sense that a formula of EFL can always be translated to an equivalent formula of LFP and an equivalent formula of PBES.

Like PBES and LFP, we have defined a notion of syntactic monotonicity of EFL formulas, which guarantees that a formula is semantically monotone, and that therefore the semantics of the formula is defined. Incidentally, this notion of syntactic monotonicity is more permissive on the PBES fragment of EFL than the notion of syntactic monotonicity on PBES is, at the cost of being slightly more complicated. The original notion of syntactic monotonicity for PBES formulas requires that all second-order variables only occur in the scope of an even number of negations (usually achieved by requiring that formulas can be transformed to *positive normal form*, in which negation only occurs on first-order atomic subformulas). In our notion of syntactic monotonicity, negations are allowed everywhere, as long as the equations in the formula do not imply an equality between a second-order variable and its negation. On the LFP fragment of EFL, our notion of syntactic monotonicity coincides with that of LFP itself.

Chapter 7

Evidence

There are many techniques that can be used to solve a model checking problem, and all of those techniques will provide you—ultimately, given enough time and other resources—with the answer to the question you pose to it. But although model checking questions are *yes*/*no* questions, in many cases the enquirer is not so much interested in the answer itself, but in *understanding* the answer.

It is difficult to define what it means to understand the answer to a model checking problem, but it is easy to think, for specific situations, about what evidence might be shown to the enquirer to help him understand. For instance, if the enquirer wishes to prove for some system: *'I can press a sequence of buttons to make the light go on,'* then presenting him with such a sequence is more helpful than just telling him that this statement can indeed be proven. In the setting of automated verification, where systems are usually represented by a Kripke structure or labelled transition system, generation of such sequences can be automated for all questions about systems that allow the answer to be motivated by a trace through the system. Such questions can be posed in LTL, and most LTL model checkers provide functionality to automatically extract traces from the system under scrutiny to motivate the answer to the question.

Not all questions can be explained by just looking at a trace through the system. For example, *'if the light burns after pressing the left button, then the light would not have burned if I had pressed the right button'* is a statement that can only be shown to be true by examining two distinct traces in the system: one in which I press the left button, and one in which I press the right button. Apparently, for some questions about systems it is essential to provide evidence that takes into account the *branching behaviour* of the system. Clarke *et al.* showed that questions of a specific form allow for explanations that are in a way the branching equivalent of traces through the system [Cla+02].

For certain classes of questions, we have seen that there are notions of evidence that can be automatically generated by model checking tools. Unfortunately, these notions are tied to subsets of ECTL*, and the algorithms to extract evidence are all defined as extensions of model checking algorithms that only check these subsets. The mCRL2 toolset [Cra+13] has chosen FO-L_μ as its modal logic, because of its expressivity (in particular, it is more expressive than ECTL*). The toolkit encodes model checking

problems in a particular equational fixpoint logic called parameterized Boolean equation systems. The parameterized Boolean equation system (PBES) that encodes the problem is then either solved directly by a first-order variant of Gauß elimination, or it is translated to a parity game. In the latter case, the solution of the PBES can be derived from the solution of the parity game.

This choice of modal logic and solving method provides a lot of flexibility. The user can use the full power of FO-L_μ, and the PBES formalism is so generic that many results and tools for first-order logic, fixpoint logic and boolean equation systems can be reused. But the translation of the model checking problem to this generic setting comes at the cost of not having much feedback from the model checker, other than the plain yes/no answer to the question it was posed. This is especially problematic and time consuming if the question that was posed to the model checker is not the question that was *intended* to be posed to the model checker.

Ultimately, the mCRL2 tools could be used a lot more efficiently if the verification via encoding to PBES would allow for the generation of evidence. The wish to extract evidence, however, poses a few tough questions:

– Is there a sensible notion of evidence for the FO-L_μ model checking problem?
– Can existing notions of evidence (traces, tree-like evidence) be obtained when solving model checking problems via encoding to PBES?

The first question is rather subjective, and it is not difficult to think of model checking problems for which no real interesting evidence can be given. For instance, the only evidence that you could logically give to disprove the statement *'I can reach a state in which the light is on'* is to show the user all the behaviour that the system can display. While this would be a logical notion of evidence, it may not be very helpful, as it is the model checker's way of saying 'I just checked every state in the system, but if you don't believe me, you can just check all those states yourself,' which defeats the point of performing model checking automatically.

We therefore focus on the second question. In this chapter, we will present a notion of evidence for fixpoint logics that allows us to answer that question positively, and which induces a notion of evidence for FO-L_μ. We leave it up to the reader to decide whether that notion is sensible.

The fact that evidence extraction is usually dependent on the model checking algorithm that is used, was previously acknowledged by Tan et al. [Tan02; TC02]. In the setting of L_μ model checking, they investigated how evidence could be retrieved if the model checking problem was first encoded into Boolean equation systems (the propositional variant of PBES). This led them to define a notion of *support sets*, which can be understood to be proof objects for Boolean equation systems, *i.e.*, structures that explain the solution to a Boolean equation system. They showed that existing, specialised model checking algorithms are easily adapted to create support sets during verification. Moreover, support sets can be generated for any Boolean equation system. They claim that if a Boolean equation system encodes a model checking problem in a certain way, system traces may be extracted that explain the answer of the encoded problem. A similar approach, albeit slightly less generic, is followed by Mateescu in [Mat00].

In [CLW13] we showed a counterexample to Tan's claim that counterexamples could be retrieved from support sets, but the idea of using proof objects for equation systems that contain enough information to extract diagnostics is a sound one. Our approach therefore was to adapt Tan's notion of support set to the setting of PBESs by extending it with concepts from first-order logic. This led to the definition of the notion of *proof graph* in that same paper. This definition turned out to be a good proof object for PBESs in the sense that these proof graphs formed an alternative characterisation of the solution of PBESs, but it did not fulfil our requirement that it should contain all information necessary to extract evidence for the encoded model checking problem.

This chapter describes our second attempt at defining a proof graph for PBESs that enables us to extract model checking diagnostics. In order to make the results more widely applicable, we give our definitions in the setting of EFL. As we have seen, PBES, LFP and EFL are equivalent, and we hope that providing our definitions and proofs in the setting of EFL will make it easier to share results between communities that prefer different fixpoint logics.

The chapter is organised as follows. We start by presenting the notion of proof graph for EFL in Section 7.1, and show that it characterises the solution of EFL formulas. Section 7.2 then presents a notion of evidence for EFL, and shows how this notion can be specialised for EFL formulas that encode certain types of problems. Finally, in Section 7.3, we show how the existing notions of counterexample and witness proposed by Clarke et al. can be retrieved from proof graphs. The chapter is concluded with some closing remarks in Section 7.4.

7.1 Proof graphs

In this section, we will be considering *proof graphs*, a type of graph that represents a proof for an EFL formula of the form $[X\,\bar{t} : \mathcal{E}]$. Before we look at the formal definition, let us try to understand on a more intuitive level what we are trying to achieve.

A proof graph for a structure $\mathfrak{A}$, an environment θ and an equation system $\mathcal{E}$ is a collection of nodes that represent truths or falsehoods in $\mathfrak{A}$ and θ, connected by directed edges that indicate dependencies between them. Every such truth or falsehood is a statement of the form 'the tuple $\bar{a}$ is / is not an element of the relation represented by X', where $\bar{a} \in A^*$ is a tuple over the domain of discourse, and X a syntactic element representing some semantic relation over A. The nodes of the graph will be represented by tuples from the set

$$\mathcal{S} = \{\langle \alpha, X, \bar{a}\rangle \in \mathbb{B} \times (\mathcal{R} \cup \mathcal{X}) \times A^* \mid \mathsf{ar}(X) = |\bar{a}|\}.$$

We will often be considering subsets of $\mathcal{S}$ in which only relation symbols and second-order variables from some set $\mathcal{Y}$ occur. For such occasions, we introduce the following shorthand:

$$\mathcal{S}_{\mathcal{Y}} = \{\langle \alpha, X, \bar{a}\rangle \in \mathbb{B} \times \mathcal{Y} \times A^* \mid \mathsf{ar}(X) = |\bar{a}|\}.$$

The Boolean (usually named α, β, etc.[1]) indicates whether the node represents a true statement or a false statement. X is either a relation symbol or a second-order variable: in any case, a symbol that represents a relation over A. If α is true, then the tuple $\bar{a}$ represents the assertion that $\bar{a}$ is an element of the relation represented by X; if α is false, then it represents the assertion that $\bar{a}$ is not an element of that relation. We explicitly introduce notation that denotes that the statement represented by a node holds in a model $\mathfrak{A}$ and environment θ:

$$\mathfrak{A}, \theta \models \langle \alpha, X, \bar{a} \rangle \quad \text{denotes} \quad \bar{a} \in X^{\mathfrak{A},\theta} \iff \alpha = \mathbb{t}.$$

The edge relation of a proof graph represents dependencies between fixpoint variables. Therefore, nodes that represent truths and falsehoods that can be immediately seen from $\mathfrak{A}$ or θ (this is the case when the second element of the node is a relation symbol or second-order variable unbound in $\mathcal{E}$) have no successors. Nodes $\langle \alpha, X, \bar{a} \rangle$ that say something about the validity of a bound variable X of $\mathcal{E}$, must have successors that together give enough information to conclude that the right-hand side of X in $\mathcal{E}$ is indeed true if $\alpha = \mathbb{t}$, or false if $\alpha = \mathbb{f}$.

The intuition here is that we interpret each successor as a statement about the solution of $\mathcal{E}$: if $\langle \mathbb{t}, Y, \bar{b} \rangle$ is a successor of some node $\langle \alpha, X, \bar{a} \rangle$ and Y is bound in some subsystem $\mathcal{E}'$, then we interpret this successor to mean that $\bar{b} \in \mathbf{S}(\mathbf{S}(\theta, \mathcal{E}), \mathcal{E}')(Y)$. This is equivalent to $\mathfrak{A}, \mathbf{S}(\theta, \mathcal{E}) \models [Y\,\bar{t} : \mathcal{E}']$ if $\bar{t}^{\mathfrak{A},\mathbf{S}(\theta,\mathcal{E})} = \bar{b}$, so the information that this successor conveys is enough to conclude that certain $[Y\,\bar{t} : \mathcal{E}']$ hold when evaluating $\mathcal{E}$ in the environment $\mathbf{S}(\theta, \mathcal{E})$. From Lemma 6.6, we know that $\bar{a}$ being an element of $\mathbf{S}(\theta, \mathcal{E})(X)$ (*i.e.*, the value of α) depends on evaluating the right-hand side of X in $\mathbf{S}(\theta, \mathcal{E})$. Intuitively, including such a successor for every fixpoint subformula $[Y\,\bar{t} : \mathcal{E}']$ that occurs in the right-hand side of X in $\mathcal{E}$ should therefore be sufficient to explain why that right-hand side does or does not hold.

Let us start translating this intuition into a more formal statement. Remember the function $\mathsf{fo}(\varphi)$ (Definition 6.5) that replaces all $[X\,\bar{t} : \mathcal{E}] \sqsubseteq \varphi$ by $X\,\bar{t}$. From the semantics of EFL we can see that if $\eta(Y) = \mathbf{S}(\mathbf{S}(\theta, \mathcal{E}), \mathcal{E}')(Y)$ for all $[Y\,\bar{t} : \mathcal{E}']$ occurring as direct subformulas of φ, then $\mathfrak{A}, \eta \models \mathsf{fo}(\varphi)$ is equivalent to $\mathfrak{A}, \mathbf{S}(\theta, \mathcal{E}) \models \varphi$ (a generalised version of this idea will be formalised in Lemma 7.5).

Later in this chapter, we will try to identify those portions of a model that are necessary to explain the validity or invalidity of φ. Given a proof graph with nodes S, we will be looking at substructures of $\mathfrak{A}$ that have a domain of discourse that includes all the elements that are referenced by the nodes in S. The smallest such model, given a set of nodes S referencing elements of $\mathfrak{A} = \langle \Sigma, A, \mathcal{I} \rangle$, is defined as

$$\mathfrak{A} \restriction S = \langle \Sigma, A', \mathcal{I}^0 \rangle,$$

where A' is the smallest superset of $\{a \in A \mid \exists_{\langle \alpha, R, \bar{c} \rangle \in S}\ a \in \bar{c}\}$ that is closed under $\mathcal{I}^0$, and $\mathcal{I}^0$ is defined as $\mathcal{I}$, except that $\mathcal{I}^0(R) = \emptyset$ for all first-order relation symbols R.

The following definition formalises what it means for a node to be 'sufficiently explained' by its successors; in essence, it checks for a node $\langle \alpha, X, \bar{a} \rangle$ that for any η

[1] The letter α is a reasonable choice not only because we are rapidly running out of Roman letters, but also because it is the initial of the Greek word for 'truth': αληθεια

and a $\mathfrak{B}$ that include the information that is encoded by the successors of that node, $\mathfrak{B}, \eta \models \mathsf{fo}(\varphi_X)$ (or $\mathfrak{B}, \eta \not\models \mathsf{fo}(\varphi_X)$ if $\alpha = \mathbb{f}$). To avoid having to distinguish between the different values for α every time, we introduce the following shorthand notation:

$$\mathfrak{A}, \theta \,{}^{\alpha}\!\models \varphi \quad \text{denotes} \quad \mathfrak{A}, \theta \models \varphi \Leftrightarrow \alpha = \mathbb{t}.$$

The set of successors (the *postset*) of a node v is denoted $v^\bullet$. We enforce consistency of the reasoning represented by the graph by requiring that the successors of a node are never contradictory: a relation $\rightarrow \subseteq \mathcal{S} \times \mathcal{S}$ is *consistent* if and only if for all v, α, X and $\bar{a}$, not both $\langle \alpha, X, \bar{a} \rangle \in v^\bullet$ and $\langle \neg\alpha, X, \bar{a} \rangle \in v^\bullet$.

Definition 7.1. *A* dependency graph *for $\mathfrak{A}, \theta$ and $\mathcal{E}$ is a directed graph $\langle S, \rightarrow \rangle$ with $S \subseteq \mathcal{S}$ and $\rightarrow \subseteq S \times S$, such that $\rightarrow$ is consistent, and for all $\langle \alpha, X, \bar{a} \rangle \in S$:*

– *if $X \in \mathsf{bnd}^*(\mathcal{E})$:*

$$\begin{array}{l} \textit{for all } \mathfrak{A} \restriction S \sqsubseteq \mathfrak{B} \sqsubseteq \mathfrak{A} \textit{ and all } \eta \textit{ such that } \eta \equiv_{\mathsf{fv}(\mathcal{E})} \theta, \\ \quad \textit{if } \mathfrak{B}, \eta \models \nu \textit{ for all } \nu \in \langle \alpha, X, \bar{a} \rangle^\bullet \textit{ then } \mathfrak{B}, \eta[\bar{x}_X \mapsto \bar{a}] \,{}^{\alpha}\!\models \mathsf{fo}(\varphi_X) \end{array}$$

– *if $X \notin \mathsf{bnd}^*(\mathcal{E})$:*

$$\mathfrak{A}, \theta \models \langle \alpha, X, \bar{a} \rangle \qquad \textit{and} \qquad \langle \alpha, X, \bar{a} \rangle^\bullet = \emptyset$$

The universal quantification over η and $\mathfrak{B}$ ensures that the right information is included in the proof graph; in a sense the quantification discards all information from θ and $\mathfrak{A}$ that we do not consider to be valid information to base a proof on. The quantification over η, for instance, allows a proof to use information about the value of the free variables of $\mathcal{E}$ in θ via η, but if information about other second-order variables is needed to prove $\mathfrak{B}, \eta[\bar{x}_X \mapsto \bar{a}] \,{}^{\alpha}\!\models \mathsf{fo}(\varphi_X)$, then this dependency must be represented by an edge in the proof graph. Similarly, the quantification over $\mathfrak{B}$ creates the obligation for a dependency graph to include a node for every fact about first-order relations that is needed to prove $\mathfrak{B}, \eta[\bar{x}_X \mapsto \bar{a}] \,{}^{\alpha}\!\models \mathsf{fo}(\varphi_X)$ according to the semantics of first-order logic.

Although a dependency graph captures 'local' reasoning about right-hand sides of bound variables, it does not take into account the nature of the fixpoints associated with those variables. Roughly speaking, a least fixpoint should be explained by an inductive argument, and a greatest fixpoint by a coinductive argument. Such reasoning should in turn correspond to a structure in a proof graph. An inductive argument always combines some basic facts to derive new truths using some finite reasoning. The proof of an inductively proven fact (*i.e.*, the proof for a tuple $\bar{a}$ being an element of a relation that is a least fixpoint) always takes the form of a finite, acyclic graph. For coinductive reasoning, this restriction is not present; a coinductive proof essentially explains that there is no inductive proof of any size for the statement *not* holding—a strategy that may require infinite or cyclic reasoning to simulate quantifying over all inductive proofs. The requirements for a valid reasoning about least and greatest fixpoints are stated formally in the following definition, in which $I_X(\pi)$ denotes the set of indices on path π such that π_i is an X-node for all $i \in I_X(\pi)$ (by X-node we mean a node $\langle \alpha, Y, \bar{a} \rangle$ with $Y = X$).

Definition 7.2. *A dependency graph $G = \langle S, \to \rangle$ for $\mathfrak{A}, \theta$ and $\mathcal{E}$ is a* proof graph *iff for every infinite path π in G, the smallest X w.r.t. $<_{\mathcal{E}}$ for which $I_X(\pi)$ is infinite satisfies:*

- *if $\sigma_X = \mathbf{gfp}$, then $\{i \in I_X(\pi) \mid \exists_{\bar{a}}\ \pi_i = \langle \mathbb{f}, X, \bar{a} \rangle\}$ is finite.*
- *if $\sigma_X = \mathbf{lfp}$, then $\{i \in I_X(\pi) \mid \exists_{\bar{a}}\ \pi_i = \langle \mathbb{t}, X, \bar{a} \rangle\}$ is finite.*

Note that any infinite path in a dependency graph consists only of nodes from $\mathcal{S}_{\mathcal{X}}$, as the nodes from $\mathcal{S}_{\mathcal{V}}$ have no successors. The following theorem shows that proof graphs provide an alternative semantics for the EFL fixpoint operator. It is the main result of this section.

Theorem 7.1. *For all $\mathfrak{A}$, θ and monotone $\mathcal{E}$, and for all $\alpha \in \mathbb{B}$, $X \in \mathsf{bnd}(\mathcal{E})$ and $\bar{a} \in A^{\mathsf{ar}(X)}$, the following are equivalent:*

- *$\bar{a} \in \mathbf{S}(\theta, \mathcal{E})(X) \iff \alpha = \mathbb{t}$,*
- *there exists a proof graph $\langle S, \to \rangle$ such that $\langle \alpha, X, \bar{a} \rangle \in S$.*

The proof of this theorem is rather involved. We will split it into a soundness proof and a completeness proof. The soundness proof, given in Section 7.1.3, shows that if such a proof graph exists with $\langle \alpha, X, \bar{a} \rangle \in S$, then $\bar{a} \in \mathbf{S}(\theta, \mathcal{E})$ iff $\alpha = \mathbb{t}$. The completeness proof in Section 7.1.4 shows the converse, namely that such a proof graph can always be found. Before we start on these proofs, we first show how Theorem 7.1 can be used to provide an alternative semantics for EFL, and illustrate the notion of proof graph with an example.

Remark. We could have chosen to not quantify over η in Definition 7.1, but instead check that the implication holds for a specific η that is in some sense minimal (*i.e.*, that includes *only* the information provided by the successors). This is the approach that was taken in [CLW13]. We choose this stronger formulation in this thesis because it is slightly more convenient to work with in our proofs.

The quantification over $\mathfrak{B}$ is also not present in the definition of proof graph in [CLW13]. This quantification is actually not necessary to make proof graphs a sound and complete notion, but is introduced to ensure that dependency graphs contain enough information to extract evidence from them. The soundness proof for Theorem 7.1 therefore does not actually use this quantification at all, and for the completeness proof, the only implication is that we must add some extra nodes to our proof graph that would otherwise not be there. Only in Section 7.2 we need this quantification to be able to retrieve diagnostics from proof graphs.

The *proof graph semantics* of EFL is defined like the usual EFL semantics, except that the truth value for the fixpoint operator is established using proof graphs:

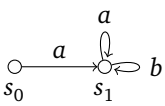
$\mathfrak{A}, \theta \models R\bar{t}$	iff $\bar{t}^{\mathfrak{A},\theta} \in \mathcal{I}(R)$
$\mathfrak{A}, \theta \models [X\bar{t} : \mathcal{E}]$	iff there is a proof graph for $\mathfrak{A}, \theta$ and $\mathcal{E}$ that has a node $\langle \mathbb{t}, X, \bar{t}^{\mathfrak{A},\theta} \rangle$
$\mathfrak{A}, \theta \models \neg\varphi$	iff $\mathfrak{A}, \theta \models \varphi$ does not hold
$\mathfrak{A}, \theta \models \varphi \vee \psi$	iff $\mathfrak{A}, \theta \models \varphi$ or $\mathfrak{A}, \theta \models \psi$
$\mathfrak{A}, \theta \models \exists_x\ \varphi$	iff $\mathfrak{A}, \theta[x \mapsto a] \models \varphi$ for some $a \in A$

Although $\not\models$ is usually given as the complement of $\models$, note that Theorem 7.1 is strong enough to turn this around, by defining $\models$ to be the complement of the relation $\not\models$ defined by:

$\mathfrak{A}, \theta \not\models R\bar{t}$	iff $\bar{t}^{\mathfrak{A},\theta} \notin \mathcal{I}(R)$
$\mathfrak{A}, \theta \not\models [X\bar{t} : \mathcal{E}]$	iff there is a proof graph for $\mathfrak{A}, \theta$ and $\mathcal{E}$ that has a node $\langle \mathbb{f}, X, \bar{t}^{\mathfrak{A},\theta} \rangle$
$\mathfrak{A}, \theta \not\models \neg\varphi$	iff $\mathfrak{A}, \theta \not\models \varphi$ does not hold
$\mathfrak{A}, \theta \not\models \varphi \vee \psi$	iff $\mathfrak{A}, \theta \not\models \varphi$ and $\mathfrak{A}, \theta \not\models \psi$
$\mathfrak{A}, \theta \not\models \exists_x \varphi$	iff $\mathfrak{A}, \theta[x \mapsto a] \not\models \varphi$ for all $a \in A$

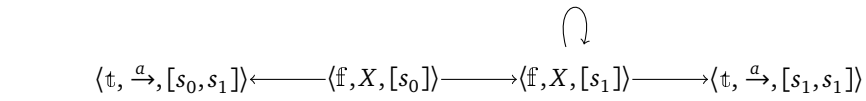
So for any formula $[X\bar{t} : \mathcal{E}]$, using one of the variants of the proof graph semantics above, we get a 'free' proof graph explaining why the formula holds or why it does not hold. We can in fact assume that we have such a proof graph for *any* EFL formula, by using the fact that we can add a redundant fixpoint to it: if $X_0 \notin \mathsf{bnd}(\varphi) \cup \mathsf{fv}(\varphi)$, then $[X_0 : \mathbf{lfp}\, X_0 = \varphi]$ is equivalent to φ. This construction is convenient when we extract evidence for the validity or invalidity of formulas from proof graphs, because we can always associate a single proof graph with every EFL formula φ, rather than having to present a collection of proof graphs for every fixpoint subformula of φ.

Example. Consider the following labelled transition system (LTS) and EFL formula φ, expressing that only finitely many a steps can be taken from s_0:



$$\varphi = [Xs_0 : \mathbf{lfp}\, Xs = \forall_{s'}\, s \xrightarrow{a} s' \Rightarrow Xs']$$

We assume that the LTS is represented as a model $\mathfrak{A}$ that has $\{s_0, s_1\}$ as its domain of discourse, and that additionally only defines the binary relations $\xrightarrow{a}$ and $\xrightarrow{b}$. A proof graph for $\mathfrak{A} \not\models \varphi$ might be the following:



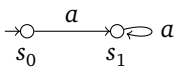
To convince ourselves that this is indeed a proof graph, we need to check that the graph is a dependency graph, and that the graph satisfies the requirement from Definition 7.2. Starting with the latter, note that the only infinite paths in the graph are those including the self-loop. On all such paths, the most significant (and only) second-order variable occurring infinitely often is X, which has the **lfp** operator associated with it. Furthermore, all nodes on this path prove that some state is *not* part of the solution of X (*i.e.*, $\alpha = \mathbb{f}$). This infinite path is therefore allowed by Definition 7.2.

Intuitively, this cycle is allowed because we are giving a coinductive proof to show that s_0 and s_1 are not in the solution of X: s_0 and s_1 are not in the first approximation for X (because it is a least fixpoint, the first approximation is the bottom element of the lattice, which assigns the empty relation to X), and because s_0 and s_1 are not in the next approximation if s_1 is not in the current approximation, s_0 and s_1 cannot be in the

next approximation of X (this is the information that is encoded in the edge relation of the proof graph). This reasoning can be repeated *ad infinitum* to show that s_0 and s_1 will *never* be in any approximation of X. The infinitary character of this reasoning is the reason we need this cycle in the proof graph.

To check that the graph is a dependency graph, consider for instance the node with the self-loop. We need to show that for any submodel $\mathfrak{B}$ of $\mathfrak{A}$ and any environment η such that $\mathfrak{B}, \eta \models s_1 \xrightarrow{a} s_1$ and $\mathfrak{B}, \eta \not\models Xs_1$, also $\mathfrak{B}, \eta \not\models \forall_{s'}\, s_1 \xrightarrow{a} s' \Rightarrow Xs'$. As the truth value of universally quantified formulas is preserved in submodels, we only need to consider the case $\mathfrak{B} = \mathfrak{A}$. In this model, $\mathfrak{A}, \eta \not\models \forall_{s'}\, s_1 \xrightarrow{a} s' \Rightarrow Xs'$ simplifies to $\mathfrak{A}, \eta \not\models s_1 \xrightarrow{a} s_1 \Rightarrow Xs_1$, which is indeed the case because we assumed that $s_1 \xrightarrow{a} s_1$ holds in $\mathfrak{A}, \eta$, but not Xs_1.

Every node in this proof of invalidity represents a statement that has a truth value that corresponds to the first element of the node. For instance, we can see from this proof graph that $s_0 \xrightarrow{a} s_1$ is valid. Note that we are taking some liberty in our notation here; to say that $s_0 \xrightarrow{a}{}^{\mathfrak{A}} s_1$, or even $\langle s_0, s_1 \rangle \in \xrightarrow{a}{}^{\mathfrak{A}}$, would be more accurate. Using the information about truths and falsehoods in $\mathfrak{A}$ allows us to construct the following submodel of $\mathfrak{A}$:



On this submodel, the EFL formula still does not hold, and moreover this can be proven with the same proof graph. Also note that it is essential that first-order relations are part of the proof graph: they provide us with information about which transition from s_1 (the a- or the b-transition) is causing the formula not to hold. Had the quantification over $\mathfrak{B}$ not been present in Definition 7.1 (but had we instead used $\mathfrak{B} = \mathfrak{A}$), then the leftmost node and the rightmost node would not necessarily have been included in the proof graph. ■

7.1.1 Monotonicity

Syntactic monotonicity of EFL formulas has a natural counterpart in the proof graph setting. It is captured in the following definition, which simulates the reachability analysis that pos and neg perform.

Definition 7.3. *A dependency graph $\langle S, \to \rangle$ is* monotone *iff for all $\langle \alpha, X, \bar{a} \rangle, \langle \neg\alpha, X, \bar{c} \rangle \in S$, if there is a path from $\langle \alpha, X, \bar{a} \rangle$ to $\langle \neg\alpha, X, \bar{c} \rangle$, then this path goes via a node $\langle \beta, Y, \bar{d} \rangle \in S$ with $Y <_{\mathcal{E}} X$.*

The claim that monotonicity of dependency graphs corresponds to syntactic monotonicity of EFL formulas is substantiated by the following lemma. It is easy to see that if we apply this lemma repeatedly, we can always construct a monotone dependency graph out of any dependency graph for $\mathfrak{A}, \theta$ and some syntactically monotone system $\mathcal{E}$.

