Generalized Simulation Relations with Applications in Automata Theory

Lorenzo Clemente

Doctor of Philosophy

Laboratory for Foundations of Computer Science
School of Informatics
University of Edinburgh

2011

Abstract

Finite-state automata are a central computational model in computer science, with numerous and diverse applications. In one such application, *viz.* model-checking, automata over infinite words play a central rôle. In this thesis, we concentrate on *Büchi automata* (BA), which are arguably the simplest finite-state model recognizing languages of infinite words. Two algorithmic problems are paramount in the theory of automata: *language inclusion* and *automata minimization*. They are both PSPACE-complete, thus under standard complexity-theoretic assumptions no deterministic algorithm with worst case polynomial time can be expected. In this thesis, we develop techniques to tackle these problems.

In automata minimization, one seeks the smallest automaton recognizing a given language ("small" means with few states). Despite PSPACE-hardness of minimization, the size of an automaton can often be reduced substantially by means of quotienting. In *quotienting*, states deemed equivalent according to a given equivalence are merged together; if this merging operation preserves the language, then the equivalence is said to be *Good for Quotienting* (GFQ). In general, quotienting cannot achieve exact minimization, but, in practice, it can still offer a very good reduction in size. The central topic of this thesis is the design of GFQ equivalences for Büchi automata.

A particularly successful approach to the design of GFQ equivalences is based on simulation relations. *Simulation relations* are a powerful tool to compare the local behavior of automata. The main contribution of this thesis is to generalize simulations, by relaxing locality in three perpendicular ways: by fixing the input word in advance (*fixed-word simulations*, Ch. 3), by allowing jumps (*jumping simulations*, Ch. 4), and by using multiple pebbles (*multipebble simulations for alternating BA*, Ch. 5). In each case, we show that our generalized simulations induce GFQ equivalences. For fixed-word simulation, we argue that it is the coarsest GFQ simulation implying language inclusion, by showing that it subsumes a natural hierarchy of GFQ multipebble simulations.

From a theoretical perspective, our study significantly extends the theory of simulations for BA; relaxing locality is a general principle, and it may find useful applications outside automata theory. From a practical perspective, we obtain GFQ equivalences coarser than previously possible. This yields smaller quotient automata, which is beneficial in applications. Finally, we show how simulation relations have recently been applied to significantly optimize *exact* (exponential) language inclusion algorithms (Ch. 6), thus extending their practical applicability.

Acknowledgements

Writing a thesis is a complex endeavor, and it is but the last one in the long journey of Ph.D. studies. In these few lines, I take occasion for thanking those people which made this voyage worthwhile and rewarding.

First and foremost, I thank my supervisor Richard Mayr. If in these three years I have achieved anything at all, then I cannot even imagine how this could have been possible without his guidance and acumen. His expertise allowed me to discover many beautiful topics, which would otherwise had remained buried under an overwhelming literature. I'm indebted to him for being coauthor of part of the work presented here, and mentor for the rest of it.

I would also like to thank Alex Simpson and my second supervisor Kousha Etessami for kindly agreeing to become members of my progress panels.

I am honored to have Leonid Libkin and Thomas Wilke as my examiners. Thank you for agreeing to embark upon reading this thesis, and for providing valuable feedback.

I thank Anca Muscholl and Igor Walukiewicz for hosting my stay in Bordeaux after I left Edinburgh, when the last part of this thesis has been written.

I especially thank my former advisor Prof. Alberto Pettorossi for introducing me to Theoretical Computer Science. From Alberto, I learned that *"L'essentiel est invisible pour les yeux"*; more practically, his valuable map of the *Athens of the North* prevented me from getting lost in Edinburgh. Thanks also to Prof. Alberto Pettorossi and Prof. Anton Wakolbinger who, by writing two very convincing recommendation letters, made my Ph.D. application successful.

Working with other people has been great, and made me feel part of a larger project. For this, I thank my co-authors Parosh A. Abdulla, Yu-Fang Chen, Lukáš Holík, Chih-Duo Hong, and Tomáš Vojnar. I especially thank Parosh for hosting me in Uppsala twice, which, amongst other things, allowed me to experience some *real* winter.