Lemma 7.1. *If $\langle S, \to \rangle$ is a dependency graph for $\mathfrak{A}, \theta$ and syntactically monotone $\mathcal{E}$, and there is a path $\langle \alpha, X, \bar{a} \rangle \to \ldots \to \langle \neg\alpha, X, \bar{c} \rangle$ in this graph which only visits nodes*

$\langle \beta, Y, \bar{d} \rangle$ such that $Y \not\leq_{\mathcal{E}} X$, then there is an edge e on this path such that $\langle S, \to \setminus \{e\} \rangle$ is a dependency graph for $\mathfrak{A}, \theta$ and $\mathcal{E}$.

Proof. Let $v_0 = \langle \alpha_0, Y_0, \bar{a}_0 \rangle \to \ldots \to \langle \alpha_n, Y_n, \bar{a}_n \rangle = v_n$, with $Y_0 = Y_n = X$, be such a path in $\langle S, \to \rangle$, and towards a contradiction, suppose that if we leave out one of these edges, the new graph is no longer a dependency graph. This must then be due to the first requirement in the dependency graph no longer holding, as this is the only requirement that requires edges to be present. Formally, we are therefore assuming that for all i there exist $\mathfrak{A} \restriction S \sqsubseteq \mathfrak{B} \sqsubseteq \mathfrak{A}$ and θ' such that $\theta' \equiv_{\mathsf{fv}(\mathcal{E})} \theta$ and $\mathfrak{B}, \theta' \models u$ for all $v_i^\bullet \setminus \{v_{i+1}\}$ but not $\mathfrak{B}, \theta'[\bar{x}_{Y_i} \mapsto \bar{a}_i]^{\alpha_i} \models \mathsf{fo}(\varphi_{Y_i})$.

Let $0 \leq i < n$. We show that

$$\mathsf{pos}_{\mathcal{E},X}(\varphi_{Y_i}) \Rightarrow \mathsf{pos}_{\mathcal{E},X}(\varphi_{Y_{i+1}}) \text{ and } \mathsf{neg}_{\mathcal{E},X}(\varphi_{Y_i}) \Rightarrow \mathsf{neg}_{\mathcal{E},X}(\varphi_{Y_{i+1}}) \text{ if } \alpha_i = \alpha_{i+1}, \text{ and}$$
$$\mathsf{pos}_{\mathcal{E},X}(\varphi_{Y_i}) \Rightarrow \mathsf{neg}_{\mathcal{E},X}(\varphi_{Y_{i+1}}) \text{ and } \mathsf{neg}_{\mathcal{E},X}(\varphi_{Y_i}) \Rightarrow \mathsf{pos}_{\mathcal{E},X}(\varphi_{Y_{i+1}}) \text{ if } \alpha_i \neq \alpha_{i+1}.$$

Suppose $\alpha_i = \alpha_{i+1}$. Let $\mathfrak{B}$ and θ' be such that $\mathfrak{A} \restriction S \sqsubseteq \mathfrak{B} \sqsubseteq \mathfrak{A}$, $\theta' \equiv_{\mathsf{fv}(\mathcal{E})} \theta$ and $\mathfrak{B}, \theta' \models u$ for all $v_i^\bullet \setminus \{v_{i+1}\}$, but not $\mathfrak{B}, \theta'[\bar{x}_{Y_i} \mapsto \bar{a}_i]^{\alpha_i} \models \mathsf{fo}(\varphi_{Y_i})$. Now define an environment η that is equal to θ', except for $\eta(Y_{n+1}) = \theta'(Y_{n+1}) \cup \{\bar{a}_{n+1}\}$. Because v_i is a node in the dependency graph $\langle S, \to \rangle$, and because $\mathfrak{B}, \eta \models u$ for *all* $u \in v_i^\bullet$, we know that $\mathfrak{B}, \eta[\bar{x}_{Y_i} \mapsto \bar{a}_i]^{\alpha_i} \models \mathsf{fo}(\varphi_{Y_i})$. We have therefore found two environments that only differ on Y_{n+1}, such that one environment assigns a larger relation to it than the other. Furthermore, $\mathsf{fo}(\varphi_{Y_i})$ does not hold for the environment in which Y_{n+1} is assigned the smaller relation, but it does hold for the environment in which Y_{n+1} is assigned the larger one. Given how the semantics of first-order logic is defined, this can only mean that there is some subformula $[Y_{i+1}\bar{t} : \varepsilon] \sqsubseteq \mathsf{fo}(\varphi_{Y_i})$ occurring in the scope of an even number of negations. By the definition of fo, there must therefore be a subformula $\psi = [Y_{i+1}\bar{t} : \mathcal{F}] \sqsubseteq \varphi_{Y_i}$ occurring in the scope of an even number of negations. From Definition 6.2, it immediately follows that for these subformulas, $\mathsf{pos}_{\mathcal{E},X}(\psi) = \mathsf{pos}_{\mathcal{E},X}(\varphi_{Y_{i+1}})$ and $\mathsf{neg}_{\mathcal{E},X}(\psi) = \mathsf{neg}_{\mathcal{E},X}(\varphi_{Y_{i+1}})$. A straightforward induction on the structure of φ_{Y_i} then yields $\mathsf{pos}_{\mathcal{E},X}(\varphi_{Y_i}) \Rightarrow \mathsf{pos}_{\mathcal{E},X}(\varphi_{Y_{i+1}})$ and $\mathsf{neg}_{\mathcal{E},X}(\varphi_{Y_i}) \Rightarrow \mathsf{neg}_{\mathcal{E},X}(\varphi_{Y_{i+1}})$.

The proof for the case that $\alpha_i \neq \alpha_{i+1}$ is similar.

Now suppose that all Y_{i+1} occur syntactically in φ_{Y_i}, and consider $\langle \neg\alpha, X, \bar{c} \rangle$. Suppose that $\neg\alpha = \mathbb{f}$. By Definition 7.1, we have $\bar{c} \notin \eta(X) \Rightarrow \mathfrak{B}, \eta[\bar{x}_{Y_{n-1}} \mapsto \bar{a}_{n-1}] \not\models \mathsf{fo}(\varphi_{Y_{n-1}}) \iff \alpha_{n-1} = \mathbb{t}$ for all $\mathfrak{A} \restriction S \sqsubseteq \mathfrak{B} \sqsubseteq \mathfrak{A}$ and $\eta \equiv_{\mathsf{fv}(\mathcal{E})} \theta$. If $\alpha_{n-1} = \neg\alpha$, this implies that X occurs in the scope of an even number of negations in $\varphi_{Y_{n-1}}$. Therefore, by Definition 6.2, $\neg\mathsf{neg}_{\mathcal{E},X}(\varphi_{Y_{n-1}})$. If $\alpha_{n-1} \neq \neg\alpha$, this implies that X occurs in the scope of an odd number of negations in $\varphi_{Y_{n-1}}$. Therefore, by Definition 6.2, $\neg\mathsf{pos}_{\mathcal{E},X}(\varphi_{Y_{n-1}})$. In either case, we can derive with the four implications above that $\neg\mathsf{pos}_{\mathcal{E},X}(\varphi_X)$, using induction on $n-1$. This contradicts the syntactic monotonicity of $\mathcal{E}$. □

7.1.2 Minimality

Theorem 7.1 shows that a proof graph contains *enough* information to prove that an EFL formula has a certain solution, but in many cases it would be useful to have a concise proof. We assume that we will be using this definition in the setting where we

are given a proof graph G by a PBES solver, and we wish to discard any irrelevant or redundant information from the proof. We are therefore looking for a subgraph of this proof graph that is 'as small as possible', in a similar vein to what happens in [Cla+95].

Definition 7.4. *A proof graph is* minimal *w.r.t. a proof graph G and a node v in G if and only if it is a subgraph of G, includes v and there is no subgraph of G with fewer vertices or edges that is a proof graph and includes v.*

Note that there are two aspects to minimality: no minimal proof graph contains a subgraph that is a proof graph, and minimality with respect to G means that any proof graph contained in G is the same size or larger. Minimising a graph is a difficult problem: we show that the problem is NP-hard by using the same technique as Sahni used in [Sah74] to show that minimising and/or-trees is an NP-complete problem.

Theorem 7.2. *Given a proof graph G for $\mathfrak{A}, \theta$ and $\mathcal{E}$, finding a minimal proof graph w.r.t. G and some node v is NP-hard.*

Proof. We prove the theorem by reducing CNF-satisfiability to the problem of finding a minimal proof graph. Let a CNF formula φ be given as a set V of variable names, a set C of clause names, and mappings $p : C \to 2^V$ and $n : C \to 2^V$ that give, for each clause, the variables that occur as positive (resp. negative) literals in that clause.

Consider the following EFL formula, evaluated on a model with domain of discourse $V \cup C$ and which includes the mappings p and n:

$$\begin{aligned}
\Phi = [P : \mathbf{lfp}\, P &= \bigwedge_{v \in V} X(v) \wedge \bigwedge_{c \in C} S(c) \\
\mathbf{lfp}\, S c &= \bigvee_{v \in p(c)} T(v) \vee \bigvee_{v \in n(c)} F(v) \\
\mathbf{lfp}\, X v &= T(v) \vee F(v) \\
\mathbf{lfp}\, F v &= \mathbb{t} \\
\mathbf{lfp}\, T v &= \mathbb{t}]
\end{aligned}$$

Let $Q = \{\langle \mathbb{t}, P, [\,]\rangle\} \cup \{\langle \mathbb{t}, S, [c]\rangle \mid c \in C\} \cup \{\langle \mathbb{t}, X, [v]\rangle \mid v \in V\}$. Consider the graph made from all nodes from Q and $\langle \mathbb{t}, T, [v]\rangle$ and $\langle \mathbb{t}, F, [v]\rangle$ for *all* $v \in V$, and transitions from every node to the nodes referred to in the associated right-hand side. It is straightforward to check that this graph is a proof graph. We will argue that φ is satisfiable if and only if there is a proof graph with $1 + |C| + 2|V|$ nodes for Φ, and that this is also the minimum number of nodes needed in any proof graph for Φ. Minimizing the previously defined proof graph (with respect to itself and $\langle \mathbb{t}, P, [\,]\rangle$) would therefore solve the encoded CNF-satisfiability problem.

Suppose that there is a proof graph of size $1 + |C| + 2|V|$. Notice that in any proof graph that proves Φ, we must include all nodes in Q, as the right-hand side of P refers to all $X(v)$ and all $S(c)$. Because of the definition of X, either $\langle \mathbb{t}, T, [v]\rangle$ or $\langle \mathbb{t}, F, [v]\rangle$ must be in the graph for every $v \in V$. Therefore, only $\langle \mathbb{t}, F, [v]\rangle$ or $\langle \mathbb{t}, T, [v]\rangle$ is included for all $v \in V$, because apart from Q—which has size $1 + |C| + |V|$—there can only be an additional $|V|$ nodes in our proof graph. In that case, the assignment that assigns $\mathbb{f}$ to all nodes for which $\langle \mathbb{t}, F, [v]\rangle$ occurs in the graph and $\mathbb{t}$ to all nodes for which

$\langle \mathbb{t}, T, [v] \rangle$ occurs in the graph is a satisfying assignment for φ (this can easily be seen from the definition of S).

Conversely, given a satisfying assignment for φ, it is easy to check that the graph constructed from nodes $\langle \mathbb{t}, F, [v] \rangle$ for v which do not hold in the assignment and $\langle \mathbb{t}, T, [v] \rangle$ for v which hold in the assignment and Q, with edges from every node to the nodes that are referred to in their corresponding right-hand sides, is a proof graph. □

7.1.3 Soundness

Call two nodes $\langle \alpha, X, \bar{a} \rangle$ and $\langle \beta, Y, \bar{b} \rangle$ *contradictory* if $X = Y$ and $\bar{a} = \bar{b}$, but $\alpha \neq \beta$. As a preparation for our soundness proof, we would like to first establish that a proof graph cannot contain a proof for two contradictory nodes.

Lemma 7.2. *If $\langle S, \rightarrow \rangle$ is a proof graph for $\mathfrak{A}, \theta$ and $\mathcal{E}$, then no two nodes in S are contradictory.*

Proof. Let $v = \langle \alpha, X, \bar{a} \rangle$ and $v' = \langle \neg\alpha, X, \bar{a} \rangle$ be contradictory nodes. If $X \notin \mathsf{bnd}^*(\mathcal{E})$, then it follows immediately from Definition 7.1 that $v \in S$ implies $v' \notin S$, because that definition requires that $\mathfrak{A}, \theta \models v$ and $\mathfrak{A}, \theta \models v'$.

If $X \in \mathsf{bnd}^*(\mathcal{E})$, then note that if $v \in S$ and $v' \in S$, then $v^\bullet \cup v'^\bullet$ must contain contradictory nodes again. If this were not the case, then we could construct η such that $\eta \equiv_{\mathsf{fv}(\mathcal{E})} \theta$ such that $\mathfrak{A}, \eta \models u$ for all $u \in v^\bullet \cup v'^\bullet$. By Definition 7.1 then $\mathfrak{A}, \eta[\bar{x}_X \mapsto \bar{a}]^\alpha \models \mathsf{fo}(\varphi_X)$, but this is contradicted by $\mathfrak{A}, \eta[\bar{x}_X \mapsto \bar{a}]^{\neg\alpha} \models \mathsf{fo}(\varphi_X)$. The fact that $\rightarrow$ is consistent makes that the contradictory nodes in $v^\bullet \cup v'^\bullet$ cannot both be in $v^\bullet$ or $v'^\bullet$; ergo, we have one of them in $v^\bullet$, and one in $v'^\bullet$.

This argument can be repeated indefinitely, thus constructing two infinite paths through the proof graph in which the nodes are pairwise contradictory: the node at index i on one path contradicts the node at index i on the other. Obviously, if one path satisfies the parity condition from Definition 7.2, then the other does not, which contradicts the fact that $\langle S, \rightarrow \rangle$ is a proof graph. □

The claim that proof graphs are sound says that if a proof graph exists for $\mathfrak{A}, \theta$ and $\mathcal{E}$, then the nodes of that proof graph characterize the solution of $\mathcal{E}$. The proof is rather complicated, and uses induction on the size of proof graphs. We introduce two lemmas that allow us create smaller proof graphs from existing ones. The first lemma is very straightforward.

Lemma 7.3. *Let $\langle S, \rightarrow \rangle$ be a proof graph for $\mathfrak{A}, \theta$ and $\mathcal{E}$. If $s \in S$, and $S' \subseteq S$ is the set of all vertices reachable from s, then $\langle S', \rightarrow \cap\, S' \times S' \rangle$ is again a proof graph for $\mathfrak{A}, \theta$ and $\mathcal{E}$.*

Proof. Because all nodes in the subgraph have the same successors as the nodes in the original one, the subgraph must also be a dependency graph. It is moreover also a proof graph, because no infinite paths could have been introduced by removing nodes and edges from the original graph. □

The second lemma is more complicated. In our soundness proof, we will need a way to construct from a proof graph for a system $\mathcal{E}$ a proof graph for a subsystem of $\mathcal{E}$. The difficulty is that in a subsystem of $\mathcal{E}$, variables from $\mathcal{X}$ that were bound in $\mathcal{E}$ may have become free variables in the subsystem. It may therefore be the case that if the original proof graph was a proof graph for $\mathcal{E}$ and environment θ, the smaller proof graph is no longer a proof graph for the same environment. The next lemma tells us for which environments the smaller graph *is* indeed still a proof graph. Essentially, the subgraph is only a proof graph for those environments that agree on the information about variables in $\mathcal{X}$ that was encoded in the nodes that were removed from the original graph to obtain the subgraph.

Lemma 7.4. *Let $\langle S, \to\rangle$ be a proof graph for $\mathfrak{A}$ (with relation symbols $\mathcal{R}$), θ and $\mathcal{E}$, and let $\mathcal{E}' \sqsubseteq \mathcal{E}$. Define $S' = S \cap \mathcal{S}_{\mathsf{bnd}^*(\mathcal{E}')\cup\mathcal{R}}$ and $\to' = \to \cap (S' \times S')$.*

For all environments θ' such that $\theta' \equiv_{\mathsf{fv}(\mathcal{E})} \theta$ and $\mathfrak{A}, \theta' \models u$ for all $u \in (S'^{\bullet} \setminus S') \cap \mathcal{S}_{\mathsf{fv}(\mathcal{E}')}$, the graph $\langle S', \to'\rangle$ is a proof graph for $\mathfrak{A}, \theta'$ and $\mathcal{E}'$ (where $S'^{\bullet}$ is the postset w.r.t $\to$).

Proof. It is sufficient to show that $\langle S', \to'\rangle$ is a dependency graph; because $\to' \subseteq \to$, the requirement from Definition 7.2 still holds. We therefore check for every $v = \langle \alpha, X, \bar{a}\rangle \in S'$ that the requirements from Definition 7.1 hold. Distinguish cases on X:

Case $X \notin \mathsf{bnd}^*(\mathcal{E}')$, then we must show that $\mathfrak{A}, \theta' \models v$ and $v^{\bullet} = \emptyset$. By the definition of S', $X \in \mathcal{R}$, so $\mathfrak{A}, \theta' \models v$ is equivalent to $\bar{a} \in X^{\mathfrak{A}} \iff \alpha = \mathbb{t}$, which in turn is equivalent to $\mathfrak{A}, \theta \models v$.

Case $X \in \mathsf{bnd}^*(\mathcal{E}')$, then we know that

$$\begin{aligned} &\text{for all } \mathfrak{A} \restriction S \sqsubseteq \mathfrak{B} \sqsubseteq \mathfrak{A} \text{ and all } \eta \text{ such that } \eta \equiv_{\mathsf{fv}(\mathcal{E})} \theta, \\ &\quad \text{if } \mathfrak{B}, \eta \models u \text{ for all } u \in v^{\bullet} \text{ then } \mathfrak{B}, \eta[\bar{x}_X \mapsto \bar{a}]^{\alpha} \models \mathsf{fo}(\varphi_X). \end{aligned} \tag{$*$}$$

We need to prove the corresponding requirement for $\mathfrak{A}, \mathcal{E}'$ and θ'. By v° denote the postset of v with respect to $\to'$. Now let $\mathfrak{B}$ and η be such that $\mathfrak{A} \restriction S \sqsubseteq \mathfrak{B} \sqsubseteq \mathfrak{A}$ and $\eta \equiv_{\mathsf{fv}(\mathcal{E}')} \theta'$, and assume that $\mathfrak{B}, \eta \models u$ for all $u \in v^{\circ}$. We need to show that $\mathfrak{B}, \eta[\bar{x}_X \mapsto \bar{a}]^{\alpha} \models \mathsf{fo}(\varphi_X)$.

Note that $v^{\circ} \subseteq v^{\bullet}$. Let $u = \langle \beta, Y, \bar{b}\rangle$ be in $v^{\bullet} \setminus v^{\circ}$. Then we have $Y \notin \mathsf{bnd}^*(\mathcal{E}')$. If $Y \notin \mathsf{fv}(\mathcal{E}')$, then Y does not appear at all in $\mathcal{E}'$, and therefore its value in η does not influence the truth value of $\mathsf{fo}(\varphi_X)$, in which case the implication in ($*$) still holds if we replace $v^{\bullet}$ by $v^{\bullet} \setminus \{u\}$. If $Y \in \mathsf{fv}(\mathcal{E}')$, then $\mathfrak{A}, \theta' \models u$ by the assumption of the lemma, and also $\mathfrak{B}, \eta \models u$, because $\eta \equiv_{\mathsf{fv}(\mathcal{E}')} \theta'$.

Either way, we have $\mathfrak{B}, \eta \models u$ for all u in a set U that includes all nodes from $v^{\bullet}$ referring to variables that are relevant for the evaluation of $\mathsf{fo}(\varphi_X)$, allowing us to conclude from ($*$) that $\mathfrak{B}, \eta[\bar{x}_X \mapsto \bar{a}]^{\alpha} \models \mathsf{fo}(\varphi_X)$. □

We need one last lemma before we can present the soundness proof itself. In the soundness proof, we will find ourselves in a situation that $\mathfrak{A}, \eta^{\alpha} \models \mathsf{fo}(\varphi)$ for some $\mathfrak{A}, \eta, \alpha$ and φ, where we need to prove $\mathfrak{A}, \eta'^{\alpha} \models \varphi$ for some η' that closely resembles

η. The following lemma establishes sufficient conditions to prove $\mathfrak{A}, \eta' \,^{\alpha}\!\models \varphi$ in this situation.

We introduce a special 'direct subformula' relation to easily identify those subformulas of a formula φ that have been replaced in $\mathsf{fo}(\varphi)$. Let $\psi \sqsubseteq^1 \varphi$ denote that $\psi \sqsubseteq \varphi$ and there is no $[Y\bar{t} : \mathcal{F}] \sqsubseteq \varphi$ such that $\psi \sqsubset [Y\bar{t} : \mathcal{F}]$.

Lemma 7.5. *Let φ be an EFL formula, $\mathfrak{A}$ a structure, θ and η environments, and $\alpha \in \mathbb{B}$.*

$$\begin{aligned}
&\textit{If } \theta \equiv_{\mathsf{fv}(\varphi)} \eta \\
&\textit{and } \mathfrak{A}, \theta \,^{\beta}\!\models Y\bar{t} \textit{ implies } \mathfrak{A}, \eta \,^{\beta}\!\models [Y\bar{t} : \mathcal{F}] \textit{ for all } [Y\bar{t} : \mathcal{F}] \sqsubseteq^1 \varphi, \beta \in \mathbb{B} \\
&\textit{and } \mathfrak{A}, \theta \,^{\alpha}\!\models \mathsf{fo}(\varphi) \\
&\textit{then } \mathfrak{A}, \eta \,^{\alpha}\!\models \varphi.
\end{aligned}$$

Proof. Assume the left hand side of the implication stated by the lemma. The proof is straightforward by structural induction on φ. As our induction hypothesis, we assume that we have the result for all strict subformulas of φ.

Case $\varphi = R\bar{t}$ (with R a relation name). Then $\mathsf{fo}(\varphi) = \varphi$ and $\mathfrak{A}, \theta \,^{\alpha}\!\models \mathsf{fo}(\varphi)$ is equivalent to $\mathfrak{A}, \eta \,^{\alpha}\!\models \varphi$ because θ and η agree on all free variables in $\bar{t}$.

Case $\varphi = [Y\bar{t} : \mathcal{F}]$. Then $\mathsf{fo}(\varphi) = Y\bar{t}$, so $\mathfrak{A}, \theta \,^{\alpha}\!\models Y\bar{t}$. Because $[Y\bar{t} : \mathcal{F}] \sqsubseteq^1 \varphi$, also $\mathfrak{A}, \eta \,^{\alpha}\!\models \varphi$.

Case $\varphi = \neg\chi$. Then $\mathfrak{A}, \theta \,^{\alpha}\!\not\models \mathsf{fo}(\chi)$, so $\mathfrak{A}, \theta \,^{\neg\alpha}\!\models \mathsf{fo}(\chi)$. Note that the direct subformulas of χ that are of the shape $[Y\bar{t} : \mathcal{F}]$ are also direct subformulas of φ. Therefore, by the induction hypothesis, $\mathfrak{A}, \eta \,^{\neg\alpha}\!\models \chi$, so $\mathfrak{A}, \eta \,^{\alpha}\!\models \neg\chi$.

Case $\varphi = \chi \vee \psi$. Then $\mathfrak{A}, \theta \,^{\alpha}\!\models \mathsf{fo}(\chi) \vee \mathsf{fo}(\psi)$. Distinguish two cases.

If $\alpha = \mathbb{t}$, then either $\mathfrak{A}, \theta \,^{\alpha}\!\models \mathsf{fo}(\chi)$ or $\mathfrak{A}, \theta \,^{\alpha}\!\models \mathsf{fo}(\psi)$, and by the induction hypothesis then $\mathfrak{A}, \eta \,^{\alpha}\!\models \chi$ or $\mathfrak{A}, \eta \,^{\alpha}\!\models \psi$, which is enough to conclude that $\mathfrak{A}, \eta \,^{\alpha}\!\models \varphi$.

If $\alpha = \mathbb{f}$, then both $\mathfrak{A}, \theta \,^{\alpha}\!\models \mathsf{fo}(\chi)$ and $\mathfrak{A}, \theta \,^{\alpha}\!\models \mathsf{fo}(\psi)$. By the induction hypothesis $\mathfrak{A}, \eta \,^{\alpha}\!\models \chi$ and $\mathfrak{A}, \eta \,^{\alpha}\!\models \psi$. Therefore $\mathfrak{A}, \eta \,^{\alpha}\!\models \varphi$.

Case $\varphi = \exists_y \chi$. Then $\mathfrak{A}, \theta \,^{\alpha}\!\models \exists_y \mathsf{fo}(\chi)$, so by the semantics of $\exists$, there must be some $c \in A$ such that $\mathfrak{A}, \theta[y \mapsto c] \,^{\alpha}\!\models \mathsf{fo}(\chi)$. Note that $\eta[y \mapsto c] \equiv_{\mathsf{fv}(\varphi)} \theta[y \mapsto c]$, so by the induction hypothesis, $\mathfrak{A}, \eta[y \mapsto c] \,^{\alpha}\!\models \chi$. Therefore also $\mathfrak{A}, \eta \,^{\alpha}\!\models \varphi$. □

The final lemma of this section forms the proof of soundness. It is stated a bit more general than the soundness claim, which says that if $\langle S, \to \rangle$ is a proof graph for $\mathfrak{A}, \theta$ and $\mathcal{E}$, and includes $\langle \alpha, X, \bar{t}^{\mathfrak{A},\theta} \rangle$, then $\mathfrak{A}, \theta \,^{\alpha}\!\models [X\bar{t} : \mathcal{E}]$. The more general formulation is used in the inductive proof to make the induction hypothesis strong enough. Note that $\mathfrak{A}, \theta \,^{\alpha}\!\models [X\bar{t} : \mathcal{E}]$ is by definition equivalent to $\mathfrak{A}, \mathbf{S}(\theta, \mathcal{E}) \models \langle \alpha, X, \bar{t}^{\mathfrak{A},\theta} \rangle$.

Lemma 7.6. *If $\langle S, \to \rangle$ is a proof graph for $\mathfrak{A}, \theta$ and monotone $\mathcal{E}$, then*

$$\mathfrak{A}, \mathbf{S}(\theta, \mathcal{E}) \models v \textit{ for all } v \in S \cap \mathcal{S}_{\mathsf{bnd}(\mathcal{E})}.$$

Proof. The proof goes by induction on the size of S. If $S \cap \mathcal{S}_{\mathsf{bnd}(\mathcal{E})} = \emptyset$, then the lemma holds vacuously, so in particular the lemma holds if $S = \emptyset$. So let $\langle S, \to \rangle$ be a proof graph for $\mathfrak{A}, \theta$ and $\mathcal{E}$, and assume that the lemma holds for all proof graphs (for arbitrary $\mathfrak{A}', \theta'$ and $\mathcal{E}'$) with fewer nodes. We shall sometimes refer to this hypothesis as the *outer induction hypothesis*.