My stay in Scotland has been turned into a much more pleasing experience thanks to many colleagues and friends: Julian&Teresa, who fully embody the South American warmth; Willem&Saskia, who have never met a game they didn't like; Ezra, with whom I had many funny and interesting conversations; Patrick, who patiently agreed listening to my research in many occasions; Wenyuan Yu and Xin Wang, who taught me, perhaps unwittingly, a lot about Chinese culture; Ben, who is small; Matteo, who is Italian, and, still, can eat Scottish pizza every day; Gavin, who used the same plastic bag for as long as I can remember; Grant, whose name always *grants* puns galore; Claire,

whose irresistibly convincing glance persuaded me to organize the *Friday Cake*^{TM} for a couple of months (I hope the tradition will be kept in good shape in Patrick's hands!); Anthony W. *Lin*, who knows how to play some serious table tennis, and who knows how to change his surname (for the benefit of acknowledgement writers); Václav *Brožek*, who knows how to have some serious fun during school and conference trips (we had much!), and who *does not know* how to change his surname (for the benefit of acknowledgement writers); Lucy, who can cycle really fast; my flatmate Adriana, who has a delightful habit of not-so-quiet singing—especially at night; Adam, who can prepare an amazing, almost *lyrical* hot chocolate; Magdalena, who has a full-fledged bakery at home, Tomasz, who has the most compelling humor I ever came across, and Helena, who already knows how to win an eBay auction.

Special thanks go to Beatrice and Erik for always listening to my jokes in the hope that they would eventually improve; or perhaps definitively stop, *basically*. I also thank Beatrice and Elena for once hosting me, while continually believing that I was leaving on the following day.

In Bordeaux I have found new friends who enlightened the final phase of my Ph.D.: Sri, who has never seen a sentence which was precise enough, *never never never*; Petru, who has never produced such a sentence, but *it's OK*; Anna, who makes beautiful handwritten postcards; Dominik, who is funny, and, you know, *fun is fun*; and Sagnik W. *Catan*, who has invited me to what will be an awesome Indian wedding!

Many people sustained me during my period abroad. I regard myself lucky to be the recipient of the longstanding friendship of Alessandro Checco, Francesca Piacentini, and Francesco Marino. Every time I leave, our bond grows stronger.

I profoundly thank my family and my lovely sister Martina for always offering me untiring and indefatigable support. It is their unconditional love that allowed me to become who I am.

Last, and foremost, I thank Giulia. She deserves the greatest thank of all for her unfaltering love, unswerving commitment to a common project, and, more worldly, for the most awesome strawberry cake one can think of! Giulia, this thesis is especially dedicated to you.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Lorenzo Clemente*)

Table of Contents

Page

Abstract iii

Declaration vi

List of Figures ix

List of Tables x

List of Examples xi

1 Introduction 1

1.1 Overview 1

1.2 Formal methods 2

1.3 Model-checking 3

1.4 Automata theory 4

1.5 Simulation preorders 8

1.6 Contributions and thesis structure 11

2 Background 15

2.1 Overview 16

2.2 Basic notation 16

2.3 Games 18

2.4 Automata 20

2.5 Simulation preorders 25

2.6 Simulation-based automata simplification 30

3 Fixed-word simulations 35

3.1 Overview 35

3.2 Multipebble simulations 37

3.3 Containment preorders 42
3.4 Fixed-word simulations 45
3.5 Fixed-word multipebble delayed simulation 56
3.6 Fixed-words and ranks 57

4 Jumping simulations **71**
4.1 Overview 72
4.2 Preliminaries 74
4.3 Jumping simulation transformers 76
4.4 Language containment and inclusion 84
4.5 Reflexivity and jumping-safety 91
4.6 Proxy simulation hierarchies 98
4.7 Proxy simulations vs Mediated preorder 109
4.8 Computing jumping simulations 116

5 Multipebble simulations for ABAs **119**
5.1 Applications of alternating automata 124
5.2 Simulations for alternating Büchi automata 127
5.3 Multipebble simulations for alternating Büchi automata 130
5.4 Basic properties of multipebble simulations 135
5.5 Uniform simulation games 145
5.6 Transitivity 158
5.7 Multipebble simulations and subset-like constructions 177
5.8 Quotienting 193
5.9 Computing multipebble simulations 219
5.10 Complexity of multipebble simulations 225

6 Conclusions **245**
6.1 Overview 245
6.2 Applications 246
6.3 Recapitulation and further work 247
6.4 Advanced language inclusion checking 250
6.5 Conclusions 264

Bibliography **265**

Index **278**

List of Figures

3.1 More pebbles help Duplicator 38
3.2 Simulation, pebbles, and containment 46
3.3 Pebbles, fixed-words, and containment 54
3.4 A running example 61
3.5 Ranks of the running example 63

4.1 Composing games 83
4.2 Coherent paths 90
4.3 Proxy simulations hierarchies 102

5.1 Multipebble simulation hierarchy 138
5.2 Translation to uniform automata and games 151

6.1 Example automaton $\mathcal{B}$ and matrices 257
6.2 Example multiplication of matrices 260

List of Tables

1.1 Basic simulation preorders 10
1.2 Generalized simulations 11
1.3 Contributions 12

3.1 Main contributions (gray area; ✓= GFQ, ×= not GFQ) 37

5.1 Multipebble simulations summary 122
5.2 Quotienting summary (✓= GFQ, ×= not GFQ) 196
5.3 Complexity summary 225

List of Examples

1.4.1 Quotienting under-approximates minimization 7

2.5.1 The inclusion between direct, delayed and fair simulation is strict . . 27
2.6.1 Fair (bi)simulation is not GFQ 30
2.6.2 Little-brothers are necessary for delayed simulation 32

3.3.1 Multipebble simulations "do not reach" containment 44
3.3.2 Delayed containment is not GFQ 45
3.4.1 Fixed-word quotients can achieve arbitrary compression ratios 49
3.4.2 Fixed-word simulations are strictly included in containment (I) . . . 51
3.4.3 Fixed-word simulations are strictly included in containment (II) . . . 52
3.4.4 Multipebble simulation is strictly included in fixed-word simulation . 53

4.1.1 Simultaneous forward/backward simulation quotienting is incorrect . 73
4.4.1 Visiting arbitrarily many accepting states is not sufficient 89
4.5.1 The transformer $\tau^{\mathrm{f}}(\succeq^{\mathrm{f}}, \sqsubseteq)$ is not reflexive 92
4.5.2 The fair transformer does not preserve jumping-safe preorders 94
4.6.1 The fair transformer τ^{f} is not idempotent 100
4.6.2 Simultaneous quotienting w.r.t. $\tau^{\mathrm{di+bw}}(Id)$ and $\tau^{\mathrm{de+bw}}(Id)$ is incorrect 103
4.6.3 Proxy simulation quotients achieve arbitrarily high compression ratios 104
4.6.4 Proxy quotients can outperform iterated quotienting 107
4.7.1 Backward proxy simulation quotients can outperform mediated quotients 111
4.7.2 Forward proxy simulation quotients can outperform mediated quotients 112
4.7.3 Mediated quotients can outperform proxy simulation quotients 113

5.3.1 Multipebble existential-direct simulation does not imply ω-inclusion 134
5.4.1 The multipebble hierarchy is strict 137
5.4.2 Multipebble simulation is not necessary for language inclusion . . . 144
5.6.1 $(1, k)$-simulation with $1 < k < n$ is not transitive 158
5.6.2 $(n, 1)$ and (n, n)-delayed simulation from [29] are not transitive . . . 175

5.7.1 Lemma 5.7.6 is tight 180
5.7.2 Lemma 5.7.7 is tight 182
5.7.3 The MH-construction preserves neither direct, nor delayed simulation 183
5.7.4 n pebbles are needed in Lemma 5.7.14 187
5.7.5 Fair simulation is needed in Lemma 5.7.14 188
5.8.1 Multipebble quotients can achieve arbitrary compression ratios . . . 194
5.8.2 No fixed representatives for existential states 205
5.8.3 Mixed classes need to be declared existential 206
5.8.4 Transitions from universal states in mixed classes are needed 207
5.8.5 $(1, k)$-delayed simulation is not GFQ for $1 < k < n$ 210
5.8.6 $(k, 1)$-delayed simulation is not GFQ for $k > 1$ 212