Suppose X is the smallest (with respect to $<_{\mathcal{E}}$) variable such that there is a node $\langle \alpha, X, \bar{a} \rangle \in S$, and suppose that index i is such that $\mathcal{E}^i = (\sigma X \bar{x} = \varphi)\mathcal{E}^{i+1}$. Note that by Lemma 7.2, S contains no contradictory nodes. We can therefore define an environment $\hat{\theta}$ such that for all Y and $\bar{a}$:

$$\bar{a} \in \hat{\theta}(Y) \text{ iff } \begin{cases} \alpha = \mathbb{t}, & \text{if } Y = X \text{ and } \langle \alpha, Y, \bar{a} \rangle \in S \\ \bar{a} \in \mathbf{S}(\theta, \mathcal{E})(Y), & \text{otherwise.} \end{cases}$$

Note that $\langle S, \to \rangle$ is a proof graph for $\mathfrak{A}, \mathbf{S}(\theta, \mathcal{E})$ and $\mathcal{E}$, because $\mathbf{S}(\theta, \mathcal{E})$ and θ agree on all variables outside $\mathsf{bnd}(\mathcal{E})$ (by Lemma 6.1), and the definition of dependency graph (Definition 7.1) does not require anything about the environment on variables in $\mathsf{bnd}(\mathcal{E})$. Then $\langle S', \to' \rangle = \langle S \cap \mathcal{S}_{\mathsf{bnd}^*(\mathcal{E}^{i+1}) \cup \mathcal{R}}, \to \cap (S' \times S') \rangle$ is a proof graph for $\mathfrak{A}, \hat{\theta}$ and $\mathcal{E}^{i+1}$ by Lemma 7.4, because $\mathfrak{A}, \hat{\theta} \models u$ exactly for all $u \in S \cap \mathcal{S}_{\{X\}}$, which are the only nodes in $(S \setminus S') \cap \mathcal{S}_{\mathsf{fv}(\mathcal{E}^{i+1})}$ (and therefore in $(S'^{\bullet} \setminus S') \cap \mathcal{S}_{\mathsf{fv}(\mathcal{E}^{i+1})}$). By the induction hypothesis then, $\mathfrak{A}, \mathbf{S}(\hat{\theta}, \mathcal{E}^{i+1}) \models \nu$ for all $\nu \in S' \cap \mathcal{S}_{\mathsf{bnd}(\mathcal{E}^{i+1})}$. We will show later that

$$\mathfrak{A}, \mathbf{S}(\theta, \mathcal{E}) \models \nu \text{ for all } \nu \in S \cap \mathcal{S}_{\{X\}}. \tag{$*$}$$

Once this is established, we know that $\hat{\theta} = \mathbf{S}(\theta, \mathcal{E})$, and therefore $\mathfrak{A}, \mathbf{S}(\mathbf{S}(\theta, \mathcal{E}), \mathcal{E}^{i+1}) \models \nu$ for all $\nu \in S' \cap \mathcal{S}_{\mathsf{bnd}(\mathcal{E}^{i+1})}$. By Lemma 6.4, also $\mathfrak{A}, \mathbf{S}(\theta, \mathcal{E}) \models \nu$ for these ν, and combined with $(*)$ this proves $\mathfrak{A}, \mathbf{S}(\theta, \mathcal{E}) \models \nu$ for all $\nu \in S \cap \mathcal{S}_{\mathsf{bnd}(\mathcal{E})}$, which was the statement of the lemma.

Our only proof obligation is to show that $(*)$ holds. Suppose therefore that $\langle \alpha, X, \bar{a} \rangle$ is a node in S. We show that $\mathfrak{A}, \mathbf{S}(\theta, \mathcal{E}) \models \langle \alpha, X, \bar{a} \rangle$. At this point, we first establish sufficient conditions to prove $\mathfrak{A}, \theta'[\bar{x} \mapsto \bar{b}]^{\alpha} \models \varphi$, for given θ', α and $\bar{b}$. We state it as a sublemma, the proof of which we will postpone until we have completed our current argument.

Sublemma 7.6.1. *Let $\langle S', \to' \rangle$ be a subgraph of $\langle S, \to \rangle$ that is again a proof graph for $\mathfrak{A}, \theta$ and $\mathcal{E}$. Let $\nu = \langle \alpha, X, \bar{b} \rangle \in S$ and θ' an environment such that $\theta' \equiv_{\mathsf{fv}(\mathcal{E})} \theta$.*

If $\mathfrak{A}, \theta' \models u$ for all $u \in (\{\nu\} \cup (S' \setminus \mathcal{S}_{\mathsf{bnd}(\mathcal{E})}))^{\bullet} \cap \mathcal{S}_{\mathsf{bnd}(\mathcal{E})}$, then $\mathfrak{A}, \theta'[\bar{x} \mapsto \bar{b}]^{\alpha} \models \varphi$.

We will use this lemma to prove that $\mathfrak{A}, \mathbf{S}(\theta, \mathcal{E}) \models \langle \alpha, X, \bar{a} \rangle$. Let $\langle S^*, \to^* \rangle$ be the part of $\langle S, \to \rangle$ reachable from $\langle \alpha, X, \bar{a} \rangle$. This is still a proof graph for $\mathfrak{A}, \theta$ and $\mathcal{E}$ by Lemma 7.3.

Assume $\sigma = \mathbf{lfp}$ (the proof for $\sigma = \mathbf{gfp}$ is dual), and distinguish cases based on the value of α.

Case $\alpha = \mathbb{t}$**.** Observe that no node in S^* can have $\langle \alpha, X, \bar{a} \rangle$ as a successor, because then there is a cycle with $\langle \alpha, X, \bar{a} \rangle$ on it, violating the parity condition (see Definition 7.2, remember that X was chosen the smallest variable with respect to $<_{\mathcal{E}}$).

Therefore, the reachable graph from any $u \in S^* \cap \mathcal{S}_{\mathsf{bnd}(\mathcal{E})} \setminus \{v\}$ is smaller than $\langle S^*, \to^* \rangle$, and is a proof graph for $\mathfrak{A}, \theta$ and $\mathcal{E}$ by Lemma 7.3. By the outer induction hypothesis therefore, $\mathbf{S}(\theta, \mathcal{E}) \models u$ for such u. It now follows from Sublemma 7.6.1 that $\mathfrak{A}, \mathbf{S}(\theta, \mathcal{E})[\bar{x} \mapsto \bar{a}]^\alpha \models \varphi$.

Case $\alpha = \mathbb{f}$. We need to show $\bar{a} \notin \mathbf{S}(\theta, \mathcal{E})(X)$. Let $F = \{\bar{b} \mid \langle \alpha, X, \bar{b} \rangle \in S^*\}$. Let $\mathbf{T} = \mathbf{T}_{\mathcal{E}^i}^{\mathbf{S}(\theta,\mathcal{E})}$; note that by Lemma 6.5, $\mathbf{S}(\theta, \mathcal{E})(X) = \mathbf{lfp}\,\mathbf{T}$. Let $R = \mathbf{S}(\theta, \mathcal{E})(X) \setminus F$. Then, because $\mathbf{T}$ is monotone, and because $R \subseteq \mathbf{S}(\theta, \mathcal{E})(X)$, we have $\mathbf{T}(R) \subseteq \mathbf{T}(\mathbf{S}(\theta, \mathcal{E})(X))$, and therefore (because $\mathbf{S}(\theta, \mathcal{E})(X)$ is a fixpoint of $\mathbf{T}$) $\mathbf{T}(R) \subseteq \mathbf{S}(\theta, \mathcal{E})(X)$.

We will show later that $\mathbf{T}(R) \cap F = \emptyset$. Then $\mathbf{T}(R) \subseteq R$: suppose $\bar{b} \in \mathbf{T}(R)$ and $\bar{b} \notin R$ for some $\bar{b}$, then $\bar{b} \notin F$ because $\mathbf{T}(R) \cap F = \emptyset$, therefore $\bar{b} \notin \mathbf{S}(\theta, \mathcal{E})(X)$ because $\bar{b} \notin R$ (see the definition of R), but this contradicts $\mathbf{T}(R) \subseteq \mathbf{S}(\theta, \mathcal{E})(X)$.

Define for ordinals $\gamma > 0$ and for $S \subseteq A^{\mathsf{ar}(X)}$:

$$\mathbf{lfp}_S^0\,\mathbf{T} = S \qquad\qquad \mathbf{lfp}_S^\gamma\,\mathbf{T} = \mathbf{T}(\bigcup_{\delta<\gamma} \mathbf{lfp}_S^\delta\,\mathbf{T})$$

Note that $\mathbf{lfp}_\emptyset^{\gamma+1}\,\mathbf{T} = \mathbf{lfp}^\gamma\,\mathbf{T}$ for all γ. Also, $\mathbf{lfp}_R^1\,\mathbf{T} = \mathbf{T}(R)$ and $\mathbf{lfp}_R^2\,\mathbf{T} = \mathbf{T}(\mathbf{lfp}_R^0\,\mathbf{T} \cup \mathbf{lfp}_R^1\,\mathbf{T}) = \mathbf{T}(R \cup \mathbf{T}(R)) = \mathbf{T}(R) = \mathbf{lfp}_R^1\,\mathbf{T}$, and similarly, by induction, $\mathbf{lfp}_R^\gamma\,\mathbf{T} = \mathbf{T}(R)$ for all $\gamma > 0$. We know that $\emptyset \subseteq R \subseteq \mathbf{S}(\theta, \mathcal{E})(X)$, and a straighforward transfinite induction shows that $\mathbf{lfp}_\emptyset^\gamma\,\mathbf{T} \subseteq \mathbf{lfp}_R^\gamma\,\mathbf{T} \subseteq \mathbf{S}(\theta, \mathcal{E})(X)$ for all γ. There is some γ such that $\mathbf{lfp}_\emptyset^\gamma\,\mathbf{T} = \mathbf{S}(\theta, \mathcal{E})(X)$, and for that γ, also $\mathbf{S}(\theta, \mathcal{E})(X) = \mathbf{lfp}_R^\gamma\,\mathbf{T} = \mathbf{T}(R)$. Because $\bar{a} \in F$ and $\mathbf{T}(R) \cap F = \emptyset$, $\bar{a} \notin \mathbf{T}(R)$, and then also $\bar{a} \notin \mathbf{S}(\theta, \mathcal{E})(X)$.

Proof for $\mathbf{T}(R) \cap F = \emptyset$. Abbreviate $\mathbf{S}(\mathbf{S}(\theta, \mathcal{E})[X \mapsto R], \mathcal{E}^{i+1})$ to θ'. Filling in the definition of the predicate transformer, we prove for all $\bar{b} \in F$ (*i.e.*, for all $\langle \alpha, X, \bar{b} \rangle \in S^*$) that $\mathfrak{A}, \theta'[\bar{x} \mapsto \bar{b}] \not\models \varphi$. Note that $\theta \equiv_{\mathsf{fv}(\mathcal{E})} \theta'$ by Lemma 6.1.

Note that from any node $u = \langle \mathbb{t}, X, \bar{b} \rangle \in S^*$ there cannot be a path to v, as this would violate the parity condition of Definition 7.2 since u is reachable from v. The reachable subgraph from such u is therefore smaller than $\langle S, \to \rangle$, and therefore by Lemma 7.3 and the induction hypothesis, $\mathfrak{A}, \mathbf{S}(\theta, \mathcal{E}) \models u$ for such u. Furthermore, $\mathfrak{A}, \mathbf{S}(\theta, \mathcal{E})[X \mapsto R] \models u$, because of how R is defined. Because also $R \cap F = \emptyset$, we have[2]:

$$\mathfrak{A}, \mathbf{S}(\theta, \mathcal{E})[X \mapsto R] \models u \text{ for all } u \in S^* \cap \mathcal{S}_{\{X\}}. \tag{$\dagger$}$$

Consider the graph $\langle S', \to' \rangle = \langle S^* \cap \mathcal{S}_{\mathsf{bnd}^*(\mathcal{E}^{i+1}) \cup \mathcal{R}}, \to \cap (S' \times S') \rangle$. Observe that $(S'^\bullet \setminus S') \cap \mathsf{fv}(\mathcal{E}) \subseteq S^* \cap \mathcal{S}_{\{X\}}$, so because of ($\dagger$) we may apply Lemma 7.4 to derive that $\langle S', \to' \rangle$ is a proof graph for $\mathfrak{A}, \mathbf{S}(\theta, \mathcal{E})[X \mapsto R]$ and $\mathcal{E}^{i+1}$. By the induction hypothesis, $\mathfrak{A}, \theta' \models u$ for all $u \in S' \cap \mathcal{S}_{\mathsf{bnd}(\mathcal{E}^{i+1})}$. By ($\dagger$) and Lemma 6.1, also $\mathfrak{A}, \theta' \models u$ for all $u \in S^* \cap \mathcal{S}_{\{X\}}$, and therefore $\mathfrak{A}, \theta' \models u$ for all $u \in S^* \cap \mathcal{S}_{\mathsf{bnd}(\mathcal{E})}$.

We can now apply Sublemma 7.6.1 again to derive $\mathfrak{A}, \theta'[\bar{x} \mapsto \bar{b}] \not\models \varphi$. □

[2] The argument here is that we also know $\mathfrak{A}, \mathbf{S}(\theta, \mathcal{E})[X \mapsto R] \models u$ for all $u = \langle \mathbb{f}, X, \bar{b} \rangle \in S^*$, because $\bar{b} \in F$ for such u, and $R \cap F = \emptyset$.

We have now concluded the proof for Lemma 7.6, but have left open the proof for Sublemma 7.6.1. To prove this sublemma, we take a two-stage approach. First we give sufficient requirements on $\nu^\bullet$ to derive $\mathfrak{A}, \theta'[\bar{x} \mapsto \bar{b}]^\alpha \models \varphi$. We then show that we can instead pose requirements on a subset of $\mathcal{S}_{\mathsf{bnd}(\mathcal{E})}$ to come to the same result. Effectively, Sublemma 7.6.1 isolates the proof obligations for nodes in the proof graph that refer to varables bound in subformulas of φ.

Sublemma 7.6.2. *For all θ' and $\nu = \langle \alpha, X, \bar{b} \rangle \in S$, if for all $u = \langle \beta, Y, \bar{c} \rangle \in \nu^\bullet$,*

- *$\mathfrak{A}, \theta' \models u$ if $Y \in \mathsf{fv}(\varphi)$, and*
- *$\mathfrak{A}, \mathbf{S}(\theta', \mathcal{F}) \models u$ if $Y \in \mathsf{bnd}^*(\varphi)$ and $[Y\bar{t} : \mathcal{F}] \sqsubseteq^1 \varphi$ for some $\bar{t}$,*

then $\mathfrak{A}, \theta'[\bar{x} \mapsto \bar{b}]^\alpha \models \varphi$.

Proof. Consider the environment η defined as follows:

$$\eta(Y) = \begin{cases} \mathbf{S}(\theta', \mathcal{F})(Y) & \text{if } Y \in \mathsf{bnd}^*(\varphi) \text{ and } [Y\bar{t} : \mathcal{F}] \sqsubseteq^1 \varphi \text{ for some } \bar{t}, \\ \theta'(Y) & \text{otherwise.} \end{cases}$$

Note that this defines a unique η because of the unique naming assumption (every Y can only be defined in a single subsystem $\mathcal{F}$). We have chosen η such that

1. $\eta \equiv_{\mathsf{fv}(\varphi)} \theta'$, and
2. $\mathfrak{A}, \eta^\beta \models Y\bar{t}$ implies $\mathfrak{A}, \theta'^\beta \models [Y\bar{t} : \mathcal{F}]$ for all $[Y\bar{t} : \mathcal{F}] \sqsubseteq^1 \varphi$ and all $\beta \in \mathbb{B}$.

Note that for all $u \in \nu^\bullet$, if $\mathfrak{A}, \eta \not\models u$, then $Y \notin \mathsf{fv}(\mathsf{fo}(\varphi))$. Therefore, there is some η' such that $\eta' \equiv_{\mathsf{fv}(\mathsf{fo}(\varphi))} \eta$ such that $\mathfrak{A}, \eta' \models u$ for all $u \in \nu^\bullet$. By Definition 7.1 (because $\nu \in S$), then $\mathfrak{A}, \eta'[\bar{x} \mapsto \bar{b}]^\alpha \models \mathsf{fo}(\varphi)$. Because $\eta' \equiv_{\mathsf{fv}(\mathsf{fo}(\varphi))} \eta$, also:

3. $\mathfrak{A}, \eta[\bar{x} \mapsto \bar{b}]^\alpha \models \mathsf{fo}(\varphi)$.

The proof for the latter is by a straightforward induction on the structure of $\mathsf{fo}(\varphi)$, appealing to Lemma 6.2 in the fixpoint operator case. We can now use Lemma 7.5 on observations 1–3 to conclude that $\mathfrak{A}, \theta'[\bar{x} \mapsto \bar{b}]^\alpha \models \varphi$. □

Now we have shown how to derive $\mathfrak{A}, \theta'[\bar{x} \mapsto \bar{b}]^\alpha \models \varphi$ by establishing some facts about the successors of $\nu = \langle \alpha, X, \bar{b} \rangle$, we show that for those $\langle \beta, Y, \bar{c} \rangle \in \nu^\bullet$ mentioned in the second bullet point in Sublemma 7.6.2, we can derive the required information if we know that $\mathfrak{A}, \theta' \models u$ for enough nodes in $\mathcal{S}_{\mathsf{bnd}(\mathcal{E})}$. In the cases $\alpha = \mathbb{t}$ and $\alpha = \mathbb{f}$ above, the induction hypothesis gives us this information for the required nodes in $\mathcal{S}_{\mathsf{bnd}(\mathcal{E}^{i+1})}$. In the case $\alpha = \mathbb{t}$ the induction hypothesis also gives it for the required nodes in $\mathcal{S}_{\{X\}}$; in the case $\alpha = \mathbb{f}$ the information for the required nodes in $\mathcal{S}_{\{X\}}$ can be derived because R was chosen to satisfy all X-nodes in S^*.

Sublemma 7.6.1. *Let $\langle S', \to' \rangle$ be a subgraph of $\langle S, \to \rangle$ that is still a proof graph for $\mathfrak{A}, \theta$ and $\mathcal{E}$. Let $\nu = \langle \alpha, X, \bar{b} \rangle \in S$ and θ' an environment such that $\theta' \equiv_{\mathsf{fv}(\mathcal{E})} \theta$.*

If $\mathfrak{A}, \theta' \models u$ for all $u \in (\{\nu\} \cup (S' \setminus \mathcal{S}_{\mathsf{bnd}(\mathcal{E})}))^\bullet \cap \mathcal{S}_{\mathsf{bnd}(\mathcal{E})}$, then $\mathfrak{A}, \theta'[\bar{x} \mapsto \bar{b}]^\alpha \models \varphi$.

Proof. We will be using Lemma 7.6.2 to reach our conclusion. Assume the left hand side of the implication, and note that $\mathfrak{A}, \theta \models u$ and therefore $\mathfrak{A}, \theta' \models u$ for all $u \in S' \cap \mathcal{S}_{\mathsf{fv}(\mathcal{E})}$ by Definition 7.1. Observe that $\mathsf{fv}(\varphi) \subseteq \mathsf{fv}(\mathcal{E}) \cup \mathsf{bnd}(\mathcal{E})$, so we have $\mathfrak{A}, \theta' \models \langle \beta, Y, \bar{c} \rangle$ for all $\langle \beta, Y, \bar{c} \rangle \in \nu^\bullet$ such that $Y \in \mathsf{fv}(\varphi)$, satisfying part of the precondition for Lemma 7.6.2.

We set out to prove the remainder of that precondition. Let $\langle \beta, Y, \bar{c} \rangle \in \nu^\bullet$ and $\mathcal{F}$ be such that $Y \in \mathsf{bnd}^*(\varphi)$ and $[Y\bar{t} : \mathcal{F}] \sqsubseteq^1 \varphi$ for some $\bar{t}$. We must prove $\mathfrak{A}, \mathbf{S}(\theta', \mathcal{F}) \models \langle \beta, Y, \bar{c} \rangle$.

Consider the graph $\langle S'', \to'' \rangle = \langle S' \cap \mathcal{S}_{\mathsf{bnd}^*(\mathcal{F}) \cup \mathcal{R}}, \to \cap (S'' \times S'') \rangle$. Because $\mathcal{F}$ occurs in φ, $\mathsf{fv}(\mathcal{F}) \subseteq \mathsf{fv}(\mathcal{E}) \cup \mathsf{bnd}(\mathcal{E})$. We therefore have $\mathfrak{A}, \theta' \models u$ for all $u \in (S''^\bullet \setminus S'') \cap \mathcal{S}_{\mathsf{fv}(\mathcal{F})}$, so by Lemma 7.4, $\langle S'', \to'' \rangle$ is a proof graph for $\mathfrak{A}, \theta'$ and $\mathcal{F}$. By the outer induction hypothesis then $\mathfrak{A}, \mathbf{S}(\theta', \mathcal{F}) \models u$ for all $u \in S'' \cap \mathcal{S}_{\mathsf{bnd}(\mathcal{F})}$. In particular, $\mathfrak{A}, \mathbf{S}(\theta', \mathcal{F}) \models \langle \beta, Y, \bar{c} \rangle$. □

7.1.4 Completeness

Given $\mathfrak{A}, \theta$ and $\mathcal{E}$ and some $\bar{a} \in A^*$ and $X \in \mathcal{X}$, we need to show that there is always a proof graph for $\mathfrak{A}, \theta$ and $\mathcal{E}$ that includes node $\langle \mathbb{t}, X, \bar{a} \rangle$ if $\bar{a} \in \mathbf{S}(\theta, \mathcal{E})(X)$, or that includes $\langle \mathbb{f}, X, \bar{a} \rangle$ otherwise. We will do so by creating a parity game based on $\mathfrak{A}, \theta$ and $\mathcal{E}$ (similar to the parity game encoding of LFP formulas in [Grä02]), restricting that parity game using a winning strategy for one of the players, and then extracting a suitable proof graph from the restricted parity game. This proof strategy has the practical implication that the extraction mechanism can be implemented on top of existing tools if the parity game was finite. Because our completeness proof uses the fact that winning strategies in parity games and proof graphs encode similar information, proving completeness and implementing an algorithm that generates proof graphs have both become relatively straightforward.

We will be using a slightly different definition of parity game than the one given in Chapter 5: we no longer require that the game graph is finite, nor do we require that the edge relation is total. Plays are then defined as maximal paths, and are no longer always infinite. We do however still require that the amount of different priorities occurring in the game is finite. The winner of a finite play is the opponent of the owner of the last vertex in the play (the player who gets stuck, loses). For infinite plays, the winner is defined as before. Our completeness proof relies only on the fact that such parity games are determined (that is, for every node in a parity game there is a winning strategy for one of the players from that node). This still holds for this adapted notion of parity game [Zie98; AN01].

Before we define our parity game encoding, we first define an auxiliary function $\Omega \colon \mathcal{X} \times \mathbb{B} \to \mathbb{N}$ that will be used to assign priorities to some of the nodes in our parity game.

$$\Omega(X, \alpha) = \begin{cases} 2 \cdot |\{Y \in \mathsf{bnd}^*(\mathcal{E}) \mid Y <_{\mathcal{E}} X\}|, & \alpha = \mathbb{t} \iff \sigma_X = \mathbf{gfp} \\ 2 \cdot |\{Y \in \mathsf{bnd}^*(\mathcal{E}) \mid Y <_{\mathcal{E}} X\}| + 1, & \text{otherwise} \end{cases}$$

Some nodes in the parity game will not correspond to any node in the proof graph that we will extract from it. These nodes do not have a counterpart in the proof graph

φ	$\mathsf{next}(v)$	$\Omega(v)$	$\Pi(v)$
$[X\bar{t} : \mathcal{E}]$	$\{\langle \alpha, \eta[\bar{x}_X \mapsto \bar{t}^{\mathfrak{A},\eta}], \mathsf{fo}(\varphi_X)\rangle\}$	$\Omega(X,\alpha)$	$\Diamond$, $X \in \mathsf{bnd}^*(\mathcal{E})$
	$\emptyset$	0	$\Box$, $X \notin \mathsf{bnd}^*(\mathcal{E}) \wedge \mathfrak{A}, \eta^{\alpha} \models \varphi$
	$\emptyset$	0	$\Diamond$, $X \notin \mathsf{bnd}^*(\mathcal{E}) \wedge \mathfrak{A}, \eta^{\alpha} \not\models \varphi$
$R\bar{t}$	$\emptyset$	0	$\Box$, $\mathfrak{A}, \eta^{\alpha} \models \varphi$ $\Diamond$, $\mathfrak{A}, \eta^{\alpha} \not\models \varphi$
$\neg\psi$	$\{\langle \neg\alpha, \eta, \psi\rangle\}$	$\Omega^{\max}$	$\Diamond$
$\psi_1 \vee \psi_2$	$\{\langle \alpha, \eta, \psi_1\rangle, \langle \alpha, \eta, \psi_2\rangle\}$	$\Omega^{\max}$	$\Diamond$, $\alpha = \mathbb{t}$ $\Box$, $\alpha = \mathbb{f}$
$\exists_x\ \psi$	$\{\langle \alpha, \eta[x \mapsto a], \psi\rangle \mid a \in A\}$	$\Omega^{\max}$	$\Diamond$, $\alpha = \mathbb{t}$ $\Box$, $\alpha = \mathbb{f}$

Table 7.1: Definitions of $\mathsf{next}(v)$, $\Omega(v)$ and $\Pi(v)$ for $v = \langle \alpha, \eta, \varphi\rangle$ in the context of $\mathfrak{A}$ and $\mathcal{E}$, by case distinction on φ.

because they represent some fine-grained information about the first-order logic operators in $\mathcal{E}$, that do not affect the reasoning about the fixpoint operators in $\mathcal{E}$. We will make these nodes the least relevant vertices of the parity game by assigning them the highest priority in the game, which we will call $\Omega^{\max}$. The parity of $\Omega^{\max}$ is irrelevant, because vertices with this priority represent proofs of first-order logic formulas, which do not contain loops (so whenever $\Omega^{\max}$ occurs infinitely often along a play, it is not the most significant priority occurring infinitely often, and therefore does not influence the winner of the play). We define $\Omega^{\max}$ as follows.

$$\Omega^{\max} = \max\{\Omega(X,\alpha) \mid X \in \mathsf{bnd}^*(\mathcal{E}) \wedge \alpha \in \mathbb{B}\} + 1$$

We are now ready to define our parity game $G = \langle V, \to, \Omega, \Pi\rangle$, given $\mathfrak{A}, \theta, \mathcal{E}, \bar{a} \in A^*$, $X \in \mathcal{X}$ and $\alpha \in \mathbb{B}$. Our objective is to construct a parity game that allows us to extract a proof graph from it that contains either the node $\langle \alpha, X, \bar{a}\rangle$, or $\langle \neg\alpha, X, \bar{a}\rangle$.

We define Ω, Π and a function next as specified in Table 7.1. Using the next function, V and $\to$ are then inductively defined as V^ω and $\to^\omega$ in:

$$V^0 = \{\langle \alpha, \theta[\bar{x} \mapsto \bar{a}], [X\bar{x}_X : \varepsilon]\rangle\} \qquad \to^0 = \emptyset$$
$$V^{i+1} = V^i \cup \bigcup_{v \in V} \mathsf{next}(v) \qquad \to^{i+1} = \to^i \cup \bigcup_{v \in V} \{\langle v, v'\rangle \mid v' \in \mathsf{next}(v)\}$$

Now we have defined a parity game, we aim to use the fact that parity games are determined to help us create a proof graph. The memoryless determinacy of parity games ensures that for the parity game we just created, we can always find a strategy—either for player $\Diamond$ or for player $\Box$—that wins from the node in V^0. Our hypothesis will be that if such a strategy can be found for player $\Diamond$, it must be the case that $\mathfrak{A}, \mathbf{S}(\theta, \mathcal{E}) \models \langle \alpha, X, \bar{a}\rangle$, and if such a strategy exists for player $\Box$, then $\mathfrak{A}, \mathbf{S}(\theta, \mathcal{E}) \models \langle \neg\alpha, X, \bar{a}\rangle$. So

depending on the player that wins the node in V^0, we extract a proof graph that contains either $\langle \alpha, X, \bar{a} \rangle$ or $\langle \neg\alpha, X, \bar{a} \rangle$. The soundness result from the previous section will then confirm our hypothesis.

We first establish that if two vertices differ only in the value of α, then they are won by different players.