Chapter 1

Introduction

Contents

1.1 Overview **1**

1.2 Formal methods **2**

1.3 Model-checking **3**

1.4 Automata theory **4**

1.4.1 Language universality and inclusion 5

1.4.2 Automata minimization 5

1.4.3 Quotienting 6

1.5 Simulation preorders **8**

1.5.1 Intuition 8

1.5.2 Simulation preorders for Büchi automata 9

1.5.3 Generalizations 10

1.6 Contributions and thesis structure **11**

1.1 Overview

We can synthetically summarize the topic of this thesis as

quotienting automata w.r.t. generalized simulation equivalences.

In the rest of this chapter, we informally explain the meaning of the sentence above.

- In Section 1.2 and Section 1.3 we give motivations for the study of the theory of finite automata. We present the general area of *formal methods*, and the specific approach of *model-checking*.

- In Section 1.4 we introduce finite-state automata, and we define the problem of reducing their size.

- In Section 1.4.3 we explain quotienting, which is one specific method of reducing the size of automata.

- In Section 1.5 we introduce simulation equivalences, which provide a structured approach for quotienting automata.

- Finally, in Section 1.6 we present our contributions, i.e., generalized simulation equivalences for quotienting automata.

The presentation in this chapter is informal; formal definitions are deferred to Chapter 2 for background material, and to the main Chapters 3-5 for our novel contributions.

1.2 Formal methods

Computing systems are quickly becoming pervasive, ubiquitous and invisible (cf. *everyware* [60]), and their growth in complexity daunts our capacity of properly understanding their intricate behavior. The lack of a firm and rigorous mathematical foundation for software engineering results in the truly ubiquitous, "everyware" presence of bugs; in turn, this prevents the area from becoming a mature engineering discipline.

While in most applications it is possible to cope with system malfunctions (for example, by resetting the device), in *critical systems* errors are unacceptable, as they may result in tremendous financial losses, or, worst of all, human-life loss. We just mention a few notable examples of failure due to our lack of understanding of systems' behavior.

- *The Therac 25 radiotherapy machine* [84]. The Therac 25 was a machine for radiation therapy. In the period 1985-87, it is deemed responsible to six accidents (three of them fatal) due to a software failure resulting in wrong radiation dose.

- *The London ambulance service* [48]. In 1992, London adopted a computerized system to control the dispatching of ambulances. Overwhelmed with too many

real-time requests, the system inundated the human operators with exception messages, which prevented the dispatching of important information to the ambulances, until it finally crashed within one week since installation. As many as twenty deaths have been ascribed to the system's malfunctions.

- *The Intel Pentium FDIV bug* [64, 25, 121, 93, 30, 38]. The Intel P5 Pentium microprocessor contained a bug in his Floating Point Unit (FPU). The new and faster floating-point division algorithm required a look-up table to compute intermediate results, but some entries in this table were incorrect. Eventually, Intel had to replace all buggy chips, which resulted in an estimated loss of $500 millions. Intel later turned some of the faulty chips into key rings [94].

- *The Ariane 5 rocket* [57]. The Ariane 5 Flight 501 was launched on 4th June 1996, but a malfunction in the control software triggered self-destruction only 37 seconds after launch. The bug was caused by integer conversion overflow in a procedure that was not even needed after launch. This resulted in the loss of 10 years of work and $7 billions.

Therefore, we need to improve our understanding of critical systems in order to build them correctly. The wide discipline of *formal methods* provides an answer by giving a mathematical foundation to system construction.

1.3 Model-checking

In the formal approach to system construction, mathematically based languages, models and tools are used to specify, represent and verify such systems. First, one first develops a *model* of the system, which is a abstraction of its behavior with a precise, mathematical semantics. Second, one produces a formal *specification*, which is an assertion in a logical language (usually, in temporal logic [96]). Finally, one checks whether the model satisfies the specification. This can be done in several ways: Two notable approaches are theorem-proving and model-checking.

- In *interactive theorem-proving* (cf. [65]), both the model and the property are expressed as formulas in a deductive system consisting of axioms and inference rules. The user is assisted in finding a *proof* that the model satisfies the specification.