Lemma 7.7. *If $\langle \alpha, \eta, \varphi \rangle$ and $\langle \neg\alpha, \eta, \varphi \rangle$ are both vertices in G, then they are not won by the same player.*

Proof. Given a vertex u, let $\neg u$ denote u with its α value negated, *e.g.*, $\neg\langle \beta, \eta', \chi \rangle = \langle \neg\beta, \eta', \chi \rangle$. Note that $\neg u^\bullet = \{\neg v \mid v \in u^\bullet\}$. Furthermore, if in G a vertex u has more than one successor, then it is of the form $\langle \beta, \eta', \chi \vee \psi \rangle$ or of the form $\langle \beta, \eta', \exists_x \chi \rangle$; in both cases $\Pi(u) \neq \Pi(\neg u)$. For any strategy $s \in \mathbb{S}_i$, we can therefore define a strategy $\bar{s} \in \mathbb{S}_{\neg i}$ as follows:

$$\bar{s}(u) = \begin{cases} u' & \text{if } u^\bullet = \{u'\} \\ \neg s(\neg u) & \text{otherwise} \end{cases}$$

Suppose $s \in \mathbb{S}_i$ is a winning strategy from $\langle \alpha, \eta, \varphi \rangle$ for player i. We will show that $\bar{s}$ is winning for $\neg i$. Consider any play p starting from $\langle \neg\alpha, \eta, \varphi \rangle$ that is allowed by $\bar{s}$. There must be a strategy $s' \in \mathbb{S}_i$ such that p is the only play allowed by $\bar{s}$ and s'. Now consider the unique play q from $\langle \alpha, \eta, \varphi \rangle$ that is allowed by s and $\bar{s'}$. This play must be won by player i, because s was winning for i. A straightforward induction shows that for all $k > 0$, $p_k = \neg q_k$.

If p is finite, then so is q, and the last vertex of p is owned by a different player than the last vertex of q. Hence if p is won by i, then q is won by $\neg i$. If p is infinite, then so is q, and there must be an infinite number of indices k such that $\Omega(p_k) < \Omega^{\max}$ (otherwise, we would have an infinite sequence of $\neg$-, $\vee$- and $\exists$ -vertices, which could never have been generated by the rules in Table 7.1). Because $p_k = \neg q_k$ for all these k, if $\Omega(p_k) = \Omega(X, \alpha)$, then $\Omega(q_k) = \Omega(X, \neg\alpha)$. The definition of $\Omega(X, \alpha)$ ensures that $\Omega(X, \alpha) < \Omega(Y, \beta)$ if $X <_\mathcal{E} Y$. Therefore, if $\Omega(X, \alpha)$ is the lowest infinitely often occurring priority on p, $\Omega(X, \neg\alpha)$ is the lowest infinitely often occurring priority on q. We know that the parity of $\Omega(X, \alpha)$ makes that p is won by i. Because the parity of $\Omega(X, \neg\alpha)$ is the opposite of that of $\Omega(X, \alpha)$, q is then won by $\neg i$. □

To allow a wider range of proof graphs to be extracted, we will not use regular strategies, but a nondeterministic variation on memoryless strategies.

Definition 7.5. *A* nondeterministic memoryless strategy *for player i is a partial mapping $s : V \to 2^V$ that is defined such that $s(v) \subseteq v^\bullet$ for all $v \in V$ such that $\Pi(v) = i$.*

Note that any memoryless strategy can be seen as a nondeterministic memoryless strategy that only maps nodes onto singleton sets. Therefore, for any node in a parity game, we can always find a nondeterministic strategy for one of the players that wins from that node.

We will generate a proof graph from a parity game G and a winning strategy s. Depending on the player for whom s is a strategy, the generated proof graph is either

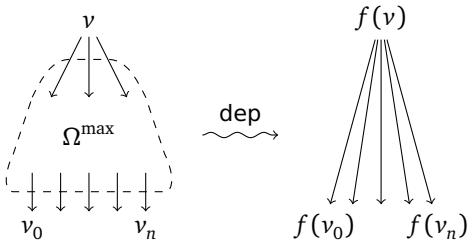


Figure 7.1: The dep transformation ignores all nodes with priority $\Omega^{\max}$.

a proof of validity, or a proof of invalidity of the encoded fixpoint formula. This is reflected in the α-values in the proof graph nodes. The partial mappings $f^+, f^-: V \to \mathcal{S}$ map nodes of the parity game to nodes of the proof graph we are creating; f^+ is used if a proof graph is generated from a winning strategy for player $\Diamond$, for player $\Box$, f^- is used.

$$f^+(v) = \left\{ \begin{array}{ll} \langle \alpha, X, \bar{t}^{\mathfrak{A},\theta} \rangle, & v = \langle \alpha, \theta, [X\bar{t} : \varepsilon] \rangle, \\ \langle \alpha, R, \bar{t}^{\mathfrak{A},\theta} \rangle, & v = \langle \alpha, \theta, R\bar{t} \rangle, \\ \text{undefined}, & \text{otherwise}, \end{array} \begin{array}{ll} \langle \neg\alpha, X, \bar{t}^{\mathfrak{A},\theta} \rangle \\ \langle \neg\alpha, R, \bar{t}^{\mathfrak{A},\theta} \rangle \\ \text{undefined} \end{array} \right\} = f^-(v)$$

We now define a mapping dep that transforms a parity game into a graph structure with nodes from $\mathcal{S}$. Define $V^- = \{v \in V \mid \Omega(v) < \Omega^{\max}\}$ and define for a nondeterministic strategy s for player i and for $f \in \{f^+, f^-\}$ that $\mathsf{dep}(G, s, f) = \langle S, \to \rangle$, in which:

$$\begin{aligned} S &= \{f(v) \mid v \in V^- \wedge i \text{ wins from } v \text{ with } s\}, \\ \to &= \{\langle f(v), f(v') \rangle \mid v, v' \in V^- \wedge \exists_{V' \subseteq V^-}\ v' \in V' \wedge v \xmapsto{s, V \setminus V^-} V'\}. \end{aligned}$$

In the above, $v \xmapsto{s, V \setminus V^-} V'$ is defined as in Section 5.1, meaning that the plays from v allowed by s can only reach nodes from V' after doing zero or more steps through nodes of priority $\Omega^{\max}$. Effectively, we are filtering out all such nodes, and applying the transformation given by f to the remaining nodes. If v could reach v' with s via a number of nodes with priority $\Omega^{\max}$, then $f(v)$ has an edge to $f(v')$ in the resulting graph. This transformation is schematically depicted in Figure 7.1.

Our claim now is that the graph generated by dep is in fact a proof graph. Note that this is all we need, because if v_0 is the node in V^0, the generated graph will always include $f^+(v_0)$ or $f^-(v_0)$, which by definition are equal to $\langle \alpha, X, \bar{a} \rangle$ and $\langle \neg\alpha, X, \bar{a} \rangle$, respectively.

Lemma 7.8. *If s is a winning nondeterministic strategy for $\Diamond$ in G, then $\mathsf{dep}(G, s, f^+)$ is a proof graph. If s is a winning nondeterministic strategy for $\Box$ in G, then $\mathsf{dep}(G, s, f^-)$ is a proof graph.*

Proof. Let s be a winning nondeterministic strategy for $\Diamond$ in G (the proof for the case that s is winning for $\Box$ is dual). Let $\mathsf{dep}(G, s, f^+) = \langle S, \to \rangle$. We first show that $\langle S, \to \rangle$ is a dependency graph. Let $v = \langle \alpha, X, \bar{a} \rangle \in S$, and distinguish two cases.

Case $X \in \mathsf{bnd}^*(\mathcal{E})$. From Table 7.1 we can see that $\Pi(v) = \Diamond$, and from the definition of dep we know that $v = f(u)$ for some $u = \langle \alpha, \theta, \varphi \rangle$. Again look at the table and notice that u has only one successor: $\langle \alpha, \theta[\bar{x}_X \mapsto \bar{a}], \mathsf{fo}(\varphi_X) \rangle$. Now define a sequence of sets as follows:

$$V_u^0 = \{\langle \alpha, \theta[\bar{x}_X \mapsto \bar{a}], \varphi_X \rangle\}$$
$$V_u^{i+1} = \{u'' \in V \mid u'' \in V_u^i \setminus V^- \vee \exists_{u' \in V_u^i \cap V^-}\, u' \xrightarrow{s} u''\}$$

Let $\mathfrak{A} \restriction S \sqsubseteq \mathfrak{B} \sqsubseteq \mathfrak{A}$, and $\eta \equiv_{\mathsf{fv}(\mathcal{E})} \theta$. We will prove by induction that for all i,

$$\eta' \equiv_{\mathsf{fv}(\mathcal{E})} \eta \text{ for all } \langle \beta, \eta', \psi \rangle \in V_u^i, \text{ and}$$
$$\text{if } \mathfrak{B}, \eta'^{\,\beta} \models \psi \text{ for all } \langle \beta, \eta', \psi \rangle \in V_u^i, \text{ then } \mathfrak{B}, \eta[\bar{x}_X \mapsto \bar{a}]^{\alpha} \models \mathsf{fo}(\varphi_X).$$

Case $i = 0$, $\theta[\bar{x}_X \mapsto \bar{a}] \equiv_{\mathsf{fv}(\mathcal{E})} \theta$ because $\mathsf{fv}(\mathcal{E}) \cap \mathcal{V} = \emptyset$, and Lemma 6.3 gives us

$$\mathfrak{B}, \theta[\bar{x}_X \mapsto \bar{a}]^{\alpha} \models \mathsf{fo}(\varphi_X) \Rightarrow \mathfrak{B}, \eta[\bar{x}_X \mapsto \bar{a}]^{\alpha} \models \mathsf{fo}(\varphi_X).$$

Case $i = j + 1$ for some j, assume as the induction hypothesis that the statements hold for j.

By the definition of V^i we know that for every $u'' = \langle \gamma, \eta'' \psi' \rangle \in V^i$ there is a $u' = \langle \beta, \eta', \psi \rangle \in V^j$ such that $u' \xrightarrow{s} u''$. It is easily seen from Table 7.1 that for such u and u', $\eta'' \equiv_{\mathsf{fv}(\mathcal{E})} \eta'$, and because $\eta' \equiv_{\mathsf{fv}(\mathcal{E})} \eta$ by the induction hypothesis, also $\eta'' \equiv_{\mathsf{fv}(\mathcal{E})} \eta$.

To prove the second statement, assume that $\mathfrak{B}, \eta'^{\,\beta} \models \psi$ for all $\langle \beta, \eta', \psi \rangle \in V_u^i$. We must show that $\mathfrak{B}, \eta[\bar{x}_X \mapsto \bar{a}]^{\alpha} \models \mathsf{fo}(\varphi_X)$. This can be obtained from the induction hypothesis if we can prove $\mathfrak{B}, \eta'^{\,\beta} \models \psi$ for all $\langle \beta, \eta', \psi \rangle \in V_u^j$. This is seen easily by fixing $w = \langle \beta, \eta', \psi \rangle \in V_u^j$ and distinguishing cases on the shape of ψ. Then establish that the required successors (either one successor or all successors, depending on the player of w) listed in Table 7.1 are all in V_u^i, and use our assumption about the vertices in V_u^i to conclude that $\mathfrak{B}, \eta'^{\,\beta} \models \psi$.

The successors of v in $\langle S, \to \rangle$ are collected in the set $\{f(u') \mid u' \in V_u^k\}$ for some k. We have shown that for all $\mathfrak{A} \restriction S \sqsubseteq \mathfrak{B} \sqsubseteq \mathfrak{A}$ and $\eta \equiv_{\mathsf{fv}(\mathcal{E})} \theta$ that

$$\text{if } \mathfrak{B}, \eta'^{\,\beta} \models \psi \text{ for all } \langle \beta, \eta', \psi \rangle \in V_u^k, \text{ then } \mathfrak{B}, \eta[\bar{x}_X \mapsto \bar{a}]^{\alpha} \models \mathsf{fo}(\varphi_X).$$

This is equivalent to the requirement that Definition 7.1 poses for X:

$$\text{if } \mathfrak{B}, \eta \models \nu' \text{ for all } \nu' \in \nu^{\bullet}, \text{ then } \mathfrak{B}, \eta[\bar{x}_X \mapsto \bar{a}]^{\alpha} \models \mathsf{fo}(\varphi_X).$$

This can be seen as follows: take any $\nu' \in \nu^{\bullet}$. Then $\nu' = f(u')$ for some $u' \in V_u^k$. Suppose that $u' = \langle \beta, \eta', [X\bar{t} : \varepsilon] \rangle$, then $\nu' = \langle \beta, X, \bar{t}^{\mathfrak{A}, \eta'} \rangle$. So $\mathfrak{B}, \eta \models \nu'$ is equivalent to $\bar{t}^{\mathfrak{A}, \eta'} \in \eta(X) \iff \beta = \mathbb{t}$, which is equivalent to $\bar{t}^{\mathfrak{B}, \eta'} \in \eta'(X) \iff \beta = \mathbb{t}$ (because $\mathfrak{B} \sqsubseteq \mathfrak{A}$ and $\eta' \equiv_{\mathcal{X}} \eta$, which can be seen from Table 7.1), which

is in turn equivalent to $\mathfrak{B}, \eta'^{\beta} \models [X\bar{t} : \varepsilon]$. A similar reasoning can be held for the case that $u' = \langle \beta, \theta', R\bar{t} \rangle$.

We must still show that the set of successors of v does not violate the requirement that $\to$ should be consistent. In other words, we must show that no two vertices in $v^\bullet$ are conflicting. This can be seen as follows. Suppose towards a contradiction that there are two conflicting nodes $f(w)$ and $f(w')$ in $v^\bullet$. By Lemma 7.7, either w is won by $\Box$, or w' is won by $\Box$. By definition of $v^\bullet$, there are paths from u to w and from u to w' that are allowed by s. But this means s cannot be winning for $\Diamond$.

Case $X \notin \mathsf{bnd}^*(\mathcal{E})$. From Table 7.1 we can see that v has no successors, so $\Pi(v) = \Box$, because $\Pi(s) = \Diamond$ wins from v. From the definition of dep we also know that $v = f^+(\langle \alpha, \theta, \varphi \rangle)$ for some node such that $\mathfrak{A}, \theta^{\alpha} \models \varphi$ (this can be seen from the table again), which is equivalent to $\bar{a} \in X^{\mathfrak{A},\theta}$.

This completes our argument that $\langle S, \to \rangle$ is a dependency graph. To show that it is a proof graph, consider an infinite path p in the dependency graph traversing some set of nodes S'. All nodes on such a path have successors, so by the definition of dep and V, there must be a corresponding path q in G going through parity game nodes V' such that $S' = \{f(v) \mid u \in V' \text{ and } f \text{ is defined on } v\}$. Because this path is allowed by s, it is a winning path. Therefore, the least n such that $\Omega(q_k) = n$ for an infinite number of values for k, is even.

Note that every $u \in V'$ is of the form $\langle \alpha, \theta, [X\bar{t} : \mathcal{E}'] \rangle$, and that for such u, $\Omega(u) = \Omega(X, \alpha)$. So $n < \Omega^{\max}$, because $q_k \in V'$ for an infinite number of values for k, and therefore $n = \Omega(X, \alpha)$ for some X and α. Note that Ω is defined such that $\Omega(Y, \beta) < \Omega(Z, \gamma)$ implies $Y \leq_{\mathcal{E}} Z$. Therefore, X is the smallest variable occurring infinitely often on p, and there are infinitely many occurrences of a node $\langle \alpha, X, \psi \rangle$ (for some ψ) on p. Because $\Omega(X, \alpha)$ is even, we know that $\sigma_X = \mathbf{gfp} \iff \alpha = \mathbb{t}$. □

7.2 Evidence for EFL

Fixpoint logics are capable of encoding a diverse set of decision problems. We would however like to have a notion of evidence that is independent of the expressed problems. This notion should allow us to extract more specialised diagnostics (in particular: diagnostics that have already been defined in literature) for various types of decision problem that can be encoded in EFL. We propose the following definition of evidence for EFL formulas, which formalises that evidence allows for reconstructing proof that a formula is true.

Definition 7.6. *Given* $\mathfrak{A}, \theta, \varphi$ *and* α, evidence *for* $\mathfrak{A}, \theta^{\alpha} \models \varphi$ *is a substructure* $\mathfrak{B} \sqsubseteq \mathfrak{A}$ *such that there is a proof graph for* $\mathfrak{B}, \theta^{\alpha} \models \varphi$ *that is also a proof graph for* $\mathfrak{A}, \theta^{\alpha} \models \varphi$.

Intuitively, we define evidence for a formula φ (not) holding on a structure $\mathfrak{A}$ as a substructure of $\mathfrak{A}$ on which φ can be proven (not) to hold *using exactly the same proof*. To obtain the smallest such substructure, we simply construct a substructure in which at least those facts hold that were used in the proof.

Definition 7.7. *Given a proof graph $\langle S, \to \rangle$ for $\mathfrak{A}, \theta \models^{\alpha} \varphi$, define* $\mathsf{ev}(\langle S, \to \rangle)$ *as the smallest $\mathfrak{B} \sqsubseteq \mathfrak{A}$ such that for each $\langle \beta, X, \bar{a} \rangle \in S$ (where X is a variable or a relation name) we have $\bar{a} \in B^*$ and for each $v \in S \setminus S_{\mathcal{X}}$, we have $\mathfrak{B}, \theta \models v$.*

It is easy to see that such a substructure always exists, as $\mathfrak{A}$ itself is always a candidate. It is also quite straightforward to compute the smallest such substructure, as it must at least contain everything that occurs syntactically in φ or in the proof graph. Additional elements need only be inserted in the domain of the substructure if this is needed to make the interpretations of function symbols closed under the substructure.

Theorem 7.3. *If G is a proof graph for $\mathfrak{A}, \theta \models^{\alpha} \varphi$, then* $\mathsf{ev}(G)$ *is evidence for $\mathfrak{A}, \theta \models^{\alpha} \varphi$.*

Proof. If G is a dependency graph for $\mathsf{ev}(G), \theta$ and φ, then G is also a proof graph, as the extra requirement from Definition 7.2 obviously still holds.

To see that G is indeed a dependency graph for $\mathsf{ev}(G), \theta$ and φ, pick any node $v = \langle \alpha, R, \bar{a} \rangle$ in G. If $R \notin \mathsf{bnd}^*(\varphi)$, then we need to show that $\mathsf{ev}(G), \theta \models v$. This is given directly by Definition 7.7. If $R \in \mathsf{bnd}^*(\varphi)$, we need to show that for all $\mathfrak{B} \sqsubseteq \mathsf{ev}(G)$ and all $\eta \equiv_{\mathsf{bnd}^*(\varphi)} \theta$:

$$\begin{aligned} &\text{if } \ \mathfrak{B}, \eta \models u \text{ for all } u \in v^{\bullet} \\ &\text{then } \ \mathfrak{B}, \eta[\bar{x}_X \mapsto \bar{a}] \models \mathsf{fo}(\varphi_X) \end{aligned}$$

This is however very easily obtained: because G is a proof graph for $\mathfrak{A}, \theta$ and φ, we have the above for all $\mathfrak{B} \sqsubseteq \mathfrak{A}$ and all $\eta \equiv_{\mathsf{bnd}^*(\varphi)} \theta$. Because every substructure of $\mathsf{ev}(G)$ is also a substructure of $\mathfrak{A}$, the result follows immediately. □

7.2.1 Counterexamples and witnesses

Some problems that can be encoded in fixpoint logic consist of checking an 'implementation' against a 'specification'. For instance, if the behaviour of some system is described as a Kripke structure, and we want to establish correctness properties on that Kripke structure, then we may view it as an 'implementation' of sorts, which we could check against a set of CTL* formulas, the 'specifications'. We might also want to check if this Kripke structure refines another, more abstract Kripke structure. In this case, the 'specification' is not a formula, but another Kripke structure. We refer to such problems as *model checking* problems.

For problems that have this characteristic of a division into implementation and specification, we tend to think of the specification as being given and well-understood, whereas the implementation may contain mistakes that need to be clarified with diagnostics. Such diagnostics should highlight the parts of the implementation that cause a problem, but should not include details from the specification. To achieve this, we propose a general scheme that combines an implementation $\mathfrak{A}$ with a specification $\mathfrak{B}$ using an operator $\sqcup$, and that extracts the information from $\mathfrak{A}$ from evidence $\mathfrak{C}$ relating to the combined model using an operator $\sqcap$.

Definition 7.8. *A* model checking problem *is a tuple* $\langle \mathfrak{A}, \mathfrak{B}, \theta, \sqcup, \sqcap, \varphi \rangle$ *where* $\mathfrak{A}$ *and* $\mathfrak{B}$ *are models,* $\sqcup$ *and* $\sqcap$ *are binary functions that combine two structures into a single new structure, and* φ *is an EFL formula such that if* $\mathfrak{C}$ *is evidence for* $\mathfrak{A} \sqcup \mathfrak{B}, \theta \models^{\alpha} \varphi$*, then*

$$\mathfrak{C} \sqcap \mathfrak{A} \sqsubseteq \mathfrak{A} \quad \textit{and} \quad (\mathfrak{C} \sqcap \mathfrak{A}) \sqcup \mathfrak{B}, \theta \models^{\alpha} \varphi.$$

We call $\mathfrak{C} \sqcap \mathfrak{A}$ *a* witness *if* $\alpha = \mathbb{t}$*. We call it a* counterexample *if* $\alpha = \mathbb{f}$*.*

This scheme enables us to encode the semantics of a logic or equivalence in a single EFL formula. Usually, φ will be a closed formula, in which case the value of θ is irrelevant. In such cases, we will not explicitly mention θ, but assume that an arbitrary environment is given. In the following sections we will give an example of a formula that encodes stuttering equivalence checking, in which case $\mathfrak{A}$ and $\mathfrak{B}$ are Kripke structures, and an example of a formula that encodes ∃ECTL* model checking, in which case $\mathfrak{A}$ is a Kripke structure, and $\mathfrak{B}$ is a model that represents an ∃ECTL* formula. This approach differs from those in [Che+07; GM99], in which a different fixpoint formula is generated for every $\mathfrak{A}$ and $\mathfrak{B}$.

To apply our notions of witness and counterexample, we need to define the operations $\sqcup$ and $\sqcap$ from Definition 7.8. Essentially, the $\sqcup$ operator must merge two structures together, and the $\sqcap$ operator must be able to retrieve a substructure of one of the original structures again from the merged structure. Natural candidates to implement this operation on the domain of discourse of the two structures are the set union and set intersection operations. We do however also need to define what happens to the relations and functions in the structures. If the two structures that are to be merged define the same function f, but with different interpretations $\mathcal{I}_1(f)$ and $\mathcal{I}_2(f)$, then this poses a problem, because there is no natural merge operation for such interpretations. If the interpretations agree on the intersection of their respective domains however, then a natural merge operation is to take the interpretation that assigns to every input the same output as $\mathcal{I}_1(f)$ does if the input is in the domain of $\mathcal{I}_1(f)$, or the output of $\mathcal{I}_2(f)$ if the input is in the domain of $\mathcal{I}_2(f)$.

For pairs of models in which the interpretation of the function symbols are compatible in this way, we define union and intersection operators in the following definition. We then show that using these operators, witnesses and counterexamples can be extracted as per Definition 7.8.

Definition 7.9. *Two structures* $\mathfrak{A} = \langle A, \mathcal{R}_1, \mathcal{F}_1, \mathcal{I}_1 \rangle$ *and* $\mathfrak{B} = \langle B, \mathcal{R}_2, \mathcal{F}_2, \mathcal{I}_2 \rangle$ *are called* composable *if for all* $f \in \mathcal{F}_1 \cap \mathcal{F}_2$*,* $\mathcal{I}_1(f)$ *and* $\mathcal{I}_2(f)$ *agree on the intersection of their domains. For such composable models we define* $\mathfrak{A} \cup \mathfrak{B} = \langle C^{\cup}, \mathcal{R}^{\cup}, \mathcal{F}^{\cup}, \mathcal{I}^{\cup} \rangle$ *and* $\mathfrak{A} \cap \mathfrak{B} = \langle C^{\cap}, \mathcal{R}^{\cap}, \mathcal{F}^{\cap}, \mathcal{I}^{\cap} \rangle$ *in which:*

$$\begin{aligned} C^{\cup} &= A \cup B & \qquad C^{\cap} &= A \cap B \\ \mathcal{R}^{\cup} &= \mathcal{R}_1 \cup \mathcal{R}_2 & \qquad \mathcal{R}^{\cap} &= \mathcal{R}_1 \cap \mathcal{R}_2 \\ \mathcal{F}^{\cup} &= \mathcal{F}_1 \cup \mathcal{F}_2 & \qquad \mathcal{F}^{\cap} &= \mathcal{F}_1 \cap \mathcal{F}_2 \end{aligned}$$

The interpretation function $\mathcal{I}^{\cup}$ is defined such that:

$$\mathcal{I}^{\cup}(R) = \begin{cases} \mathcal{I}_1(R), & R \in \mathcal{R}_1 \setminus \mathcal{R}_2 \\ \mathcal{I}_2(R), & R \in \mathcal{R}_2 \setminus \mathcal{R}_1 \\ \mathcal{I}_1(R) \cup \mathcal{I}_2(R) & R \in \mathcal{R}_1 \cap \mathcal{R}_2 \end{cases} \begin{cases} \mathcal{I}_1(R), \\ \mathcal{I}_2(R), \\ \mathcal{I}_1(R) \cap \mathcal{I}_2(R) \end{cases} = \mathcal{I}^{\cap}(R)$$

$$\mathcal{I}^{\cup}(f) = \begin{cases} \mathcal{I}_1(f), & R \in \mathcal{F}_1 \setminus \mathcal{F}_2 \\ \mathcal{I}_2(f), & R \in \mathcal{F}_2 \setminus \mathcal{F}_1 \\ \mathcal{I}_1(f) \cup \mathcal{I}_2(f) & R \in \mathcal{F}_1 \cap \mathcal{F}_2 \end{cases} \begin{cases} \mathcal{I}_1(f), \\ \mathcal{I}_2(f), \\ \mathcal{I}_1(f) \restriction C^{\cap} \end{cases} = \mathcal{I}^{\cap}(f)$$

Here, $\mathcal{I}_1(f) \restriction C^{\cap}$ is the restriction of $\mathcal{I}_1(f)$ to $C^{\cap}$. The union $\mathcal{I}_1(f) \cup \mathcal{I}_2(f)$ is the function that agrees with $\mathcal{I}_1(f)$ and $\mathcal{I}_2(f)$ on their respective domains. Note that such a function exists because $\mathcal{I}_1(f)$ and $\mathcal{I}_2(f)$ agree on the intersection of their domains.

Theorem 7.4. *If $\mathfrak{A}$ and $\mathfrak{B}$ are composable, θ is an environment and φ is an EFL formula over $\mathfrak{A} \cup \mathfrak{B}$, then $\langle \mathfrak{A}, \mathfrak{B}, \theta, \cup, \cap, \varphi \rangle$ is a model checking problem.*

Proof. Suppose that $\mathfrak{C}$ is evidence for $\mathfrak{A} \cup \mathfrak{B}, \theta \models^{\alpha} \varphi$. It is trivial to check that $\mathfrak{C} \cap \mathfrak{A} \sqsubseteq \mathfrak{A}$. To see that $(\mathfrak{C} \cap \mathfrak{A}) \cup \mathfrak{B}, \theta \models^{\alpha} \varphi$, consider a proof graph $G = \langle S, \rightarrow \rangle$ that is a proof both for $\mathfrak{A} \cup \mathfrak{B}, \theta \models^{\alpha} \varphi$ and for $\mathfrak{C}, \theta \models^{\alpha} \varphi$, as per Definition 7.6. Note that $\mathfrak{C} \sqsubseteq (\mathfrak{C} \cap \mathfrak{A}) \cup \mathfrak{B} \sqsubseteq \mathfrak{A} \cup \mathfrak{B}$. Now check that G is also a proof for $(\mathfrak{C} \cap \mathfrak{A}) \cup \mathfrak{B}, \theta \models^{\alpha} \varphi$ by checking on each node the conditions from Definition 7.1 (the condition from Definition 7.2 still holds). The first condition is still valid because it is independent of the model. The second condition follows from the fact that $(\mathfrak{A} \cup \mathfrak{B}) \restriction S \sqsubseteq (\mathfrak{C} \cap \mathfrak{A}) \cup \mathfrak{B} \sqsubseteq \mathfrak{A} \cup \mathfrak{B}$ and G is proof for $\mathfrak{A} \cup \mathfrak{B}, \theta \models^{\alpha} \varphi$; the third from the fact that $\mathfrak{C} \sqsubseteq (\mathfrak{C} \cap \mathfrak{A}) \cup \mathfrak{B}$ and $\mathfrak{C}, \theta \models^{\alpha} \varphi$. □

7.2.2 Example: stuttering bisimulation

To illustrate the use of the $\cup$ and $\cap$ operators on models, and to illustrate how counterexamples can be extracted for an equivalence checking problem, we consider the problem of checking that two systems are stuttering bisimilar. We use Namjoshi's formulation of stuttering bisimulation [Nam97], because it already closely resembles our definition in EFL.

Definition 7.10. *Given a Kripke structure $\langle A, AP, \rightarrow, \ell \rangle$, a relation $X \subseteq A \times A$ is a* stuttering bisimulation *if and only if it is symmetric, and there exist a well-founded order $\langle W, \prec \rangle$ and some mapping $rank : A \times A \times A \rightarrow W$ such that for all s, t such that Xst:*

$$\ell(s) = \ell(t) \wedge \forall_u\, s \rightarrow u \Rightarrow ((Xut \wedge rank(u,u,t) \prec rank(s,s,t)) \vee \exists_v\, t \rightarrow v \wedge ((Xsv \wedge rank(u,s,v) \prec rank(u,s,t)) \vee Xuv)).$$

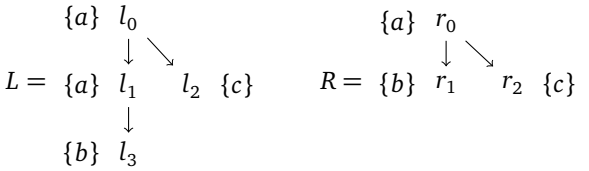
States s and t are said to be *stuttering bisimilar*, denoted $s \simeq t$, if a stuttering bisimulation exists that relates s and t.

Proposition 7.1. *Let $\mathfrak{A}$ be a Kripke structure $\langle A, AP, \to, \ell\rangle$.*

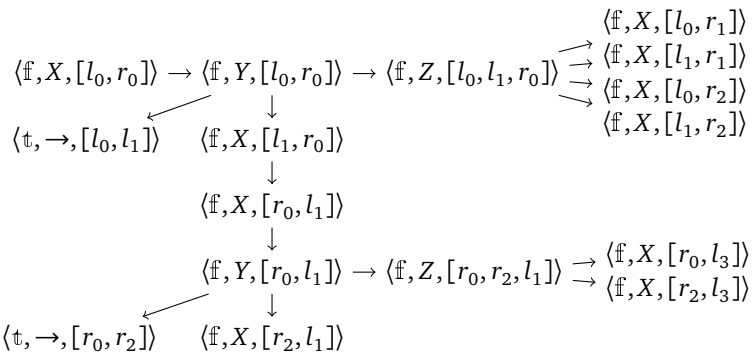
$$\begin{aligned}\Phi lr = \; & [Xlr : (\mathbf{gfp}\, Xst = Xts \wedge \ell(s) = \ell(t) \wedge \\ & [Yst : (\mathbf{lfp}\, Yst = \forall_u\, s \to u \Rightarrow ((Xut \wedge Yut) \vee \\ & [Zsut : (\mathbf{lfp}\, Zsut = \exists_v\, t \to v \wedge ((Xsv \wedge Zsuv) \vee Xuv))])])]\end{aligned}$$

If l and r are terms of $\mathfrak{A}$ and $s = l^{\mathfrak{A}}$ and $t = r^{\mathfrak{A}}$, then $\mathfrak{A} \models \Phi lr$ if and only if $s \simeq t$.

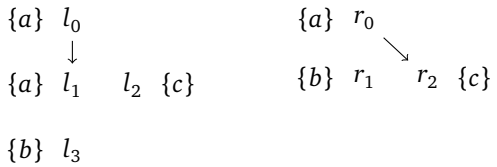
Consider the following two Kripke structures, that are stutter trace equivalent, but not stutter bisimulation equivalent.



Let $\mathfrak{A} = L \cup R$, and suppose that $l^L = l_0$ and $r^R = r_0$. Consider the following proof graph for $\mathfrak{A} \not\models \Phi lr$ (the nodes for $\ell(s) = \ell(t)$ are left out to save some space; note that technically speaking, the structures L and R should include the appropriate subsets of AP in their domains of discourse and define a relation symbol '=' that represents equality on those sets).



To extract evidence from this proof graph, we construct an evidence projection as per Definition 7.7. That is, we construct a submodel $\mathfrak{B} \sqsubseteq \mathfrak{A}$ which must contain at least those nodes from L and R referred to in the refutation graph (all nodes from L and R), and which satisfies $l_0 \to l_1$ and $r_0 \to r_2$. This yields the following Kripke structure $\mathfrak{B}$ as evidence. Note that $\mathfrak{B} \cap L$ and $\mathfrak{B} \cap R$ return the offending parts of L and R, respectively.



Observe that in $\mathfrak{B}$, l_0 and r_0 are again not stuttering equivalent, and moreover, they can be shown not to be equivalent with the same reasoning: the transition from l_0 to a state unrelated to r_0 with label a cannot be mimicked by r_0. All the states from $\mathfrak{A}$ are retained in the evidence, because the evidence gives an explanation for the invalidity of *every* X-node in the proof graph. Taking the projection of the refutation graph minimised with respect to $\mathfrak{B}$ would yield evidence in which only the reachable nodes from l_0 and r_0 are included.

Other refutation graphs are possible, leading to different evidence. For instance, if we had chosen $\langle \mathbb{f}, X, [l_0, r_0] \rangle$ to depend on $\langle \mathbb{f}, X, [r_0, l_0] \rangle$ (using the symmetry of stuttering bisimulation), we could have obtained evidence in which only the edge $r_0 \to r_1$ was retained. The explanation here is that it is sufficient to show that r_0 can reach an equivalence class labelled with b, without moving through another class first, whereas l_0 cannot do so.

We would like to remark that there are alternatives to our notion of evidence for bisimulation and stuttering bisimulation. For instance, a common notion is a distinguishing formula in Hennesy-Milner logic (for bisimulation [Cle91]) or CTL*\X (for stuttering bisimulation [Kor92]). However, in our experience, such formulas tend to get very unwieldy and do not always offer much insight. We believe that our notion of evidence is a more practical alternative to distinguishing formulas in such cases.

7.3 Evidence for LTL and ACTL* model checking

In [Cla+02], Clarke et al. noted that for certain model checking problems, one can restrict the notion of witness and counterexample to graphs of a specific form: for LTL model checking, counterexamples are usually defined as a single (possibly infinite) trace through the model that does not satisfy the specification. These traces can again be seen as Kripke structures that do not satisfy the desired property. For model checking ∀CTL*—a subset of CTL* which adds to LTL universal quantification over branches—counterexamples consist of a number of traces that are attached to each other in a tree-like fashion. More formally, a tree-like counterexample is a Kripke structure that can be simulated by the system under scrutiny, which does not satisfy the desired property, in which every strongly connected component (SCC) consists of a single cycle, and of which the SCC decomposition is a tree.

In the remainder of this section, we show how these special types of counterexample can be obtained from proof graphs.

To simplify presentation, we do not consider the generation of counterexamples for ∀CTL*, but rather the dual problem of generating witnesses for ∃CTL*. Furthermore, to also capture the expressivity of the ω-regular extensions used in [Cla+02], we consider the extended logic ∃ECTL* (originally presented in [Tho89], see Section 2.4 for our definition), which uses Büchi automata as primitives. We note that it is also possible to define what follows directly for ∃CTL*, but this requires encoding the translation of LTL to Büchi automata in first-order logic, as was done in Chapter 4.

An ∃ECTL* formula f can be described by a model $\mathfrak{B}_f$ over a domain that includes at least one element for every subformula and every set of subformulas of f, and

for each subformula of the form $\mathsf{E}(\mathcal{B})$ a unique element for every state of $\mathcal{B}$. We let $\mathfrak{B}_f$ contain an element representing AP and an element representing the set F of accepting Büchi states. We assume that it also includes the usual relations on sets, relations to recover the structure of formulas, and a ternary transition relation $\rightarrow$ for the Büchi automata. To distinguish CTL* operators from Boolean connectives, we add a dot to the CTL* operators: $\dot{\neg}$ for CTL* negation, and $\dot{\wedge}, \dot{\forall}$ for CTL* conjunction and disjunction.

Proposition 7.2. *Let Φ be defined as:*

$$\Phi sf = [Xsf : \mathbf{lfp}\, Xsf = \begin{cases} f \in \ell(s) & \text{if } f \in AP \\ g \notin \ell(s) & \text{if } f = \dot{\neg} g \\ Xsg \vee Xsh & \text{if } f = g \dot{\forall} h \\ Xsg \wedge Xsh & \text{if } f = g \dot{\wedge} h \\ Ysb_0 & \text{if } f = \mathsf{E}(\mathcal{B}(b_0)) \end{cases}$$

$$\mathbf{gfp}\, Ysb = Zsb$$

$$\mathbf{lfp}\, Zsb = \exists_{s',b',g}\ s \rightarrow s' \wedge b \xrightarrow{g} b' \wedge Xsg \wedge ((b' \in F \wedge Ys'b') \vee (b' \notin F \wedge Zs'b'))]$$

Let $\mathfrak{A}$ be a Kripke structure over AP, and let f be an $\exists$ECTL formula over AP. If s is a term of $\mathfrak{A}$ and f is a term of $\mathfrak{B}$, and $\hat{a} = s^{\mathfrak{A}}$ and $\hat{b} = f^{\mathfrak{B}_f}$, then $\mathfrak{A} \cup \mathfrak{B}_f \models \Phi sf$ if and only if $\mathfrak{A}, \hat{a} \models \hat{b}$.*

Let G be a minimal proof graph for $\mathfrak{A} \cup \mathfrak{B}_f \models \Phi sf$. Firstly, the first element of all nodes in G that are also in $\mathcal{S}_{\mathcal{X}}$ is equal to $\mathbb{t}$. Notice that G cannot contain cycles that pass through $\mathcal{S}_{\{X\}}$, because of Definition 7.2. Furthermore, because G is minimal, nodes from $\mathcal{S}_{\{Y,Z\}}$ have exactly one successor in $\mathcal{S}_{\{Y,Z\}}$. Therefore, the only cycles in G are cycles through $\mathcal{S}_{\{Y,Z\}}$, and every node can be in at most one cycle. In other words, every SCC in G consists of a single cycle.

Our goal is to obtain a tree-like witness from a proof graph for $\mathfrak{A} \cup \mathfrak{B}_f \models \Phi sf$, if state $\hat{a}$ satisfies $\exists$ECTL* formula $\hat{b}$. We do so by first finding a witness in which every SCC is again a single cycle. We can however not use the witness obtained from G by Definition 7.8, because disjunct cycles in G may correspond to cycles in $\mathfrak{A}$ that share nodes. We therefore need to 'unroll' some of these cycles in order to make them disjunct again. In [Cla+02] this is done by running a model checking algorithm not on $\mathfrak{A}$, but on a bisimilar *indexed Kripke structure* $\mathfrak{A}^\omega$, which contains for every cycle in $\mathfrak{A}$ an infinitely unrolled path. We adopt the same approach: we adapt G so that it becomes a proof graph for $\mathfrak{A}^\omega \cup \mathfrak{B}_f \models \Phi sf$, and then use Definition 7.8 to extract a witness from the resulting proof graph.

Definition 7.11. *Given a Kripke structure $\mathfrak{A} = \langle A, AP, \rightarrow, \ell \rangle$, its corresponding* indexed Kripke structure *$\mathfrak{A}^\omega$ is the Kripke structure $\langle A^\omega, AP, \rightarrow^\omega, \ell^\omega \rangle$ such that:*

- *$A^\omega = A \times \mathbb{N}$,*
- *$\rightarrow^\omega$ is such that $\langle a, i \rangle \rightarrow^\omega \langle a', j \rangle$ iff $a \rightarrow a'$ (for all a, a', i and j),*

– ℓ^ω *is such that* $\ell^\omega(\langle a, i\rangle) = \ell(a)$.

Note that every $a \in A$ is bisimilar to all $\langle a, i\rangle \in A^\omega$. Therefore, fixing some $i \in \mathbb{N}$ and replacing every $a \in A$ occurring as a parameter of a node in G by $\langle a, i\rangle$ yields a valid proof graph G^i (for $\mathfrak{A}^\omega \cup \mathfrak{B}_f$). For distinct i and j, the sets of nodes of G^i and G^j that have outgoing edges are disjunct, so $G^i \cup G^j$ is again a valid proof graph. By the same reasoning, so is $G^\omega = \bigcup_{i\in\mathbb{N}} G^i$. Associate with every node v of G a distinct number $k(v)$, and extend k to nodes of G^ω by defining for v in G and v' in G^ω that $k(v') = k(v)$ iff v' is equal to v in which every $a \in A$ is replaced by $\langle a, i\rangle$.

For every v in $G^\omega \cap \mathcal{S}_{\{Z\}}$, we replace every edge $v \to \langle \mathbb{t}, V, [\langle a, i\rangle, b]\rangle$ such that $V \in \{Y, Z\}$ and $i \neq k(v)$ by $v \to \langle \mathbb{t}, V, [\langle a, k(v)\rangle, b]\rangle$. Note that v also has a successor $\langle \mathbb{t}, \to, [\langle a', i\rangle, \langle a, i\rangle]\rangle$. Replace this successor by the node $\langle \mathbb{t}, \to, [\langle a', i\rangle, \langle a, k(v)\rangle]\rangle$. Let G^{t} be the result of this transformation, restricted to the part that is reachable from v_0.

G^{t} is a valid dependency graph again; the restriction to the reachable part from v_0 is easily seen to preserve the conditions of Definitions 7.1 and 7.2. In the replacements we made, only the first and last conjunct in the right-hand side of the equation for Z are affected by a different choice for s'. These two conjuncts are represented by the new successors we introduced, satisfying the constraint from Definition 7.1.

To see that G^{t} is also a proof graph, notice that no 'bad' cycles were introduced during the transformation: if we view proof graphs as Kripke structures in which two nodes are labelled identically if and only if they differ only in the index of a state in $\mathfrak{A}^\omega$ (*i.e.*, if they are of the form $\langle \mathbb{t}, V, [\langle a, i\rangle, b]\rangle$ and $\langle \mathbb{t}, V, [\langle a, j\rangle, b]\rangle$), then all identically labelled nodes in G^ω are bisimilar. Moreover, we only replaced edges $v \to u$ by $v \to u'$ such that u and u' are bisimilar.

We now define a witness $\mathfrak{C}$ as defined in Definition 7.8, *i.e.*, $\mathfrak{C} = \mathsf{ev}(G^{\mathrm{t}}) \cap \mathfrak{A}^\omega$. The following theorem establishes a correspondence between the transition relations of G^{t} and $\mathfrak{C}$. This allows us to say something about the shape of $\mathfrak{C}$, based on what we already know about G^{t}.

Theorem 7.5. *Let* $\mathfrak{C} = \mathsf{ev}(G^t) \cap \mathfrak{A}^\omega$*, and let* S *be the set of states of* G^t*. Then there is a bijective mapping* α *from* C *to* S *such that for all* $c, c' \in C$*,*

$$c \twoheadrightarrow c' \Rightarrow \alpha(c) \twoheadrightarrow \alpha(c'),$$

where $\twoheadrightarrow$ *denotes the transitive closure of* $\to$*.*

Proof. Let G be a proof graph for $\mathfrak{A} \cup \mathfrak{B}_f \models \Phi s f$ again. Let $v_0^G = \langle X, [\hat{a}, \hat{b}]\rangle$ and let $c_0 = \langle \hat{a}, k(v_0^G)\rangle$ be the corresponding node in $\mathfrak{C}$. Denote by v_0 the node $\langle X, [c_0, \hat{b}]\rangle$ in G^{t}. Now define:

$$S' = \{v_0\} \cup \{v \mid \exists_{v'}\ (v, v') \in \to \cap\, \mathcal{S}_{\{Z\}} \times \mathcal{S}_{\{Y,Z\}}\}.$$

Let α be a bijective mapping from C to S such that $\forall_{\langle \mathbb{t}, V, [c,b]\rangle \in S'}\ \alpha(c) = \langle \mathbb{t}, V, [c, b]\rangle$. Note that every $v = \langle \mathbb{t}, V, [c, b]\rangle \in S'$ has a predecessor $\langle \mathbb{t}, Z, [c, b]\rangle$, and because G^{t} is minimal, v has either $\langle \mathbb{t}, Y, [c, b]\rangle$ or $\langle \mathbb{t}, Z, [c, b]\rangle$ as a successor, but not both. Because $c = \langle a, k(v)\rangle$ for some a, and all nodes in G^{t} are reachable from v_0, $\langle \mathbb{t}, Y, [c, b]\rangle$ and $\langle \mathbb{t}, Z, [c, b']\rangle$ cannot both occur in S' (for all c, b and b').

Let $c, c' \in C$. We show that $c \twoheadrightarrow c' \iff \alpha(c) \twoheadrightarrow \alpha(c')$.

We first show $c \twoheadrightarrow c' \Rightarrow \alpha(c) \twoheadrightarrow \alpha(c')$ by proving $c \to c' \Rightarrow \alpha(c) \twoheadrightarrow \alpha(c')$. Suppose $c \to c'$. Consider the node $\langle \mathbb{t}, \to, [c, c'] \rangle \in S$ that caused us to add this transition to $\mathfrak{C}$. This node must have a predecessor $u = \langle \mathbb{t}, Z, [c, b] \rangle$, and per construction, u must have $\alpha(c')$ as a successor. We know that u is reachable from v_0, so if $\alpha(c) = v_0$, then definitely $\alpha(c) \twoheadrightarrow \alpha(c')$. Otherwise, note that $c = \langle a, i \rangle$, and $i \neq 0$. Any node that refers to a node in $\mathfrak{A}^\omega$ with index i, so in particular u, can only be reached by passing through a transition from a node in $\mathcal{S}_{\{Z\}}$ to a node $u' \in \mathcal{S}_{\{Y,Z\}}$ with $k(u') = i$. By construction, $\alpha(c)$ is the only such node, so also in this case, $\alpha(c) \twoheadrightarrow \alpha(c')$.

For the implication in the other direction, suppose $\alpha(c) \twoheadrightarrow \alpha(c')$. Consider the path from $\alpha(c)$ to $\alpha(c')$, and let $N \geq 2$ be the number of times this path traverses a node from S'. Along every such path, for each pair of successive nodes $\langle \mathbb{t}, V, [d, b] \rangle \to \langle \mathbb{t}, V', [d', b'] \rangle$ we have that $d = d'$ unless $\alpha(d') = \langle \mathbb{t}, V', [d', b'] \rangle$ (the only place where d can change to d' is in the right-hand side of Z, and these are the dependencies we have included in our mapping α).

Note that because of the way we constructed G^{t}, $\alpha(c')$ must have a predecessor $\langle \mathbb{t}, Z, [c'', b] \rangle$ that has a successor $\langle \mathbb{t}, \to, [c'', c'] \rangle$. By Definition 7.7 therefore $c'' \to c'$.

We prove that $c \twoheadrightarrow c'$ by induction on N. If $N = 2$, then $c = c''$, and therefore $c \to c'$. If $N > 2$, then notice that $\alpha(c')$ can only be reached from $\alpha(c)$ via $\alpha(c'')$. The induction hypothesis is that $c \twoheadrightarrow c''$, and we also have $c'' \to c'$; therefore also $c \twoheadrightarrow c'$. □

We already noted that in G, and therefore in G^{t}, every SCC consists of a single cycle. Using Theorem 7.5, we may conclude that all nodes in $\mathfrak{C}$ are reachable (because all nodes in G^{t} are), and that also every SCC in $\mathfrak{C}$ is a single cycle. Therefore, $\mathfrak{C}$ can be transformed to a bisimilar, tree-like model $\mathfrak{C}^{\mathrm{t}}$ by duplicating SCCs with more than one incoming transition. Moreover, $\mathfrak{A}$ simulates $\mathfrak{C}^{\mathrm{t}}$ because $\mathfrak{A}$ is bisimilar to $\mathfrak{A}^\omega$, $\mathfrak{C} \sqsubseteq \mathfrak{A}^\omega$, and $\mathfrak{C}$ is again bisimilar to $\mathfrak{C}^{\mathrm{t}}$. The fact that f holds on $\mathfrak{C}$ follows directly from Proposition 7.2, so we may conclude that it also holds on the bisimilar $\mathfrak{C}^{\mathrm{t}}$.

Corollary 7.1. *$\mathfrak{C}^{\mathrm{t}}$ is a tree-like witness.*

In case of ∃ELTL model checking, there can only be one cycle in G, corresponding to the single Büchi automaton in the formula. The unfolding operation described above is then unnecessary.

Corollary 7.2. *If f was an ∃ELTL formula, then $\mathfrak{C}$ is a linear witness.*

7.4 Closing remarks

We have presented the notion of proof graph for the fixpoint logic EFL. This notion can be used directly for the LFP and PBES logics, because those are syntactic fragments of EFL. A proof graph represents a proof of validity or invalidity of an EFL formula. Proof graphs can therefore be used to define an alternative semantics for EFL: a formula holds if and only if there is a proof graph that demonstrates this, and likewise, a formula does not hold if and only if there is a proof graph that demonstrates it.

Proof graphs by themselves have a number of uses. Firstly, they can be used in proofs instead of the doubly recursive fixpoint semantics of Definition 6.1 that is normally used. In some cases, this can make proofs significantly easier (see, *e.g.*, the proofs of Lemmas 6.14 and 6.16).

The second use is generation of diagnostics. We have shown that counterexamples and witnesses that can be obtained from classical model checking algorithms, can also be extracted from proof graphs. Because proof graphs can be constructed from winning parity game strategies, this yields a practical diagnostics generation framework for fixpoint model checkers, as they often already calculate winning strategies in parity games to solve the fixpoint formula that encodes the model checking problem.

A third use is certification, as was suggested by Tan and Cleaveland in [TC02]. Although we have not discussed this purpose in this chapter, their results are easily seen to carry over to our setting. We must note however that this only yields a certification method for the propositional fragment of EFL in positive normal form (in which all second-order variables occur only in the scope of an even number of negations). Checking whether a graph G is a proof graph for a formula in this fragment can be decided in $O(|G| \log \frac{k}{2})$ time, where k is length of the longest $<_\varepsilon$-chain in the formula (it amounts to solving the even-cycle problem, see [Tan02]). For formulas outside this fragment, it is not very clear how successful proof graphs can be as a certification mechanism. Especially in the case of formulas with first-order quantification, it becomes harder to check the requirements from Definition 7.1. For arbitrary proof graphs for propositional EFL formulas, checking the first condition of Definition 7.1 for a node with no successors amounts to tautology checking $\mathsf{fo}(\varphi_X)$, which is already a co-NP complete problem. It is unclear to what extent this problem would manifest itself in practical situations.

Chapter 8

Discussion and conclusion

Fixpoint logic can be applied to solve model checking problems. In the first chapter of this thesis, a case study was presented in which part of an industrial standard was modelled in the process specification language mCRL2. Requirements that should be satisfied by the standard were formulated in a first-order modal μ-calculus. Those requirements were checked on the model by translating them, together with the model, into formulas of a fixpoint logic (a parameterized Boolean equation system, or PBES, in this case). These formulas were subsequently evaluated by generating a parity game from it, and solving the resulting parity game.

The case study confirmed the presence of an error in the standard reported by Steiner [Ste05b], and uncovered one previously unknown error in the standard. Some aspects of the procedure that was used to come to these results, make it difficult and time consuming (and therefore costly) to perform this kind of analysis. The modal μ-calculus that was used to formulate the requirements is difficult to understand, making it difficult to specify the requirements correctly. Evaluating the PBES is time consuming due to the large number of states that have to be checked. Moreover, if the outcome of evaluating the PBES is unexpected, it is very difficult to find out what the cause is. In the remainder of the thesis, we have looked into these three issues, and made steps towards improving these aspects of our model checking procedure.

The μ-calculus that was used to describe properties in the case study of Chapter 3 appears rather arcane to the untrained eye, and requires a fair amount of training before one develops enough intuition to easily write down properties of interest. Chapter 4 discusses a translation from the more user-friendly (but less expressive) temporal logic CTL* to a propositional variant of the modal μ-calculus that was used in the first chapter. We showed that model checking the resulting FO-L_μ formula is no harder than checking the original CTL* formula. Firstly, this result gives some insight into the expressivity and compactness of FO-L_μ. Emerson and Lei suggested in 1986 [EL86] that there is no need for specialized CTL* model checkers, if CTL* can be efficiently encoded into L_μ. The best known translation from CTL* to L_μ however yields formulas of a size that is doubly expential in the size of the original [Dam92]. Bhat and Cleaveland gave a translation from CTL* to an equational variant of L_μ, yielding formulas that are only exponentially larger than their originals [BC96]. We gave a translation to a non-equational μ-calculus that allows first-order quantification, and showed that

the size of the resulting formula is linear in the size of the original.

The practical upshot of this result is that CTL* can be checked efficiently using a FO-L$_\mu$ model checker (a similar approach, using the translation by Bhat and Cleaveland, was taken in the Concurrency Workbench of North Carolina [CS02]). In particular, for the mCRL2 toolkit this means that there is an easy way to support more user friendly logics, without having to change the model checking infrastructure that is already there. A side note here is that the translation should be adapted to work for an action-based variant of CTL*, as mCRL2 uses labelled transition systems as it semantic model. One could for instance use the logic that was proposed by De Nicola and Vaandrager [DV90].

An issue for any model checker is that it has to deal with very large structures, resulting in excessive time and resource usage when model checking industrial scale systems. Much of the research in the model checking community is towards finding ways to avoid this problem. A well-known approach is to find an equivalence on states of the system that preserves the property of interest, and to then use the quotient—induced by this equivalence—of the system model to check the property on. Coarser equivalences give better speedups when using this method, as their quotient is smaller.

In Chapter 5, we applied this technique not to system models, but to parity games that encode a model checking problem. Parity games are graph structures in which every vertex is 'won' by one of two players, according to a fixed set of rules. The solution to a model checking problem can be inferred from the winner of a specific vertex in the parity game that encodes that problem. Instead of calculating the winner of that vertex directly, one can calculate the winner of the vertex in the quotient of the game (with respect to some equivalence relation) that represents the corresponding equivalence class in the original game. As long as the equivalence that is used does not relate two vertices in the parity game that are won by different players, the winner of the vertex in the quotient indeed also wins the vertex in the original game. We do not need to worry—as we do when performing classical state space reduction—that certain properties are not preserved by the quotient: as long as an equivalence relation does not relate vertices that are won by different players, this reduction technique is sound.

Several equivalences, the coarsest of which is called *governed stuttering bisimilarity*, were investigated. Earlier results indicated that minimizing a parity game using these equivalences and then solving it could be faster than solving the parity game directly. In the FlexRay case study, for instance, application of the parity game reduction technique saved a lot of time. Once an efficient implementation of the recursive solving algorithm for parity games became available, however, these results were deprecated. It turned out that the recursive solver was in most cases so fast on realistic model checking problems, that reducing the game before solving them only caused additional overhead.