- In *model-checking* [26, 97], the model is specified as a state machine and the property as a logical formula. The state space of the model is explored to show conformance to the property. The method is *fully automated*, *complete*, and, when the property fails, it can generate informative *counter-examples*, explaining which behavior of the system violates the property.

Model-checking and theorem-proving have been successfully applied in many real-life applications, which include, but are not limited to [27]: avionics, large database systems, household electronic devices, electronic processors, medical devices, nuclear plants, security systems, telephone switching systems, transportation systems, hardware communication and coherence protocols, and anti-earthquake buildings. In this thesis, we develop methods that find their application in model-checking.

1.4 Automata theory

There are several approaches to model-checking. In the *automata-theoretic approach* [114], both the system and the specification are compiled into finite-state automata, and the model-checking problem is reduced to a language-inclusion problem between automata: The system is correct w.r.t. the specification iff every behavior of the system is allowed by the specification, that is, if the language of the system is included in that of the specification, and the former is a purely automata-theoretic problem. While finite-state automata [98, 16] are ordinarily considered as descriptors of *finite* objects[1], the kind of automata in use within formal methods recognize *infinite* objects. The adoption of automata on infinite objects stems from the need of modelling systems which have an on-going interaction with the environment, and which are not supposed to stop; therefore, their behavior is infinite. These are collectively called reactive systems [6]; examples include operating systems, communicating protocols and embedded systems. Moreover, the behavior of a reactive system can be modelled either as an infinite word or as an infinite tree, depending on whether we want to study its future time evolution as linear or branching, respectively (cf. [116]).

Here, we opt for the linear-time view. There are several finite-state models for language of infinite words. We concentrate on *Büchi automata*, which accept an infinite word if there exists a corresponding computation visiting a designated set of accepting states infinitely often. Historically, such automata had been introduced by R. Büchi in

[1] In [98], they are called automata over finite *tapes*, in analogy to Turing machines.

1962 (hence the name) as a tool to prove decidability of monadic second-order logic (MSO); his fundamental result states that languages recognized by Büchi's automata are (effectively) closed under complementation. Other models for infinite words include Muller automata, Rabin automata, Street automata, and parity automata (cf. [58]).

As already mentioned, Büchi automata are used for modelling both the system and the specification. If the specification is given as a formula in a logical language, then this formula needs to be "compiled" into a Büchi automaton before proceeding. For specifications in MSO, Büchi's proof entails that these can be translated into automata of non-elementary size (since each complementation step produces an exponentially larger automaton), and this blow-up cannot be avoided in general. For specifications in other temporal logics, the translation can be performed at a cheaper price; for example, for LTL, a single exponential blow-up suffices [118].

In this thesis, we concentrate on two important problems in the theory of Büchi automata: language inclusion and automata minimization.

1.4.1 Language universality and inclusion

The *language inclusion problem* asks whether the language of an automaton is included in the language of another automaton, and *language universality problem* is the subproblem of inclusion where the first automaton accepts every infinite word (see Section 2.4.3 for a formal definition). As we have seen, the model-checking problem can be reduced to a language inclusion problem between the system and the specification [114]. Also language universality has applications in formal verification; for example, it is used in sanity checking of temporal properties [103], where one wants to ensure that the specification automaton does not trivially accept every word (due to a design error).

From a computational complexity perspective, both universality and inclusion checking are PSPACE-complete problems [111], which means that no efficient algorithms are known, and it is unlikely that provably efficient algorithms will be developed in the future. Despite theoretical intractability, there has recently been a rekindled interest into finding algorithms which can solve problems of practical relevance [35, 51] (see also [122, 13, 36, 37, 5]).

1.4.2 Automata minimization

Another cornerstone automata-theoretic problem is minimization of automata. Its precise formulation depends on a notion of size of an automaton; one way of measuring

it is in terms of number of states. Smaller automata are easier to represent, manipulate and reason about.[2] *Automata minimization* asks to find an automaton with least number of states recognizing a given language. Minimal automata need not be unique in general. Like for inclusion checking, it is a computationally hard question: Minimality checking is already PSPACE-complete for finite-word automata [71] (while it is in polynomial time for deterministic automata [69]), and even approximating the minimal automaton retains the same high complexity [59]. While theoretical characterizations of minimal nondeterministic automata are known [74, 21, 9], algorithms working in cases of practical relevance are still missing.