We showed that most model checking problems result in parity games with only few, easily computable equivalence classes (where by 'easily' we mean: in $O(n^2m)$ time). If we can lift governed stuttering bisimilarity to the setting of fixpoint logic, then we have a way of reducing the parity game associated with a fixpoint logic formula prior to generating it. This would be good news indeed, as instantiating such formulas

to parity games is currently a major bottleneck: it costs a lot of time, and because during instantiation the algorithm must check whether a state was visited before, it also costs a lot of memory. Even if not the complete reductive power of governed stuttering equivalence can be lifted to fixpoint logic, the reductions that were seen in our experiments give us the hope that any heuristic that approximates it can still give rise to a considerable size reduction. Given that governed stuttering bisimilarity combines the notions of stuttering and idempotence, it seems that on the level of fixpoint logic, governed stuttering bisimilarty should correspond to a confluence-like notion; if a formula contains conjunctions and disjunctions in which second-order variables occur, then governed stuttering bisimilarity identifies those conjuncts and disjuncts that are seen to be equivalent after a number of fixpoint approximation steps. To find such conjuncts and disjuncts using static analysis, one might find inspiration in termination proving, especially in the context of proving termination of loop constructs in programming languages.

Another direction for future research could be to find equivalences that are even weaker than governed stuttering bisimilarity, but that are still computable in polynomial time. This would increase our understanding of the complexity of the problem of solving parity games, which is currently known to be in UP ∩ co-UP. Governed stuttering bisimilarity does not relate vertices with different priorities. There are equivalence relations on parity games that relate such vertices [FW06], but they are incomparable to governed stuttering bisimilarity [Kei13]. It would be very interesting to know if there is any equivalence that is coarser than both and computable in polynomial time, and that refines winner equivalence.

Model checking via fixpoint logic can be used to verify the behaviour of industrial scale systems. Over the years, many case studies have been performed, of which the FlexRay case study is just one example. What this method is still missing is a universally applicable way to provide diagnostics that allow the user to interpret the answers that the model checker provides. Such diagnostics are available for automata-based LTL and CTL* model checkers, but model checkers based on fixpoint logic can only provide them in very specific circumstances. In Chapters 6 and 7, we try to improve on this state of affairs. We introduce a new notion of *evidence*, based on a graph structure called *proof graph*, which is closely related to parity games. This notion is defined for a fixpoint logic called EFL, which generalises two popular first-order fixpoint logics, to which our results are therefore also immediately applicable. The notion of evidence for EFL can be used to define counterexamples and witnesses for formulas that encode a model checking problem. These counterexamples and witnesses are of the same shape as those defined for LTL and CTL* model checking. Our notions are general enough to extend to other problem domains, such as equivalence checking.

Our notion is based on *support sets*, as introduced by Tan and Cleaveland [Tan02], but is generalized to deal with first-order constructs and formulas that are not in normal form. In our generalized notion, the graph structure of proofs for fixpoint formulas is more explicitly exposed than it was in support sets. In that respect they are reminiscent of tableaux such as described by Janin [Jan97], and it would be interesting to investigate the exact relationship between the two.

While proof graphs are useful to give concise proofs for statements about fixpoint

logics, we have already come across one situation in which we desired a notion of proof graph that carries more information. In [Cra+ed], the *history-wise* proof graph is introduced; where regular proof graphs show a correspondence to winning *memoryless* strategies in parity games (as is detailed in Section 7.1.4), these proof graphs correspond to arbitrary winning strategies. It could be convenient to merge the two concepts, and view the proof graphs presented in this thesis as special instances of a more general notion, in analogy to the parity game strategies.

To put our notion of evidence to the test, the concepts from Chapter 7 should be implemented in an actual model checker. Such a model checker must be able to perform the extraction of counterexamples and witnesses, and must therefore be aware of the fact that the proof graphs it generates represent proofs in a mathematical structure that is in fact the composition of a structure that represents the implementation, and a structure that represents the specification (as described in Definition 7.8). This requires some bookkeeping that is unlikely to be present in current model checking tools, as they usually treat the fixpoint logic as an intermediate datastructure, rather than a central notion. It may therefore prove to be difficult to implement diagnostics generation in existing tools.

That said, a toolset that is centered around fixpoint logic could be very useful in practice. At the time of writing, there seem to be no model checking tools that have fixpoint logic as a central notion. Roughly speaking[1], one can distinguish tools that are built around a process algebra or specification language (CADP, mCRL2, SPIN), a specific model checking (or refinement checking) algorithm (FDR, (Nu)SMV, XMC), specialize in timed or probabilistic model checking (UPPAAL, PRISM), or aim to provide a uniform structure in which a wide range of different input languages and algorithms can be used (LTSmin, Edinburgh CWB, CWB-NC).

A set of tools that focuses on fixpoint logic, allowing symbolic manipulation and instantiation-based solving methods, could be a valuable addition. Existing techniques for the PBES logic are already powerful enough to solve practical verification problems, showing that such a toolset could be a serious candidate for large-scale use. The results from Chapter 5 indicate that symbolic manipulation has the potential to combat the state space explosion problem that inevitably arises when applying these tools in industry. Using the results from Chapter 7, diagnostics generation can be implemented in a generic manner. Furthermore, a set of tools based on fixpoint logic would operate in a setting that is quite similar to that of SAT/SMT solving and theorem proving. Hopefully, techniques from these areas can be more easily (and more widely) integrated into a fixpoint-based model checker than is possible in existing tools and methodologies.

It should be noted that the PBES tools developed over the last couple of years in the mCRL2 toolset are already a step in this direction. Yet due to its history as a process algebraic toolset, it is difficult to separate those parts concerned with fixpoint logic from parts dealing with other core concepts in the toolset, and to revert design decisions that were made when fixpoint logic was not yet a central notion.

We conclude with some remarks about the verification of industrial systems in general. Model checking research largely focuses on overcoming technical issues, such as

[1] There are many more features one could use to categorize these tools.

state space explosion and the answering of ever more difficult questions about systems (with a current trend towards probabilistic systems, timed systems, and the combination of the two). Every so often we also see usability aspects taken into account (see, *e.g.*, [Tan02; Don03]), but the approach is always to adapt or augment model checking techniques to make them more usable in current (software) engineering practice. A subject that deserves more attention, is which minimal changes the engineering disciplines could make to increase the effectiveness of current model checking techniques.

As an example, consider the FlexRay protocol specification. This specification is difficult to understand. While the 300 page document is very precise, it fails to explain the choices that were made. The level of detail of the specification is that of a reference implementation, and no precise high level descriptions are available of the components that each node is composed of. In many cases, variables and constants (such as the constraints on the variables) are left unexplained; the text accompanying the SDL diagrams is merely a rephrasing of the diagram itself. It would help if the specification were given at different levels of abstraction. A clear interface description of the different components, for instance, would have made the job of modelling a FlexRay node much easier.

A second issue is the monolithic nature of the specification. Although different components are discerned in the specification of a node, these components are tied together so strongly (as seen in Figure 3.2) that it is very difficult to model indiviual components, while abstracting away from others. A solution could be to create a specification that focuses on the logical functionality that is supplied, leaving room for manufacturers to fill in implementation details. For instance, the startup phase of the protocol consists of a clock synchronisation algorithm, a leader election protocol, and some initialisation logic to prepare for communicating according to the schedule. It would be much easier to prove correctness of the Flexray standard if these protocols were separately described. Currently, the three are described by a set of SDL diagrams that perform these tasks simultaneously.

All in all, both industry and academia have some steps to take before model checking can be considered mainstream technology. When they finally do meet in the middle, model checking based on fixpoint logic is a good candidate to provide the computational power to solve real-life problems, and the versatility to allow for a user-friendly interface.

Appendix A

Proofs

A.1 Proofs for Chapter 6

In Chapter 6, we formulated some lemmas that allow us to convert LFP formulas to PBES formulas and *vice versa*. The proofs for two of these lemmas use proof graphs for EFL, which were only introduced in Chapter 7. These proofs are therefore presented in this appendix.

Lemma 6.14. *Let $\mathcal{E}$ be an equation system with $\mathsf{bnd}^*(\mathcal{E}) = \{X_0, \ldots, X_n\}$, and let*

$$\mathcal{E}' = (\sigma_{X_0} X_0 \bar{x}_{X_0} = \mathsf{fo}(\varphi_{X_0})) \ldots (\sigma_{X_n} X_n \bar{x}_{X_n} = \mathsf{fo}(\varphi_{X_n})),$$

such that $X_i <_{\mathcal{E}} X_j$ for all $0 \leq i < j \leq n$. Then $\mathbf{S}^{\mathfrak{A}}(\theta, \mathcal{E})(X) = \mathbf{S}^{\mathfrak{A}}(\theta, \mathcal{E}')(X)$ for all $\mathfrak{A}, \theta$ and $X \in \mathsf{bnd}(\mathcal{E})$.

Proof. We prove that $\bar{a} \in \mathbf{S}^{\mathfrak{A}}(\theta, \mathcal{E})(X) \Leftrightarrow \bar{a} \in \mathbf{S}^{\mathfrak{A}}(\theta, \mathcal{E}')(X)$ for all $\bar{a}$ and $X \in \mathsf{bnd}(\mathcal{E})$. We only show the implication from left to right, the proof for the other direction is identical. Suppose $\bar{a} \in \mathbf{S}^{\mathfrak{A}}(\theta, \mathcal{E})(X)$ for some $X \in \mathsf{bnd}(\mathcal{E})$. Use Theorem 7.1 to obtain a proof graph G that includes a state $\langle \mathbb{t}, X, \bar{a} \rangle$. Definition 7.1 only refers to $\mathsf{bnd}^*(\mathcal{E})$, $\mathsf{fv}(\mathcal{E})$, and to $\bar{x}_X$ and $\mathsf{fo}(\varphi_X)$ for all $X \in \mathsf{bnd}^*(\mathcal{E})$. By construction, these are equal to $\mathsf{bnd}^*(\mathcal{E}')$, $\mathsf{fv}(\mathcal{E}')$, etc. (because $\mathsf{fo}(\mathsf{fo}(\varphi_X)) = \mathsf{fo}(\varphi_X)$), so G is also a dependency graph for $\mathfrak{A}, \theta$ and $\mathcal{E}'$. Definition 7.2 only refers to σ_X and $<_{\mathcal{E}}$, which are again preserved in $\mathcal{E}'$. G is therefore a proof graph for $\mathfrak{A}, \theta$ and $\mathcal{E}'$, and by Theorem 7.1, therefore $\bar{a} \in \mathbf{S}^{\mathfrak{A}}(\theta, \mathcal{E}')(X)$. □

Lemma 6.16. *Given $\mathfrak{A}$, θ, a mapping $f : \mathsf{bnd}^*(\mathcal{E}_2) \to \mathcal{X} \setminus (\mathsf{bnd}^*(\mathcal{E}) \cup \mathsf{fv}(\mathcal{E}))$, and an equation system $\mathcal{E} = \mathcal{E}_1 \mathcal{E}_2$,*

$$\mathbf{S}(\theta, \mathcal{E})(X) = \mathbf{S}(\theta, f(\mathcal{E}_2)\mathcal{E})(f(X)) \text{ for all } X \in \mathsf{bnd}(\mathcal{E}_2),$$

where f is extended to range over $\mathcal{X}$ and over equation systems in the obvious way.

Proof. We construct a proof graph for $\bar{a} \in \mathbf{S}(\theta, f(\mathcal{E}_2)\mathcal{E})(f(X)) \Leftrightarrow \alpha = \mathbb{t}$ from a proof graph for $\bar{a} \in \mathbf{S}(\theta, \mathcal{E}) \Leftrightarrow \alpha = \mathbb{t}$ (for any $\bar{a}$ and α for which the former holds).

So suppose that for some $\bar{a}$ and α, $\bar{a} \in \mathbf{S}(\theta, \mathcal{E})(X) \Leftrightarrow \alpha = \mathbb{t}$, and let G be a proof graph with node $v = \langle \alpha, X, \bar{a} \rangle$. Now add to that graph the node $f(v) = \langle \alpha, f(X), \bar{a} \rangle$,

and mark it 'unfinished'. While there is a node $f(u)$ in the graph that is marked unfinished, consider the successors $w_0, \ldots, w_n$ of u, and add the nodes $f(w_0), \ldots, f(w_n)$ to the graph, if they did not already exist ($f(w_i)$ may be equal to w_i for some, or $f(w_i)$ was already added to the graph because w_i was a successor of an unfinished node that we considered before). Add edges from $f(u)$ to each of $f(w_0), \ldots, f(w_n)$, and remove the marker from $f(u)$.

This procedure obviously terminates after some time, because we only allow one copy $f(v)$ of every node v in G to be introduced. The new graph is also easily seen to satisfy the condition from Definition 7.1: consider a node $\langle \alpha', f(Y), \bar{a}' \rangle$ that was newly introduced; because $\varphi_{f(Y)} = f(\varphi_Y)$, a simple inductive argument—almost identical to that of Lemma 7.5, and therefore left as an exercise to the reader—shows that the successors we introduced are sufficient.

To see that the condition of Definition 7.2 also applies, consider an arbitrary infinite path $p_f = f(u_0), f(u_1), \ldots$ in the newly created graph.

If $f(u_i) = u_i$ for some i (that is, the path visits a node of which the fixpoint variable is in $\mathsf{bnd}^*(\mathcal{E}_1)$), then we know that for all $j > i$, $f(u_i) = u_i$, because $\mathcal{E}$ does not refer to variables in $\mathsf{bnd}^*(f(\mathcal{E}_2))$. In other words, there is an infinite suffix of p_f that was already a path in G, so it immediately follows that the condition from Definition 7.2 also holds for p_f itself.

If $f(u_i) \neq u_i$ for all i, note that $Y <_{\mathcal{E}} Y' \Leftrightarrow f(Y) <_{\mathcal{E}'} f(Y')$ for all $Y, Y' \in \mathsf{bnd}^*(\mathcal{E}_2)$. Furthermore, because of the way we added the successors of the newly introduced nodes, $u_0, u_1, \ldots$ is a path in G. Using the correspondence between the variable orderings, it now follows that the condition from Definition 7.2 also holds on p_f. □

A.2 Proofs for Chapter 7

In Chapter 7, we left out the proofs of correctness for our encodings of stuttering bisimilarity checking and ∃ECTL* model checking into EFL, so we could focus on the problem of evidence extraction.

A.2.1 Stuttering bisimilarity

For the reader's convenience, we repeat the definition of stuttering bisimulation as given by Namjoshi. Remember that two states are stuttering bisimilar if and only if there is a stuttering bisimulation that relates them.

Definition 7.10. *Given a Kripke structure $\langle A, AP, \rightarrow, \ell \rangle$, a relation $X \subseteq A \times A$ is a* stuttering bisimulation *if and only if it is symmetric, and there exist a well-founded order $\langle W, \prec \rangle$ and some mapping $rank : A \times A \times A \rightarrow W$ such that for all s, t such that Xst:*

$$\ell(s) = \ell(t) \wedge \forall_u\, s \rightarrow u \Rightarrow ((Xut \wedge rank(u,u,t) \prec rank(s,s,t)) \vee \exists_v\, t \rightarrow v \wedge ((Xsv \wedge rank(u,s,v) \prec rank(u,s,t)) \vee Xuv)).$$

The proof of correctness for our encoding of stuttering bisimilarity checking in EFL is done by showing that the *rank* function corresponds to a combination of two

decreasing measures on Y- and Z-nodes in the proof graph for the EFL formula. Intuitively, *rank* serves as a well-founded ordering on which we can base an inductive proof that all paths in a proof graph for the EFL formula contain only finitely many Y- and Z-nodes. Conversely, paths in such a proof graph induce an ordering on Y- and Z-nodes from which we can derive a suitable *rank* function.

Proposition 7.1. *Let $\mathfrak{A}$ be a Kripke structure $\langle A, AP, \to, \ell\rangle$.*

$$\begin{aligned}\Phi lr = \; & [Xlr : (\mathbf{gfp}\, Xst = Xts \wedge \ell(s) = \ell(t) \wedge \\ & [Yst : (\mathbf{lfp}\, Yst = \forall_u\, s \to u \Rightarrow ((Xut \wedge Yut) \vee \\ & [Zsut : (\mathbf{lfp}\, Zsut = \exists_v\, t \to v \wedge ((Xsv \wedge Zsuv) \vee Xuv))])])]\end{aligned}$$

If l and r are terms of $\mathfrak{A}$ and $s = l^{\mathfrak{A}}$ and $t = r^{\mathfrak{A}}$, then $\mathfrak{A} \models \Phi lr$ if and only if $s \simeq t$.

Proof. We show the implication in both directions. First, we assume $s \simeq t$ and prove $\mathfrak{A} \models \Phi lr$.

Suppose R is a stuttering bisimulation that witnesses $s \simeq t$, and let $\langle W, \prec\rangle$ be the well-founded ordering given by Definition 7.10. Then construct a proof graph for $\mathfrak{A} \models \Phi lr$ as follows. We write $a\ R\ b$ for $\langle a, b\rangle \in R$.

- Add node $\langle \mathbb{t}, X, [a, b]\rangle$ for each $a, b \in A$ such that $a\ R\ b$.
- To each node $\langle \mathbb{t}, X, [a, b]\rangle$, add successors $\langle \mathbb{t}, =, [\ell^{\mathfrak{A}}(a), \ell^{\mathfrak{A}}(b)]\rangle$ and $\langle \mathbb{t}, Y, [a, b]\rangle$, and add a transition to $\langle \mathbb{t}, X, [b, a]\rangle$. Note that this node exists, because R is symmetric.
- To each node $\langle \mathbb{t}, Y, [a, b]\rangle$, add successor nodes as follows. For each c such that $a \to c$: add successor $\langle \mathbb{t}, Y, [c, b]\rangle$ and an edge to $\langle \mathbb{t}, X, [c, b]\rangle$ if $c\ R\ b$ and $\mathit{rank}(c, c, b) \prec \mathit{rank}(a, a, b)$, or add successor $\langle \mathbb{t}, Z, [a, c, b]\rangle$ otherwise.
- For each node $\langle \mathbb{t}, Z, [a, c, b]\rangle$, find some d such that $b \to d$ and either $c\ R\ d$, or $a\ R\ d$ and $\mathit{rank}(c, a, d) \prec \mathit{rank}(c, a, b)$. We know from Definition 7.10 that such a d exists, because $\langle \mathbb{t}, Z, [a, c, b]\rangle$ was only added if $a \to c$ and not both $c\ R\ b$ and $\mathit{rank}(c, c, b) \prec \mathit{rank}(a, a, b)$. If the latter holds, add successors $\langle \mathbb{t}, \to, [b, d]\rangle$ and $\langle \mathbb{t}, Z, [a, c, d]\rangle$, and add an edge to $\langle \mathbb{t}, X, [a, d]\rangle$. Otherwise, add $\langle \mathbb{t}, \to, [b, d]\rangle$ and an edge to $\langle \mathbb{t}, X, [c, d]\rangle$.

It is trivial to check that the requirements from Definition 7.1 hold on this graph. To see that also the requirement from Definition 7.2 holds, notice that only infinite paths without X-nodes are illegal. Such paths must then end in an infinite sequence of Y-nodes or Z-nodes, as we did not add transitions from Z- to Y-nodes. But such infinite transitions would contradict that $\prec$ is well-founded ordering w.r.t. W.

We now turn to the other implication. We assume that $\mathfrak{A} \models \Phi lr$ and prove that $s \simeq t$.

Let S be the set of states of a minimal proof graph for $\mathfrak{A} \models \Phi lr$. Define a relation $R = \{\langle a, b\rangle \mid \langle \mathbb{t}, X, [a, b]\rangle \in S\}$. We claim that R is a stuttering bisimulation. Note that the proof graph transition relation is well-founded with respect to the set of Y- and Z-nodes in S. We can therefore define $\mathit{rank} : A^3 \to \mathcal{S}^2$ and $\prec \subseteq \mathcal{S}^2 \times \mathcal{S}^2$ as follows

(because $S \subseteq \mathcal{S}$):

$$rank(x,y,z) = \langle\langle \mathbb{t}, Y, [y,z]\rangle, \langle \mathbb{t}, Z, [y,x,z]\rangle\rangle$$
$$\langle v_0', v_1'\rangle \prec \langle v_0, v_1\rangle \text{ iff } v_0 \to v_0' \vee v_0 \to v_1 \to v_1'$$

Now take any $a, b \in A$ such that $a \; R \; b$. Note that $\langle \mathbb{t}, X, [a,b]\rangle$ must have a dependency on $\langle \mathbb{t}, X, [b,a]\rangle$, and therefore $b \; R \; a$, so R is symmetric. We must now show that for a and b the formula from Definition 7.10 holds. Using Definition 7.1 we can derive that $\mathfrak{A} \models \ell(a) = \ell(b)$. Furthermore, $\langle \mathbb{t}, X, [a,b]\rangle$ has a dependency on $\langle \mathbb{t}, Y, [a,b]\rangle$. Now suppose $a \to c$. Then either $\langle \mathbb{t}, Y, [a,b]\rangle \to \langle \mathbb{t}, X, [c,b]\rangle$ and $\langle \mathbb{t}, Y, [a,b]\rangle \to \langle \mathbb{t}, Y, [c,b]\rangle$, or $\langle \mathbb{t}, Y, [a,b]\rangle \to \langle \mathbb{t}, Z, [a,c,b]\rangle$. In case of the former, $c \; R \; b$ and $rank(c,c,b) \prec rank(a,a,b)$. Otherwise, by Definition 7.1 there must be some d such that $b \to d$ and either $\langle \mathbb{t}, Z, [a,c,b]\rangle \to \langle \mathbb{t}, X, [a,d]\rangle$ and $\langle \mathbb{t}, Z, [a,c,b]\rangle \to \langle \mathbb{t}, Z, [a,c,d]\rangle$, or $\langle \mathbb{t}, Z, [a,c,b]\rangle \to \langle \mathbb{t}, X, [c,d]\rangle$. In the first case, we can in the same way as before derive that $a \; R \; d$ and $rank(c,a,d) \prec rank(c,a,b)$. Otherwise, $c \; R \; d$. □

A.2.2 ∃ECTL* model checking

We will prove soundness and completeness of our ∃ECTL* translation by constructing proof graphs. This proof is inductive, and combines proof graphs for subformulas of a ∃ECTL* formula f, obtained from the induction hypothesis, into a proof graph for f. The following definition and lemma give us the tools for this compositional approach.

Definition A.1. $\langle S, \to\rangle = \langle S_1, \to_1\rangle \uplus \langle S_2, \to_2\rangle$ *iff* $S = S_1 \cup S_2$ *and*

$$\to \; = \to_1 \cup \{\langle v, v'\rangle \mid v \to_2 v' \wedge v \notin S_1\}.$$

Lemma A.1. *If G_1 and G_2 are both proof graphs for $\mathfrak{A}, \theta$ and φ, then $G_1 \uplus G_3$ is also a proof graph for $\mathfrak{A}, \theta$ and φ.*

Proof. To see that $G_1 \uplus G_2$ is a dependency graph, note that every node always has the same set of successors as it had in G_1 or in G_2. It is also a proof graph, because no new cycles were introduced: if there was a new cycle, then it would contain nodes from $S_2 \setminus S_1$ and nodes from S_1. But nodes from S_2 only have $\to$-predecessors in S_2, and can therefore never be on a cycle with nodes from S_1. □

We have reformulated the following proposition slightly to strictly conform to the syntax of EFL. We choose to introduce a mapping I that yields for each Büchi automaton its initial state (any other encoding, *e.g.*, using an 'is-initial-state' relation, would also work, but leads to a different proof graph in the proof below). We choose to encode the case distinction as a disjunction of conjunctions; here too we could have made a different choice (we could have chosen to encode it as a conjunction of disjunctions: $(f \in \ell(s) \vee f \notin AP) \wedge \ldots$). This would lead to a slightly larger proof graph, because any minimal proof graph would have to include a node for each of the conjuncts.

We treat '$f = \neg g$' as a binary predicate on f and g, '$f = g \forall h$' and '$f = g \wedge h$' as ternary predicates on f, g and h, and '$f = \mathsf{E}(\mathcal{B})$' as a binary predicate on f and $\mathcal{B}$.

Proposition 7.2. *Let Φ be defined as:*

$$\begin{aligned}
\Phi sf = [Xsf : \mathbf{lfp}\, Xsf \quad &= (f \in \ell(s) \wedge f \in AP) \vee \\
&\quad (g \notin \ell(s) \wedge f = \neg g) \vee \\
&\quad ((Xsg \vee Xsh) \wedge f = g \vee h) \vee \\
&\quad ((Xsg \wedge Xsh) \wedge f = g \wedge h) \vee \\
&\quad (Ys(I\mathcal{B}) \wedge f = \mathsf{E}(\mathcal{B}))] \\
\mathbf{gfp}\, Ysb \quad &= Zsb \\
\mathbf{lfp}\, Zsb \quad &= \exists_{s',b',g}\ s \to s' \wedge b \xrightarrow{g} b' \wedge Xsg \wedge \\
&\qquad ((b' \in F \wedge Ys'b') \vee (b' \notin F \wedge Zs'b'))]
\end{aligned}$$

Let $\mathfrak{A}$ be a Kripke structure over AP, and let f be an $\exists$ECTL formula over AP. If s is a term of $\mathfrak{A}$ and f is a term of $\mathfrak{B}$, and $\hat{a} = s^{\mathfrak{A}}$ and $\hat{b} = f^{\mathfrak{B}_f}$, then $\mathfrak{A} \cup \mathfrak{B}_f \models \Phi sf$ if and only if $\mathfrak{A}, \hat{a} \models \hat{b}$.*

Proof. We prove the 'if' part, the 'only if' part is dual (by contraposition and using $\mathbb{f}$ instead of $\mathbb{t}$ in the proof graph nodes). Assume $\mathfrak{A}, \hat{a} \models \hat{b}$ for some $\hat{a} \in A$ and $\hat{b} \in B$, and let θ be an arbitrary environment. Let s, f be such that $\hat{a} = s^{\mathfrak{A}}$ and $\hat{b} = f^{\mathfrak{B}_f}$. We will show that there is a proof graph for $\mathfrak{A} \cup \mathfrak{B}_f$, θ and Φsf which includes node $\langle \mathbb{t}, X, [\hat{a}, \hat{b}] \rangle$.

The proof proceeds by induction on the structure of $\hat{b}$. We prove that for all a, b, if $\mathfrak{A}, a \models b$ and $b \sqsubseteq \hat{b}$ (b is a subformula of $\hat{b}$), then there is a proof graph for $\mathfrak{A} \cup \mathfrak{B}_f$, θ and Φsf that includes a node $\langle \mathbb{t}, X, [a, b] \rangle$, and in which every node $\langle \mathbb{t}, X, [a, b'] \rangle$ is such that $b' \sqsubseteq b$. Our induction hypothesis is that we have such a proof graph for every a and b such that $\mathfrak{A}, a \models b$ and $b \sqsubset \hat{b}$.

So assume $\mathfrak{A}, a \models b$ and $b \sqsubseteq \hat{b}$ for some a and b. We start creating our proof graph by adding node $\langle \mathbb{t}, X, [a, b] \rangle$, and then adding edges and nodes as follows. Following the structure of Φ, distinguish cases based on the outer symbol in b, and add the appropriate successor based on which case we are in: select appropriate b', b'', and add $\langle \mathbb{t}, \in, [b, AP] \rangle$, $\langle \mathbb{t}, = \neg, [b, b'] \rangle$, $\langle \mathbb{t}, = \forall, [b, b', b''] \rangle$, $\langle \mathbb{t}, = \wedge, [b, b', b''] \rangle$, or $\langle \mathbb{t}, = \mathsf{E}, [b, \mathcal{B}] \rangle$. Now based on these cases, add the following successors to $\langle \mathbb{t}, X, [a, b] \rangle$.

- $b \in AP$: then adding single successor $\langle \mathbb{t}, \in, [b, \ell^{\mathfrak{A}}(s)] \rangle$ suffices.
- $b = \neg b'$: adding single successor $\langle \mathbb{t}, \notin, [b', \ell^{\mathfrak{A}}(s)] \rangle$ suffices.
- $b = b' \wedge b''$: according to the semantics of $\exists$ECTL*, $\mathfrak{A}, a \models b$ if and only if $\mathfrak{A}, a \models b'$ and $\mathfrak{A}, a \models b''$. By the induction hypothesis, we have two proof graphs G_1 and G_2 for $\mathfrak{A}$ and Φsf that include nodes $\langle \mathbb{t}, X, [a, b'] \rangle$ and $\langle \mathbb{t}, X, [a, b''] \rangle$, respectively. Moreover, by Lemma A.1, $G_1 \uplus G_2$ is a proof graph for $\mathfrak{A}$, θ and Φab that contains both these nodes. If $\langle \mathbb{t}, X, [a, b] \rangle$ is not a node in this graph, then it is easy to see that adding to this graph a node $\langle \mathbb{t}, X, [a, b] \rangle$ with successors $\langle \mathbb{t}, X, [a, b'] \rangle$ and $\langle \mathbb{t}, X, [a, b''] \rangle$ is still a dependency graph. It is also a proof graph, as no new cycles could have been introduced by adding this node. We therefore have a proof graph for $\mathfrak{A}$, θ and Φab which includes node $\langle \mathbb{t}, X, [a, b] \rangle$.
- $b = b' \vee b''$: similar to the $\wedge$-case, except that only one of the proof graphs is needed.

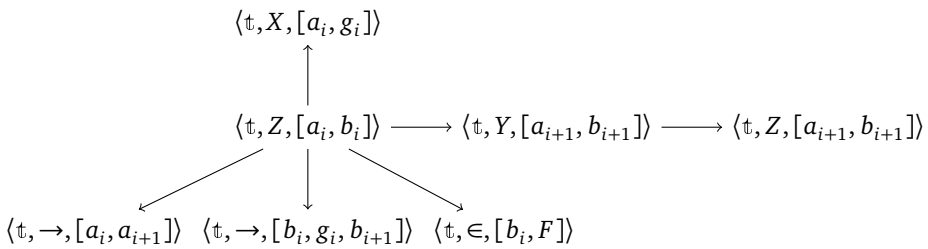


Figure A.1: Part of a proof graph for node $\langle \mathbb{t}, Y, [a_i, b_i] \rangle$ with $b_i \in F$.

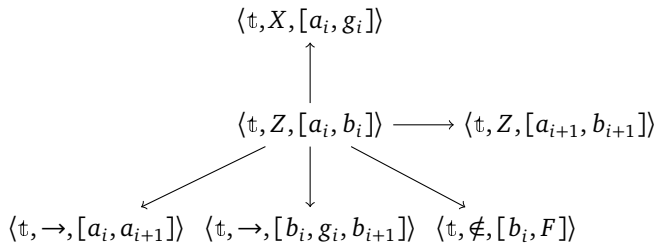


Figure A.2: Part of a proof graph for node $\langle \mathbb{t}, Y, [a_i, b_i] \rangle$ with $b_i \notin F$.

– $b = \mathsf{E}(\mathcal{B})$: let $b_0 = I^{\mathfrak{B}_f}(\mathcal{B})$. We assumed that $\mathfrak{A}, a \models b$, so there is a path $a = a_0 \to a_1 \to \ldots$ in $\mathfrak{A}$ and a path $b_0 \xrightarrow{g_0} b_1 \xrightarrow{g_1} \ldots$ in $\mathcal{B}$ such that for all g_i we have $\mathfrak{A}, a_i \models g_i$.
By the induction hypothesis we have for each such a_i and g_i a proof graph $G^{i,b'}$ for $\mathfrak{A}$ and Φsf that witnesses $\mathfrak{A}, a_i \models g_i$. Define an arbitrary complete ordering on these graphs (call them $G^0, G^1, \ldots$). By Lemma A.1, $G^* = G^0 \uplus G^1 \uplus \ldots$ is again a proof graph for $\mathfrak{A}$ and Φab. Moreover, G^* contains all nodes $\langle \mathbb{t}, X, [a_i, g_i] \rangle$ for $i \in \mathbb{N}$.
Now add to G^* the nodes $\langle \mathbb{t}, X, [a, b] \rangle$, $\langle \mathbb{t}, Y, [a, b_0] \rangle$ and $\langle \mathbb{t}, Z, [a, b_0] \rangle$. Then add the transitions $\langle \mathbb{t}, X, [a, b] \rangle \to \langle \mathbb{t}, Y, [a, b_0] \rangle$ and $\langle \mathbb{t}, Y, [a, b_0] \rangle \to \langle \mathbb{t}, Z, [a, b_0] \rangle$. Finally, for each $i \in \mathbb{N}$ (remember that $a = a_0$), add the structures from Figures A.2 and A.1.

We show that the resulting graph is a proof graph.
To see that it is a dependency graph, inspect every newly added node separately. It is easily established that the successors are sufficient to satisfy the requirements of Definition 7.1. Note that the nodes $\langle \mathbb{t}, X, [a_i, b'] \rangle$ with $[a_i, b'] \neq [a, b]$ were already present in G^*, and therefore already satisfied these constraints.
To see that it is also a proof graph, note that the only infinite path we added is the path

$$\langle \mathbb{t}, Y, [s_0, b_0] \rangle \to \langle \mathbb{t}, Z, [s_0, b_0] \rangle \to [\langle \mathbb{t}, Y, [s_1, b_1] \rangle \to] \langle \mathbb{t}, Z, [s_1, b_1] \rangle \to \ldots,$$

where the part between square brackets indicates optionally present nodes in the path. If Y occurs infinitely often on this path, then the graph is a proof graph. This

is indeed the case, because we added such a node for every $b_i \in F$. If this did not result in an infinite number of Y nodes to be added, then the original path in the Büchi automaton must have had only a finite number of accepting states in it, in which case it would not have been an accepting path of the automaton. □

Bibliography

[And92] H.R. Andersen. “Model Checking and Boolean Graphs”. In: *ESOP 1992*. Ed. by B. Krieg-Brückner. **LNCS 582**. Springer, 1992, pp. 1–19. DOI: 10.1007/3-540-55253-7_1.

[AN01] A. Arnold and D. Niwiński. *Rudiments of μ-Calculus*. **Studies in Logic and the Foundations of Mathematics 146**. North Holland, Feb. 2001. ISBN: 978-0-444-50620-7.

[AVW03] A. Arnold, A. Vincent, and I. Walukiewicz. “Games for synthesis of controllers with partial observation”. In: *Theoretical Computer Science* 303.1 (2003), pp. 7–34. DOI: 10.1016/S0304-3975(02)00442-5.

[Bae05] J.C.M. Baeten. “A brief history of process algebra”. In: *Theoretical Computer Science* 335.2–3 (2005), pp. 131–146. DOI: 10.1016/j.tcs.2004.07.036.

[BK08] C. Baier and J.-P. Katoen. *Principles of model checking*. MIT Press, 2008. ISBN: 978-0-262-02649-9.

[BNT07] D. Barsotti, L.P. Nieto, and A. Tiu. “Verification of clock synchronization algorithms: experiments on a combination of deductive tools”. In: *Formal Aspects of Computing* 19.3 (2007), pp. 321–341. DOI: 10.1007/s00165-007-0027-6.

[Bas96] T. Basten. “Branching Bisimilarity is an Equivalence Indeed!” In: *Information Processing Letters* 58.3 (1996), pp. 141–147. DOI: 10.1016/0020-0190(96)00034-8.

[BPS01] J.A. Bergstra, A. Ponse, and S.A. Smolka, eds. *Handbook of Process Algebra*. Elsevier Science Inc., Mar. 2001. ISBN: 978-0-444-82830-9.

[Ber+06] D. Berwanger, A. Dawar, P. Hunter, and S. Kreutzer. “DAG-Width and Parity Games”. In: *STACS 2006*. Ed. by B. Durand and W. Thomas. **LNCS 3884**. Springer, 2006, pp. 524–536. DOI: 10.1007/11672142_43.

[BC96] G. Bhat and R. Cleaveland. “Efficient Model Checking via the Equational μ-Calculus”. In: *LICS 1996*. IEEE Computer Society, July 1996, pp. 304–312. DOI: 10.1109/LICS.1996.561358.

[BBW06] P. Blackburn, J.F.A.K. van Benthem, and F. Wolter, eds. *Handbook of Modal Logic*. **Studies in Logic and Practical Reasoning 3**. Elsevier Science, 2006. ISBN: 978-0-444-51690-9.

[Bot+08] J. Botaschanjan, M. Broy, A. Gruler, A. Harhurin, S. Knapp, L. Kof, W.J. Paul, and M. Spichkova. "On the correctness of upper layers of automotive systems". In: *Formal Aspects of Computing* 20.6 (2008), pp. 637–662. DOI: 10.1007/s00165-008-0097-0.

[Bot+06] J. Botaschanjan, A. Gruler, A. Harhurin, L. Kof, M. Spichkova, and D. Trachtenherz. "Towards Modularized Verification of Distributed Time-Triggered Systems". In: *FM 2006*. Ed. by J. Misra, T. Nipkow, and E. Sekerinski. **LNCS 4085**. Springer, 2006, pp. 163–178. DOI: 10.1007/11813040_12.

[BS06] Julian Bradfield and Colin Stirling. "Modal mu-calculi". In: *Handbook of Modal Logic*. Ed. by P. Blackburn, J.F.A.K. van Benthem, and F. Wolter. Vol. 3. **Studies in Logic and Practical Reasoning 3**. Elsevier Science, 2006, pp. 721–756. ISBN: 978-0-444-51690-9.

[BCG88] M.C. Browne, E.M. Clarke, and O. Grumberg. "Characterizing Finite Kripke Structures in Propositional Temporal Logic". In: *Theoretical Computer Science* 59 (1988), pp. 115–131. DOI: 10.1016/0304-3975(88)90098-9.

[BS01] M. Broy and K. Stølen. *Specification and development of interactive systems: focus on streams, interfaces, and refinement*. **Monographs in Computer Science**. Springer Verlag, 2001. DOI: 10.1007/978-1-4613-0091-5.

[BFL13] F. Bruse, O. Friedmann, and M. Lange. "Guarded Transformation for the Modal mu-Calculus". In: *Computing Research Repository* (2013). arXiv: 1305.0648v2.

[Che+07] T. Chen, B. Ploeger, J.C. van de Pol, and T.A.C. Willemse. "Equivalence Checking for Infinite Systems Using Parameterized Boolean Equation Systems". In: *CONCUR 2007*. Ed. by L. Caires and V.T. Vasconcelos. **LNCS 4703**. Springer, 2007, pp. 120–135. DOI: 10.1007/978-3-540-74407-8_9.

[CGP99] E.M. Clarke Jr., O. Grumberg, and D.A. Peled. *Model Checking*. The MIT Press, 1999. ISBN: 978-0-262-03270-4.

[Cla08] E.M. Clarke. "The Birth of Model Checking". In: *25 Years of Model Checking – History, Achievements, Perspectives (part of CAV 2006)*. Ed. by O. Grumberg and H. Veith. **LNCS 5000**. Springer, 2008, pp. 1–26. DOI: 10.1007/978-3-540-69850-0_1.

[CE82] E.M. Clarke and E.A. Emerson. "Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic". In: *Logics of Programs, Workshop 1981*. Ed. by D. Kozen. **LNCS 131**. Springer, 1982, pp. 52–71. DOI: 10.1007/BFb0025774.

[Cla+95] E.M. Clarke, O. Grumberg, K.L. McMillan, and X. Zhao. "Efficient Generation of Counterexamples and Witnesses in Symbolic Model Checking". In: *DAC 1995*. Ed. by B. Preas. ACM Press, 1995, pp. 427–432. DOI: 10.1145/217474.217565.

[Cla+02] E.M. Clarke, S. Jha, Y. Lu, and H. Veith. “Tree-Like Counterexamples in Model Checking”. In: *LICS 2002*. IEEE Computer Society, 2002, pp. 19–29. DOI: 10.1109/LICS.2002.1029814.

[Cle91] R. Cleaveland. “On Automatically Explaining Bisimulation Inequivalence”. In: *CAV 1990*. Ed. by E.M. Clarke and R.P. Kurshan. **LNCS 531**. Springer, 1991, pp. 364–372. DOI: 10.1007/BFb0023750.

[CS02] R. Cleaveland and S. Sims. “Generic tools for verifying concurrent systems”. In: *Science of Computer Programming* 42.1 (2002), pp. 39–47. DOI: 10.1016/S0167-6423(01)00033-8.

[Cra10] S. Cranen. *A linear translation from LTL to the first-order modal μ-calculus to the first-order modal μ-calculus*. Computer Science Report 10-09. Technische Universiteit Eindhoven, 2010.

[Cra12a] S. Cranen. “Model Checking the FlexRay Startup Phase”. In: *FMICS 2012*. Ed. by M. Stoelinga and R. Pinger. **LNCS 7437**. Springer, 2012, pp. 131–145. DOI: 10.1007/978-3-642-32469-7_9.

[Cra12b] S. Cranen. *Model checking the FlexRay startup phase*. Computer Science Report 12-01. Technische Universiteit Eindhoven, 2012.

[Cra+ed] S. Cranen, M.W. Gazda, J.W. Wesselink, and T.A.C. Willemse. “Abstraction in Fixpoint Logic”. In: *Transactions on Computational Logic* ((*accepted*)).

[Cra+13] S. Cranen, J.F. Groote, J.J.A. Keiren, F.P.M. Stappers, E.P. de Vink, J.W. Wesselink, and T.A.C. Willemse. “An Overview of the mCRL2 Toolset and Its Recent Advances”. In: *TACAS 2013*. Ed. by N. Piterman and S.A. Smolka. **LNCS 7795**. Springer, 2013, pp. 199–213. DOI: 10.1007/978-3-642-36742-7_15.

[CGR11] S. Cranen, J.F. Groote, and M.A. Reniers. “A linear translation from CTL* to the first-order modal μ-calculus”. In: *Theoretical Computer Science* 412.28 (2011), pp. 3129–3139. DOI: 10.1016/j.tcs.2011.02.034.

[CKW11] S. Cranen, J.J.A. Keiren, and T.A.C. Willemse. “Stuttering Mostly Speeds Up Solving Parity Games”. In: *NFM 2011*. Ed. by M.G. Bobaru, K. Havelund, G.J. Holzmann, and R. Joshi. **LNCS 6617**. Springer, 2011, pp. 207–221. DOI: 10.1007/978-3-642-20398-5_16.

[CKW12a] S. Cranen, J.J.A. Keiren, and T.A.C. Willemse. *A Cure for Stuttering Parity Games*. Computer Science Report 12–05. Technische Universiteit Eindhoven, 2012.

[CKW12b] S. Cranen, J.J.A. Keiren, and T.A.C. Willemse. “A Cure for Stuttering Parity Games”. In: *ICTAC 2012*. Ed. by A. Roychoudhury and M. D’Souza. **LNCS 7521**. Springer, 2012, pp. 198–212. DOI: 10.1007/978-3-642-32943-2_16.

[CLW13] S. Cranen, B. Luttik, and T.A.C. Willemse. “Proof Graphs for Parameterised Boolean Equation Systems”. In: *CONCUR 2013*. Ed. by P.R. D’Argenio and H.C. Melgratti. **LNCS 8052**. Springer, 2013, pp. 470–484. DOI: 10.1007/978-3-642-40184-8_33.

[DPW08] A. van Dam, B. Ploeger, and T.A.C. Willemse. "Instantiation for Parameterised Boolean Equation Systems". In: *ICTAC 2008*. Ed. by J.S. Fitzgerald, A.E. Haxthausen, and H. Yenigün. **LNCS 5160**. Springer, 2008, pp. 440–454. DOI: 10.1007/978-3-540-85762-4_30.

[Dam92] M. Dam. "CTL* and ECTL* as Fragments of the Modal μ-Calculus". In: *CAAP '92, 17th Colloquium on Trees in Algebra and Programming, Rennes, France, February 26-28, 1992, Proceedings*. Ed. by J.-C. Raoult. **LNCS 581**. Springer, 1992, pp. 145–164. DOI: 10.1007/3-540-55251-0_8.

[DG08] A. Dawar and E. Grädel. "The Descriptive Complexity of Parity Games". In: *CSL 2008*. Ed. by M. Kaminski and S. Martini. **LNCS 5213**. Springer, 2008, pp. 354–368. DOI: 10.1007/978-3-540-87531-4_26.

[DV90] R. De Nicola and F.W. Vaandrager. "Action versus State based Logics for Transition Systems". In: *LITP 1990*. Ed. by I. Guessarian. **LNCS 469**. Springer, 1990, pp. 407–419. DOI: 10.1007/3-540-53479-2_17.

[DV95] Rocco De Nicola and Frits W. Vaandrager. "Three Logics for Branching Bisimulation". In: *Journal of the ACM* 42.2 (1995), pp. 458–487. DOI: 10.1145/201019.201032.

[Don03] Y. Dong. "Performance and usability issues in model checking". PhD thesis. State University of New York at Stony Brook, 2003.

[Eme97] E.A. Emerson. "Model Checking and the Mu-calculus". In: *DIMACS 1996*. Ed. by N. Immerman and P.G. Kolaitis. **DIMACS Series in Discrete Mathematics and Theoretical Computer Science 31**. American Mathematical Society, 1997, pp. 185–214. ISBN: 978-0-8218-0517-6.

[EJ91] E.A. Emerson and C.S. Jutla. "Tree Automata, Mu-Calculus and Determinacy (Extended Abstract)". In: *FOCS 1991*. IEEE Computer Society, 1991, pp. 368–377. DOI: 10.1109/SFCS.1991.185392.

[EL86] E.A. Emerson and C.-L. Lei. "Efficient Model Checking in Fragments of the Propositional Mu-Calculus (Extended Abstract)". In: *LICS 1986*. IEEE Computer Society, 1986, pp. 267–278. ISBN: 978-0-818-68720-4.

[Fle05a] Flexray consortium. *FlexRay Preliminary Central Bus Guardian Specification v2.0.9*. 2005.

[Fle05b] Flexray consortium. *FlexRay Preliminary Node Local Bus Guardian Specification v2.0.9*. 2005.

[Fle05c] Flexray consortium. *FlexRay Requirements Specification v2.1*. 2005.

[Fle09] Flexray consortium. *FlexRay Protocol Specification v3.0.1*. 2009.

[Fon+05] P. Fontaine, K. Gupta, J.Y. Marion, S. Merz, L.P. Nieto, and A. Tiu. "Towards a combination of heterogeneous deductive tools for system verification: A case study on clock synchronization". In: *APPSEM, Workshop 2005*. 2005.

[FL09] O. Friedmann and M. Lange. "Solving Parity Games in Practice". In: *ATVA 2009*. Ed. by Z. Liu and A.P. Ravn. **LNCS 5799**. Springer, 2009, pp. 182–196. DOI: 10.1007/978-3-642-04761-9_15.

[FL10] O. Friedmann and M. Lange. “A Solver for Modal Fixpoint Logics”. In: *Electronic Notes in Theoretical Computer Science* 262 (2010), pp. 99–111. DOI: 10.1016/j.entcs.2010.04.008.

[Fri05] C. Fritz. “Simulation-based simplification of omega-automata”. PhD thesis. University of Kiel, 2005.

[FW06] C. Fritz and T. Wilke. “Simulation Relations for Alternating Parity Automata and Parity Games”. In: *DLT 2006*. Ed. by O.H. Ibarra and Z. Dang. **LNCS 4036**. Springer, 2006, pp. 59–70. DOI: 10.1007/11779148_7.

[Gar+11] H. Garavel, F. Lang, R. Mateescu, and W. Serwe. “CADP 2010: A Toolbox for the Construction and Analysis of Distributed Processes”. In: *TACAS 2011*. Ed. by P.A. Abdulla and K.R.M. Leino. **LNCS 6605**. Springer, 2011, pp. 372–387. DOI: 10.1007/978-3-642-19835-9_33.

[GW12] M.W. Gazda and T.A.C. Willemse. “Consistent Consequence for Boolean Equation Systems”. In: *SOFSEM 2012*. Ed. by M. Bieliková, G. Friedrich, G. Gottlob, S. Katzenbeisser, and G. Turán. **LNCS 7147**. Springer, 2012, pp. 277–288. DOI: 10.1007/978-3-642-27660-6_23.

[GW13] M.W. Gazda and T.A.C. Willemse. “Zielonka’s Recursive Algorithm: dull, weak and solitaire games and tighter bounds”. In: *GandALF 2013*. Ed. by G. Puppis and T. Villa. **EPTCS 119**. 2013, pp. 7–20. DOI: 10.4204/EPTCS.119.4.

[Gib+14] T. Gibson-Robinson, P.J. Armstrong, A. Boulgakov, and A.W. Roscoe. “FDR3 - A Modern Refinement Checker for CSP”. In: *TACAS 2014*. Ed. by E. Ábrahám and K. Havelund. **LNCS 8413**. Springer, 2014, pp. 187–201. DOI: 10.1007/978-3-642-54862-8_13.

[Gla93] R.J. van Glabbeek. “The Linear Time - Branching Time Spectrum II”. In: *CONCUR 1994*. Ed. by E. Best. **LNCS 715**. Springer, 1993, pp. 66–81. DOI: 10.1007/3-540-57208-2_6.

[GW96] R.J. van Glabbeek and W.P. Weijland. “Branching Time and Abstraction in Bisimulation Semantics”. In: *Journal of the ACM* 43.3 (1996), pp. 555–600. DOI: 10.1145/233551.233556.

[Grä02] E. Grädel. “Model Checking Games”. In: *WoLLIC’2002*. Ed. by R. de Queiroz, L.C. Pereira, and E.H. Haeusler. **ENTCS 67**. 2002, pp. 15–34. DOI: 10.1016/S1571-0661(04)80538-3.

[GK05] J.F. Groote and M. Keinänen. “A Sub-quadratic Algorithm for Conjunctive and Disjunctive Boolean Equation Systems”. In: *ICTAC 2005*. Ed. by D.V. Hung and M. Wirsing. **LNCS 3722**. Springer, 2005, pp. 532–545. DOI: 10.1007/11560647_35.

[GM99] J.F. Groote and R. Mateescu. “Verification of Temporal Properties of Processes in a Setting with Data”. In: *AMAST 1998*. Ed. by A.M. Haeberer. **LNCS 1548**. Springer, 1999, pp. 74–90. DOI: 10.1007/3-540-49253-4_8.

[GM14] J.F. Groote and M.R. Mousavi. *Modeling and analysis of communicating systems*. The MIT Press, 2014. ISBN: 978-0-262-02771-7.

[GV90] J.F. Groote and F.W. Vaandrager. “An efficient algorithm for branching bisimulation and stuttering equivalence”. In: *ICALP 1990*. Ed. by M. Paterson. **LNCS 443**. Springer, 1990, pp. 626–638. DOI: 10.1007/BFb0032063.

[GW04] J.F. Groote and T.A.C. Willemse. “Parameterised Boolean Equation Systems (Extended Abstract)”. In: *CONCUR 2004*. Ed. by P. Gardner and N. Yoshida. **LNCS 3170**. Springer, 2004, pp. 308–324. DOI: 10.1007/978-3-540-28644-8_20.

[GW05a] J.F. Groote and T.A.C. Willemse. “Model-checking processes with data”. In: *Science of Computer Programming* 56.3 (2005), pp. 251–273. DOI: 10.1016/j.scico.2004.08.002.

[GW05b] J.F. Groote and T.A.C. Willemse. “Parameterised boolean equation systems”. In: *Theoretical Computer Science* 343.3 (2005), pp. 332–369. DOI: 10.1016/j.tcs.2005.06.016.

[HM80] M. Hennessy and R. Milner. “On Observing Nondeterminism and Concurrency”. In: *ICALP 1980*. Ed. by J.W. de Bakker and J. van Leeuwen. **LNCS 85**. Springer, 1980, pp. 299–309. DOI: 10.1007/3-540-10003-2_79.

[HR04] M. Huth and M.D. Ryan. *Logic in computer science - modelling and reasoning about systems (2. ed.)* Cambridge University Press, 2004. ISBN: 978-0-511-26401-6.

[ITU99] ITU-T. *Recommendation Z.100: Specification and Description Language (SDL)*. International Telecommunication Union (ITU), Geneva, 1999.

[Jan97] D. Janin. “Automata, Tableaus and a Reduction Theorem for Fixpoint Calculi in Arbitrary Complete Lattices”. In: *LICS 1997*. IEEE Computer Society, 1997, pp. 172–182. DOI: 10.1109/LICS.1997.614945.

[Jan05] D. Janin. *A contribution to formal methods : games, logic and automata*. Université Sciences et Technologies – Bordeaux I. 2005.

[Jur98] M. Jurdziński. “Deciding the Winner in Parity Games is in UP *cap* co-UP”. In: *Information Processing Letters* 68.3 (1998), pp. 119–124. DOI: 10.1016/S0020-0190(98)00150-1.

[Jur00] M. Jurdziński. “Small Progress Measures for Solving Parity Games”. In: *STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Lille, France, February 2000, Proceedings*. Ed. by H. Reichel and S. Tison. **LNCS 1770**. Springer, 2000, pp. 290–301. DOI: 10.1007/3-540-46541-3_24.

[JPZ06] M. Jurdziński, M. Paterson, and U. Zwick. “A deterministic subexponential algorithm for solving parity games”. In: *SODA 2006*. ACM Press, 2006, pp. 117–123. ISBN: 0-89871-605-5.
ACM: 109557.1109571.

[KP14] G. Kant and J.C. van de Pol. “Generating and Solving Symbolic Parity Games”. In: *GRAPHITE 2014*. Ed. by D. Bosnacki, S. Edelkamp, A. Lluch-Lafuente, and A.J. Wijs. **EPTCS 159**. 2014, pp. 2–14. DOI: 10.4204/EPTCS.159.2.

[Kei13] J.J.A. Keiren. “Advanced reduction techniques for model checking”. PhD thesis. Technische Universiteit Eindhoven, 2013.

[KRW12] J.J.A. Keiren, M.A. Reniers, and T.A.C. Willemse. “Structural Analysis of Boolean Equation Systems”. In: *ACM Transactions on Computational Logic* 13.1 (2012), p. 8. DOI: 10.1145/2071368.2071376.

[KWW13] J.J.A. Keiren, J.W. Wesselink, and T.A.C. Willemse. “Improved Static Analysis of Parameterised Boolean Equation Systems using Control Flow Reconstruction”. In: *Computing Research Repository* (2013). arXiv: 1304.6482v2.

[KWW14] J.J.A. Keiren, J.W. Wesselink, and T.A.C. Willemse. “Liveness Analysis for Parameterised Boolean Equation Systems”. In: *ATVA 2014*. Ed. by F. Cassez and J.-F. Raskin. **LNCS 8837**. Springer, 2014, pp. 219–234. DOI: 10.1007/978-3-319-11936-6_16.

[KW11] J.J.A. Keiren and T.A.C. Willemse. “Bisimulation Minimisations for Boolean Equation Systems”. In: *HVC 2009, Revised selected papers*. Ed. by K.S. Namjoshi, A. Zeller, and A. Ziv. **LNCS 6405**. Springer, 2011, pp. 102–116. DOI: 10.1007/978-3-642-19237-1_12.

[KKV01] V. King, O. Kupferman, and M.Y. Vardi. “On the Complexity of Parity Word Automata”. In: *FOSSACS 2001*. Ed. by F. Honsell and M. Miculan. **LNCS 2030**. Springer, 2001, pp. 276–286. DOI: 10.1007/3-540-45315-6_18.

[KV09] C. Kissig and Y. Venema. “Complementation of Coalgebra Automata”. In: *CALCO 2009*. Ed. by A. Kurz, M. Lenisa, and A. Tarlecki. **LNCS 5728**. Springer, 2009, pp. 81–96. DOI: 10.1007/978-3-642-03741-2_7.

[Kor+13] A. Kordes, B. Vermeulen, A.K. Deb, and M. Wahl. “Erhöhung der Robustheit von hybriden FlexRay-Netzwerken durch Erkennung und Eingrenzung von Laufzeitfehlern”. In: *AmE 2013*. VDE Verlag GmbH, 2013. ISBN: 978-3-8007-3485-6.

[Kor+14] A. Kordes, B. Vermeulen, A.K. Deb, and M.G. Wahl. “Startup error detection and containment to improve the robustness of hybrid FlexRay networks”. In: *DATE 2014*. IEEE, 2014, pp. 1–6. DOI: 10.7873/DATE.2014.018.