For automata over infinite words, minimization is PSPACE-complete for nondeterministic Büchi automata. Recently, it has been shown to be NP-complete for deterministic Büchi automata [107]. (The complexity reduces to polynomial time for deterministic weak Büchi automata [85].) Despite these discouraging lower bounds, concrete algorithms have been proposed, both for deterministic [39] and nondeterministic Büchi automata [40]. However, exact minimization remains unfeasible for automata with more than a handful of states, and we need to content ourselves with approximate techniques. One such technique is called *quotienting*, which we introduce next.

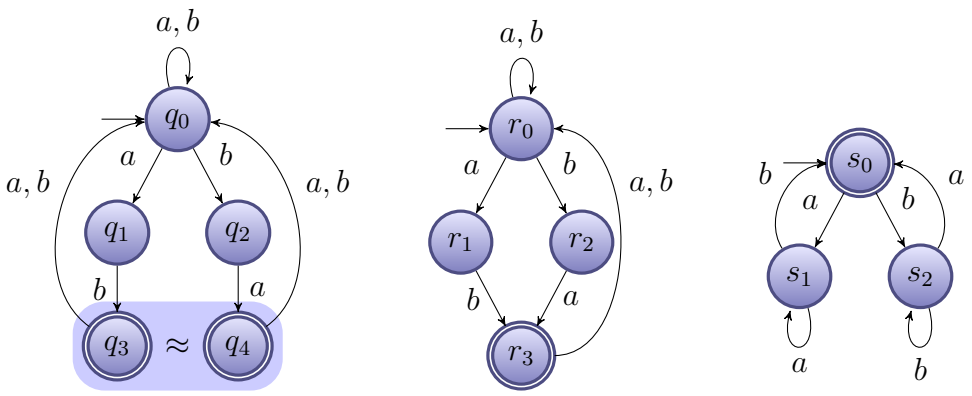
1.4.3 Quotienting

In this thesis, we study a particular approach to state-space reduction for Büchi automata, called quotienting. In *quotienting*, certain states of the automaton are identified and collapsed together, thus resulting in a reduction in size. Collapsible states are defined according to a suitable equivalence relation, which is designed to ensure that the quotienting operation is sound (i.e., language-preserving). If quotienting w.r.t. an equivalence preserves the language of the automaton, then we say that the equivalence is *Good for Quotienting* (GFQ).

In general, quotienting provides only an under-approximation to minimization, since the minimal automaton might require some intricate re-wiring of transitions not attainable by just merging states. This is shown in the example below.

[2] We shall note here that this point has been disputed in [108], where it is observed that smaller automata are not necessarily better in certain applications. Nonetheless, understanding automata minimization is still an important theoretical problem.

Example 1.4.1 - Quotienting under-approximates minimization



a) The original automaton $\mathcal{Q}$ b) The quotient automaton $\mathcal{R}$ c) The minimal automaton $\mathcal{S}$

Consider the nondeterministic Büchi automaton $\mathcal{Q}$ above, over the alphabet $\Sigma = \{a, b\}$. It has 5 states, and it recognizes the language L consisting of those infinite words where both letters a and b appear infinitely often. States q_3 and q_4 are equivalent and can be merged without changing the recognized language. Thus, $\mathcal{Q}$ is not quotient-minimal. By merging q_3 and q_4 into r_3, we obtain the 4-states quotient automaton $\mathcal{R}$, which recognizes L as well. In $\mathcal{R}$, no two states can be merged without changing the language. Therefore, $\mathcal{R}$ is quotient-minimal. However, the 3-states automaton $\mathcal{S}$ also recognizes L, but it has fewer states than $\mathcal{R}$. Therefore, $\mathcal{R}$ is not minimal, and quotienting under-approximates minimization.[3]

In spite of the fact that quotienting cannot attain exact minimization, we remark that optimal quotienting is PSPACE-complete, too. That is, it is PSPACE-complete to check whether there exists two mergeable states; a GFQ equivalence s.t. no coarser GFQ equivalence exists; intuitively, this holds since checking mergeability is essentially the same as an equivalence check (i.e., double inclusion), which is PSPACE-complete. However, to render quotienting practically feasible, one usually designs easy-to-compute equivalences which are guaranteed to be GFQ *by construction*, thus avoiding the expensive equivalence check. Thus, the quotienting problem is reduced to finding and computing suitable equivalence relations for automata. The most successful attempt in this direction is represented by equivalences derived from *simulation preorders*.