[Kor92] H. Korver. “Computing Distinguishing Formulas for Branching Bisimulation”. In: *CAV 1991*. Ed. by K.G. Larsen and A. Skou. **LNCS 575**. Springer, 1992, pp. 13–23. DOI: 10.1007/3-540-55179-4_3.

[Koz83] D. Kozen. “Results on the Propositional μ-Calculus”. In: *Theoretical Computer Science* 27 (1983), pp. 333–354. DOI: 10.1016/0304-3975(82)90125-6.

[KS06a] C. Kühnel and M. Spichkova. *FlexRay und FTCom: Formale Spezifikation in FOCUS*. Tech. rep. Technische Universität München, 2006.

[KS06b] C. Kühnel and M. Spichkova. “Upcoming Automotive Standards for Fault-Tolerant Communication: FlexRay and OSEKTime FTCom”. In: *EFTS 2006*. 2006. URL: http://www4.in.tum.de/publ/papers/FRFTCom.pdf.

[Lib04] L. Libkin. *Elements of Finite Model Theory*. Springer, 2004. ISBN: 978-3-540-21202-7.

[Lin96] H. Lin. "Symbolic Transition Graph with Assignment". In: *CONCUR 1996*. Ed. by U. Montanari and V. Sassone. **LNCS 1119**. Springer, 1996, pp. 50–65. DOI: 10.1007/3-540-61604-7_47.

[LL84] J. Lundelius and N.A. Lynch. "A new fault-tolerant algorithm for clock synchronization". In: *PODC 1984*. Ed. by T. Kameda, J. Misra, J.G. Peters, and N. Santoro. ACM, 1984, pp. 75–88. DOI: 10.1145/800222.806738.

[Mad97] A. Mader. "Verification of Modal Properties Using Boolean Equation Systems". PhD thesis. Technische Universität München, 1997.

[Mal08] J. Malinský. "The Application of the Timed Automata for FlexRay Start-Up Testing". In: *Workshop on ADC Modelling and Testing (TC4)*. IMEKO. 2008. URL: http://www.imeko.org/publications/tc4-2008/IMEKO-TC4-2008-096.pdf.

[MN10] J. Malinský and J. Novák. "Verification of Flexray Start-Up Mechanism by Timed Automata". In: *Metrology and Measurement Systems* 17.3 (2010), pp. 461–480. DOI: 10.2478/v10178-010-0039-z.

[Mat98] R. Mateescu. "Vérification des propriétés temporelles des programmes parallèles". PhD thesis. Institut National Polytechnique de Grenoble, 1998.

[Mat00] R. Mateescu. "Efficient Diagnostic Generation for Boolean Equation Systems". In: *TACAS 2000*. Ed. by S. Graf and M.I. Schwartzbach. **LNCS 1785**. Springer, 2000, pp. 251–265. DOI: 10.1007/3-540-46419-0_18.

[McN93] R. McNaughton. "Infinite Games Played on Finite Graphs". In: *Annals of Pure and Applied Logic* 65.2 (1993), pp. 149–184. DOI: 10.1016/0168-0072(93)90036-D.

[Mil80] R. Milner. *A Calculus of Communicating Systems*. **LNCS 92**. Springer, 1980. DOI: 10.1007/3-540-10235-3.

[Mos74] Y. Moschovakis. *Elementary induction on abstract structures*. North Holland, 1974. ISBN: 978-0-444-10537-0.

[Nam97] K.S. Namjoshi. "A Simple Characterization of Stuttering Bisimulation". In: *FSTTCS 1997*. Ed. by S. Ramesh and G. Sivakumar. **LNCS 1346**. Springer, 1997, pp. 284–296. DOI: 10.1007/BFb0058037.

[Obd07] J. Obdrzálek. "Clique-Width and Parity Games". In: *CSL 2007*. Ed. by J. Duparc and T.A. Henzinger. **LNCS 4646**. Springer, 2007, pp. 54–68. DOI: 10.1007/978-3-540-74915-8_8.

[OWW09] S. Orzan, J.W. Wesselink, and T.A.C. Willemse. "Static Analysis Techniques for Parameterised Boolean Equation Systems". In: *TACAS 2009*. Ed. by S. Kowalewski and A. Philippou. **LNCS 5505**. Springer, 2009, pp. 230–245. DOI: 10.1007/978-3-642-00768-2_22.

[OW10] S. Orzan and T.A.C. Willemse. “Invariants for Parameterised Boolean Equation Systems”. In: *Theoretical Computer Science* 411.11-13 (2010), pp. 1338–1371. DOI: 10.1016/j.tcs.2009.11.001.

[PT87] R. Paige and R.E. Tarjan. “Three partition refinement algorithms”. In: *SIAM Journal on Computing* 16.6 (1987), pp. 973–989. DOI: 10.1137/0216062.

[Par81] D.M.R. Park. “Concurrency and Automata on Infinite Sequences”. In: *TCS 1981*. Ed. by P. Deussen. **LNCS 104**. Springer, 1981, pp. 167–183. DOI: 10.1007/BFb0017309.

[PW08] J.C. van de Pol and M. Weber. “A Multi-Core Solver for Parity Games”. In: *Electronic Notes in Theoretical Computer Science* 220.2 (2008), pp. 19–34. DOI: 10.1016/j.entcs.2008.11.011.

[Pop+08] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei. “Timing analysis of the FlexRay communication protocol”. In: *Real-Time Systems* 39.1-3 (2008), pp. 205–235. DOI: 10.1007/s11241-007-9040-3.

[Sah74] S. Sahni. “Computationally Related Problems”. In: *SIAM Journal on Computing* 3.4 (1974), pp. 262–279. DOI: 10.1137/0203021.

[Sch07] S. Schewe. “Solving Parity Games in Big Steps”. In: *FSTTCS 2007*. Ed. by V. Arvind and S. Prasad. **LNCS 4855**. Springer, 2007, pp. 449–460. DOI: 10.1007/978-3-540-77050-3_37.

[Spi06] M. Spichkova. *FlexRay: Verification of the FOCUS Specification in Isabelle/HOL. A Case Study*. Tech. rep. Technische Universität München, 2006.

[Ste05a] W. Steiner. *An assessment of FlexRay 2.0 (safety aspects)*. Tech. rep. 45/2005. Technische Universität Wien, 2005.

[Ste05b] W. Steiner. *Model-Checking Studies of the FlexRay Startup Algorithm*. Tech. rep. 57/200. Technische Universität Wien, 2005.

[SS98] P. Stevens and C. Stirling. “Practical Model-Checking Using Games”. In: *TACAS 1998*. Ed. by B. Steffen. **LNCS 1384**. Springer, 1998, pp. 85–101. DOI: 10.1007/BFb0054166.

[Tan02] L. Tan. “Evidence-Based Verification”. PhD thesis. Department of Computer Science, State University of New York, 2002.

[TC02] L. Tan and R. Cleaveland. “Evidence-Based Model Checking”. In: *CAV 2002*. Ed. by E. Brinksma and K.G. Larsen. **LNCS 2404**. Springer, 2002, pp. 455–470. DOI: 10.1007/3-540-45657-0_37.

[Tar55] A. Tarski. “A lattice-theoretical fixpoint theorem and its applications”. In: *Pacific journal of Mathematics* 5.2 (1955), pp. 285–309. ZBM: 0064.26004.

[Tho89] W. Thomas. “Computation tree logic and regular omega-languages”. In: *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, School/Workshop 1988*. Ed. by J.W. de Bakker, W.P. de Roever, and G. Rozenberg. **LNCS 354**. Springer, 1989, pp. 690–713. DOI: 10.1007/BFb0013041.

[Ver13] M. Verver. “Practical improvements to parity game solving”. MA thesis. University of Twente, 2013. URL: http://essay.utwente.nl/64985/1/practical-improvements-to-parity-game-solving.pdf (visited on 12/10/2014).

[VJ00] J. Vöge and M. Jurdzinski. “A Discrete Strategy Improvement Algorithm for Solving Parity Games”. In: *CAV 2000*. Ed. by E.A. Emerson and A.P. Sistla. **LNCS 1855**. Springer, 2000, pp. 202–215. DOI: 10.1007/10722167_18.

[Zha06] B. Zhang. “On the Formal Verification of the FlexRay Communication Protocol”. In: *AVoCS 2006 (unpublished proceedings)*. 2006, pp. 184–189.

[Zha08] B. Zhang. “Specifying and Verifying Timing Properties of a Time-triggered Protocol for In-vehicle Communication”. In: *SNPD 2008*. IEEE Computer Society. 2008, pp. 467–472. DOI: 10.1109/SNPD.2008.99.

[ZC05] D. Zhang and R. Cleaveland. “Fast Generic Model-Checking for Data-Based Systems”. In: *FORTE 2005*. Ed. by F. Wang. **LNCS 3731**. Springer, 2005, pp. 83–97. DOI: 10.1007/11562436_8.

[Zie98] W. Zielonka. “Infinite Games on Finitely Coloured Graphs with Applications to Automata on Infinite Trees”. In: *Theoretical Computer Science* 200.1-2 (1998), pp. 135–183. DOI: 10.1016/S0304-3975(98)00009-7.

Summary

Getting the point
Obtaining and understanding fixpoints in model checking

High-tech industry has become heavily reliant on correctly functioning software and logical circuits. A case in point is the automotive industry, where so-called *by-wire* techniques, implemented as hybrid hardware/software systems, replace and enhance mechanical solutions for steering and braking.

In this thesis, we investigate how model checking techniques based on fixpoint logic can help in analysing these systems, and how these techniques might be improved to make them easier to use.

Using existing technology, we analyse a part of the FlexRay communication protocol, which is used in the automotive industry to provide a communication channel between the various—sometimes safety-critical—parts of a car. A previously unknown error is uncovered by model checking the protocol under various fault models.

We then turn our attention the the process of model checking that was used in the FlexRay case study, and develop theory to address some of the pitfalls we encountered there. We leave the context of our case study, and address the following three issues.

– Using first order modal fixpoint logic, one can describe the desired behaviour of a system, as we demonstrated in the case study. While this logic is expressive enough to formulate the properties we are interested in, it can appear rather enigmatic to those with no background in fixpoint logic. We show that the more accessible logic CTL* can be translated succinctly into the logic that was used in the case study, and that model checking the translated formula has the same time complexity as solving the original formula with a CTL* model checker. The problem of translating CTL* to modal fixpoint logic is in itself interesting, because no succinct translation existed previously; known translations caused at least an exponential blowup in the size of the formula. The translation in this thesis is linear, because we use a first-order fixpoint logic.
– A major problem in model checking is that the size of the semantic model (a graph structure) of specifications of realistic industrial systems is often immense. Fixpoint logic formulas that encode model checking problems on such systems can be solved by generating a graph structure called a *parity game*, and calculating the solution to the model checking problem from this graph. These parity games also grow impractically large. We therefore investigate means to reduce the size of these games,

while preserving their solution.

– Finally, we design a method to provide feedback to the user of a fixpoint model checker, by equipping fixpoint logic with a notion of *evidence*. In the FlexRay case study, it occurred that after two days of computing, the model checker returned with the answer 'no', where we expected it to be 'yes'. With our notion of evidence, the model checker could also have indicated what part of the protocol we needed to look at to understand *why* the answer was 'no'.

  To make our results more generally applicable, the notion of evidence is defined for a fixpoint logic that generalizes LFP (least fixpoint logic), which extends first-order logic with a fixpoint operator, and PBES (parameterized Boolean equation systems), an equational fixpoint logic that underlies the model checker we used for our case study.

Curriculum vitae

Sjoerd Cranen was born on 17 December 1984 in Wijchen, The Netherlands. In 2002, he finished his secondary education at Dominicus College in Nijmegen, and went on to study computer science at Eindhoven University of Technology. He received his bachelor's degree in 2006, and graduated *cum laude* from the Computer Science & Engineering master's programme in 2008, after doing his graduation project on the topic of speech recognition at the University of Sheffield, United Kingdom.

From 2008 to 2009, Sjoerd worked as a software engineer at Imtech ICT Technical Systems in Amersfoort, now part of Axians, mainly working on a real-time publish/subscribe protocol and on the coordination software for the Dutch-German wind tunnels in Marknesse, The Netherlands.

In October 2009, he returned to Eindhoven University of Technology, where he has worked as a PhD student until March 2015, resulting in this thesis. Since 2012, he is also a partner in Form ICT, which was founded together with dr. Frank Stappers with the goal to apply formal methods in industry.

Titles in the IPA Dissertation Series

since 2009

M.H.G. Verhoef. *Modeling and Validating Distributed Embedded Real-Time Control Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2009-01

M. de Mol. *Reasoning about Functional Programs: Sparkle, a proof assistant for Clean*. Faculty of Science, Mathematics and Computer Science, RU. 2009-02

M. Lormans. *Managing Requirements Evolution*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-03

M.P.W.J. van Osch. *Automated Model-based Testing of Hybrid Systems*. Faculty of Mathematics and Computer Science, TU/e. 2009-04

H. Sozer. *Architecting Fault-Tolerant Software Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-05

M.J. van Weerdenburg. *Efficient Rewriting Techniques*. Faculty of Mathematics and Computer Science, TU/e. 2009-06

H.H. Hansen. *Coalgebraic Modelling: Applications in Automata Theory and Modal Logic*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-07

A. Mesbah. *Analysis and Testing of Ajax-based Single-page Web Applications*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-08

A.L. Rodriguez Yakushev. *Towards Getting Generic Programming Ready for Prime Time*. Faculty of Science, UU. 2009-9

K.R. Olmos Joffré. *Strategies for Context Sensitive Program Transformation*. Faculty of Science, UU. 2009-10

J.A.G.M. van den Berg. *Reasoning about Java programs in PVS using JML*. Faculty of Science, Mathematics and Computer Science, RU. 2009-11

M.G. Khatib. *MEMS-Based Storage Devices. Integration in Energy-Constrained Mobile Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-12

S.G.M. Cornelissen. *Evaluating Dynamic Analysis Techniques for Program Comprehension*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-13

D. Bolzoni. *Revisiting Anomaly-based Network Intrusion Detection Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-14

H.L. Jonker. *Security Matters: Privacy in Voting and Fairness in Digital Exchange*. Faculty of Mathematics and Computer Science, TU/e. 2009-15

M.R. Czenko. *TuLiP - Reshaping Trust Management*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-16

T. Chen. *Clocks, Dice and Processes*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-17

C. Kaliszyk. *Correctness and Availability: Building Computer Algebra on top of Proof Assistants and making Proof Assistants available over the Web*. Faculty of Science, Mathematics and Computer Science, RU. 2009-18

R.S.S. O'Connor. *Incompleteness & Completeness: Formalizing Logic and Analysis in Type Theory*. Faculty of Science, Mathematics and Computer Science, RU. 2009-19

B. Ploeger. *Improved Verification Methods for Concurrent Systems*. Faculty of Mathematics and Computer Science, TU/e. 2009-20

T. Han. *Diagnosis, Synthesis and Analysis of Probabilistic Models*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-21

R. Li. *Mixed-Integer Evolution Strategies for Parameter Optimization and Their Applications to Medical Image Analysis*. Faculty of Mathematics and Natural Sciences, UL. 2009-22

J.H.P. Kwisthout. *The Computational Complexity of Probabilistic Networks*. Faculty of Science, UU. 2009-23

T.K. Cocx. *Algorithmic Tools for Data-Oriented Law Enforcement*. Faculty of Mathematics and Natural Sciences, UL. 2009-24

A.I. Baars. *Embedded Compilers*. Faculty of Science, UU. 2009-25

M.A.C. Dekker. *Flexible Access Control for Dynamic Collaborative Environments*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-26

J.F.J. Laros. *Metrics and Visualisation for Crime Analysis and Genomics*. Faculty of Mathematics and Natural Sciences, UL. 2009-27

C.J. Boogerd. *Focusing Automatic Code Inspections*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2010-01

M.R. Neuhäußer. *Model Checking Nondeterministic and Randomly Timed Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-02

J. Endrullis. *Termination and Productivity*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-03

T. Staijen. *Graph-Based Specification and Verification for Aspect-Oriented Languages*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-04

Y. Wang. *Epistemic Modelling and Protocol Dynamics*. Faculty of Science, UvA. 2010-05

J.K. Berendsen. *Abstraction, Prices and Probability in Model Checking Timed Automata*. Faculty of Science, Mathematics and Computer Science, RU. 2010-06

A. Nugroho. *The Effects of UML Modeling on the Quality of Software*. Faculty of Mathematics and Natural Sciences, UL. 2010-07

A. Silva. *Kleene Coalgebra*. Faculty of Science, Mathematics and Computer Science, RU. 2010-08

J.S. de Bruin. *Service-Oriented Discovery of Knowledge - Foundations, Implementations and Applications*. Faculty of Mathematics and Natural Sciences, UL. 2010-09

D. Costa. *Formal Models for Component Connectors*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-10

M.M. Jaghoori. *Time at Your Service: Schedulability Analysis of Real-Time and Distributed Services*. Faculty of Mathematics and Natural Sciences, UL. 2010-11

R. Bakhshi. *Gossiping Models: Formal Analysis of Epidemic Protocols*. Faculty of Sciences, Department of Computer Science, VUA. 2011-01

B.J. Arnoldus. *An Illumination of the Template Enigma: Software Code Generation with Templates*. Faculty of Mathematics and Computer Science, TU/e. 2011-02

E. Zambon. *Towards Optimal IT Availability Planning: Methods and Tools*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-03

L. Astefanoaei. *An Executable Theory of Multi-Agent Systems Refinement*. Faculty of Mathematics and Natural Sciences, UL. 2011-04

J. Proença. *Synchronous coordination of distributed components*. Faculty of Mathematics and Natural Sciences, UL. 2011-05

A. Moralı. *IT Architecture-Based Confidentiality Risk Assessment in Networks of Organizations*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-06

M. van der Bijl. *On changing models in Model-Based Testing*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-07

C. Krause. *Reconfigurable Component Connectors*. Faculty of Mathematics and Natural Sciences, UL. 2011-08

M.E. Andrés. *Quantitative Analysis of Information Leakage in Probabilistic and Nondeterministic Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2011-09

M. Atif. *Formal Modeling and Verification of Distributed Failure Detectors*. Faculty of Mathematics and Computer Science, TU/e. 2011-10

P.J.A. van Tilburg. *From Computability to Executability – A process-theoretic view on automata theory*. Faculty of Mathematics and Computer Science, TU/e. 2011-11

Z. Protic. *Configuration management for models: Generic methods for model comparison and model co-evolution*. Faculty of Mathematics and Computer Science, TU/e. 2011-12

S. Georgievska. *Probability and Hiding in Concurrent Processes*. Faculty of Mathematics and Computer Science, TU/e. 2011-13

S. Malakuti. *Event Composition Model: Achieving Naturalness in Runtime Enforcement*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-14

M. Raffelsieper. *Cell Libraries and Verification*. Faculty of Mathematics and Computer Science, TU/e. 2011-15

C.P. Tsirogiannis. *Analysis of Flow and Visibility on Triangulated Terrains*. Faculty of Mathematics and Computer Science, TU/e. 2011-16

Y.-J. Moon. *Stochastic Models for Quality of Service of Component Connectors*. Faculty of Mathematics and Natural Sciences, UL. 2011-17

R. Middelkoop. *Capturing and Exploiting Abstract Views of States in OO Verification*. Faculty of Mathematics and Computer Science, TU/e. 2011-18

M.F. van Amstel. *Assessing and Improving the Quality of Model Transformations*. Faculty of Mathematics and Computer Science, TU/e. 2011-19

A.N. Tamalet. *Towards Correct Programs in Practice*. Faculty of Science, Mathematics and Computer Science, RU. 2011-20

H.J.S. Basten. *Ambiguity Detection for Programming Language Grammars*. Faculty of Science, UvA. 2011-21

M. Izadi. *Model Checking of Component Connectors*. Faculty of Mathematics and Natural Sciences, UL. 2011-22

L.C.L. Kats. *Building Blocks for Language Workbenches*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2011-23

S. Kemper. *Modelling and Analysis of Real-Time Coordination Patterns*. Faculty of Mathematics and Natural Sciences, UL. 2011-24

J. Wang. *Spiking Neural P Systems*. Faculty of Mathematics and Natural Sciences, UL. 2011-25

A. Khosravi. *Optimal Geometric Data Structures*. Faculty of Mathematics and Computer Science, TU/e. 2012-01

A. Middelkoop. *Inference of Program Properties with Attribute Grammars, Revisited*. Faculty of Science, UU. 2012-02

Z. Hemel. *Methods and Techniques for the Design and Implementation of Domain-Specific*

Languages. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2012-03

T. Dimkov. *Alignment of Organizational Security Policies: Theory and Practice*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-04

S. Sedghi. *Towards Provably Secure Efficiently Searchable Encryption*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-05

F. Heidarian Dehkordi. *Studies on Verification of Wireless Sensor Networks and Abstraction Learning for System Inference*. Faculty of Science, Mathematics and Computer Science, RU. 2012-06

K. Verbeek. *Algorithms for Cartographic Visualization*. Faculty of Mathematics and Computer Science, TU/e. 2012-07

D.E. Nadales Agut. *A Compositional Interchange Format for Hybrid Systems: Design and Implementation*. Faculty of Mechanical Engineering, TU/e. 2012-08

H. Rahmani. *Analysis of Protein-Protein Interaction Networks by Means of Annotated Graph Mining Algorithms*. Faculty of Mathematics and Natural Sciences, UL. 2012-09

S.D. Vermolen. *Software Language Evolution*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2012-10

L.J.P. Engelen. *From Napkin Sketches to Reliable Software*. Faculty of Mathematics and Computer Science, TU/e. 2012-11

F.P.M. Stappers. *Bridging Formal Models – An Engineering Perspective*. Faculty of Mathematics and Computer Science, TU/e. 2012-12

W. Heijstek. *Software Architecture Design in Global and Model-Centric Software Development*. Faculty of Mathematics and Natural Sciences, UL. 2012-13

C. Kop. *Higher Order Termination*. Faculty of Sciences, Department of Computer Science, VUA. 2012-14

A. Osaiweran. *Formal Development of Control Software in the Medical Systems Domain*. Faculty of Mathematics and Computer Science, TU/e. 2012-15

W. Kuijper. *Compositional Synthesis of Safety Controllers*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-16

H. Beohar. *Refinement of Communication and States in Models of Embedded Systems*. Faculty of Mathematics and Computer Science, TU/e. 2013-01

G. Igna. *Performance Analysis of Real-Time Task Systems using Timed Automata*. Faculty of Science, Mathematics and Computer Science, RU. 2013-02

E. Zambon. *Abstract Graph Transformation – Theory and Practice*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2013-03

B. Lijnse. *TOP to the Rescue – Task-Oriented Programming for Incident Response Applications*. Faculty of Science, Mathematics and Computer Science, RU. 2013-04

G.T. de Koning Gans. *Outsmarting Smart Cards*. Faculty of Science, Mathematics and Computer Science, RU. 2013-05

M.S. Greiler. *Test Suite Comprehension for Modular and Dynamic Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2013-06

L.E. Mamane. *Interactive mathematical documents: creation and presentation*. Faculty of Science, Mathematics and Computer Science, RU. 2013-07

M.M.H.P. van den Heuvel. *Composition and synchronization of real-time components upon one processor*. Faculty of Mathematics and Computer Science, TU/e. 2013-08

J. Businge. *Co-evolution of the Eclipse Framework and its Third-party Plug-ins*. Faculty of Mathematics and Computer Science, TU/e. 2013-09

S. van der Burg. *A Reference Architecture for Distributed Software Deployment*. Faculty

of Electrical Engineering, Mathematics, and Computer Science, TUD. 2013-10

J.J.A. Keiren. *Advanced Reduction Techniques for Model Checking*. Faculty of Mathematics and Computer Science, TU/e. 2013-11

D.H.P. Gerrits. *Pushing and Pulling: Computing push plans for disk-shaped robots, and dynamic labelings for moving points*. Faculty of Mathematics and Computer Science, TU/e. 2013-12

M. Timmer. *Efficient Modelling, Generation and Analysis of Markov Automata*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2013-13

M.J.M. Roeloffzen. *Kinetic Data Structures in the Black-Box Model*. Faculty of Mathematics and Computer Science, TU/e. 2013-14

L. Lensink. *Applying Formal Methods in Software Development*. Faculty of Science, Mathematics and Computer Science, RU. 2013-15

C. Tankink. *Documentation and Formal Mathematics — Web Technology meets Proof Assistants*. Faculty of Science, Mathematics and Computer Science, RU. 2013-16

C. de Gouw. *Combining Monitoring with Runtime Assertion Checking*. Faculty of Mathematics and Natural Sciences, UL. 2013-17

J. van den Bos. *Gathering Evidence: Model-Driven Software Engineering in Automated Digital Forensics*. Faculty of Science, UvA. 2014-01

D. Hadziosmanovic. *The Process Matters: Cyber Security in Industrial Control Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-02

A.J.P. Jeckmans. *Cryptographically-Enhanced Privacy for Recommender Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-03

C.-P. Bezemer. *Performance Optimization of Multi-Tenant Software Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2014-04

T.M. Ngo. *Qualitative and Quantitative Information Flow Analysis for Multi-threaded Programs*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-05

A.W. Laarman. *Scalable Multi-Core Model Checking*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-06

J. Winter. *Coalgebraic Characterizations of Automata-Theoretic Classes*. Faculty of Science, Mathematics and Computer Science, RU. 2014-07

W. Meulemans. *Similarity Measures and Algorithms for Cartographic Schematization*. Faculty of Mathematics and Computer Science, TU/e. 2014-08

A.F.E. Belinfante. *JTorX: Exploring Model-Based Testing*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-09

A.P. van der Meer. *Domain Specific Languages and their Type Systems*. Faculty of Mathematics and Computer Science, TU/e. 2014-10

B.N. Vasilescu. *Social Aspects of Collaboration in Online Software Communities*. Faculty of Mathematics and Computer Science, TU/e. 2014-11

F.D. Aarts. *Tomte: Bridging the Gap between Active Learning and Real-World Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2014-12

N. Noroozi. *Improving Input-Output Conformance Testing Theories*. Faculty of Mathematics and Computer Science, TU/e. 2014-13

M. Helvensteijn. *Abstract Delta Modeling: Software Product Lines and Beyond*. Faculty of Mathematics and Natural Sciences, UL. 2014-14

P. Vullers. *Efficient Implementations of Attribute-based Credentials on Smart Cards*. Faculty of Science, Mathematics and Computer Science, RU. 2014-15

F.W. Takes. *Algorithms for Analyzing and Mining Real-World Graphs*. Faculty of Mathematics and Natural Sciences, UL. 2014-16

M.P. Schraagen. *Aspects of Record Linkage*. Faculty of Mathematics and Natural Sciences, UL. 2014-17

G. Alpár. *Attribute-Based Identity Management: Bridging the Cryptographic Design of ABCs with the Real World*. Faculty of Science, Mathematics and Computer Science, RU. 2015-01

A.J. van der Ploeg. *Efficient Abstractions for Visualization and Interaction*. Faculty of Science, UvA. 2015-02

R.J.M. Theunissen. *Supervisory Control in Health Care Systems*. Faculty of Mechanical Engineering, TU/e. 2015-03

T.V. Bui. *A Software Architecture for Body Area Sensor Networks: Flexibility and Trustworthiness*. Faculty of Mathematics and Computer Science, TU/e. 2015-04

A. Guzzi. *Supporting Developers' Teamwork from within the IDE*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2015-05

T. Espinha. *Web Service Growing Pains: Understanding Services and Their Clients*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2015-06

S. Dietzel. *Resilient In-network Aggregation for Vehicular Networks*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2015-07

E. Costante. *Privacy throughout the Data Cycle*. Faculty of Mathematics and Computer Science, TU/e. 2015-08

S. Cranen. *Getting the point — Obtaining and understanding fixpoints in model checking*. Faculty of Mathematics and Computer Science, TU/e. 2015-09