[3]Incidentally, $\mathcal{S}$ is minimal, since L cannot be recognized with less than 3 states.

1.5 Simulation preorders

Simulation preorder is a means for comparing the behavior of programs. While the informal intuition of step-wise "mimicry" has its origins in the early days of computer science, its formal definition in the context of programs came with [91], where an algorithm (or an approximation thereof) is actually *defined* to be the set of simulation-equivalent programs realizing it.

1.5.1 Intuition

The notion of simulation between programs $\mathcal{A}$ and $\mathcal{B}$ is best explained as a pebble game, where pebbles are put on states of the program. There are two players, called *Spoiler* and *Duplicator*[4], where the former controls a pebble in $\mathcal{A}$ and the latter a pebble in $\mathcal{B}$. The objective of Duplicator is to show that $\mathcal{B}$ simulates $\mathcal{A}$, while Spoiler has the opposite goal. They play in turns: At the beginning of each turn, Spoiler moves her pebble along a transition in $\mathcal{A}$. Then, Duplicator replies by moving her pebble along a corresponding transition in $\mathcal{B}$. If Duplicator's pebble gets stuck since no matching transition is available, then she loses and the game ends; otherwise, the game goes on forever and Duplicator wins. Then, $\mathcal{B}$ simulates $\mathcal{A}$ if, and only if, Duplicator always wins the simulation game. (In game-theoretic terms, this means that Duplicator has a strategy that is winning against every Spoiler's counter-strategy.)

Simulation desiderata for automata

The original notion of simulation is suitable to compare programs which either execute some visible action, or halt. In the context of automata, simulations need to additionally deal with acceptance conditions (and initial states). That is, in an automaton certain states are *accepting*, and simulations need to respect accepting states, in some sense. Simulations are expected to satisfy the following desiderata (see, e.g., [43]):

(Da) *Sufficient for language inclusion:* Simulation should imply language inclusion.

(Db) *Good for Quotienting (GFQ):* Quotienting w.r.t. simulation equivalence should preserve the language of the automaton.

(Dc) *Efficiently computable:* Simulation should be computable in deterministic polynomial time.

[4]These names come from Ehrenfeucht-Fraïssé games; [45] attributes the current terminology to [72].

(Dd) *Transitive:* Simulation should be transitive.

The transitivity requirement is not usually stated explicitly in the literature, since most simulation relations are transitive, and easily established so. However, neither of these two facts hold for certain simulations studied in this thesis (especially for *multipebble simulations*, see below), so we list transitivity as an additional desideratum.

Perhaps surprisingly, (Da) is not needed in all applications, e.g., for quotienting (Db). As an example, consider backward simulation, which is a GFQ preorder which do not imply language inclusion. Other examples are studied in this thesis.

1.5.2 Simulation preorders for Büchi automata

In the context of Büchi automata, *direct simulation* [34] has been the first attempt of an acceptance-sensitive notion of simulation. (They call it *BSR-aa*, for "accepting-accepting Büchi simulation relation".) The name indicates that it preserves accepting states step-wise: Whenever Spoiler's pebble is accepting (i.e., it is on an accepting state), Duplicator's pebble has to be accepting too. Direct simulation is a sound and efficient under-approximation of language inclusion [34]; moreover, it is also GFQ [10]. (Both properties also hold w.r.t. finite-word automata.)

In the context of Büchi automata, the exact time an accepting state is visited is not important, and direct simulation can be relaxed. Therefore, there have been several attempts at defining more liberal notions of refinement for comparing systems with acceptance constraints. This strand of research successfully identified the notion of *fair simulation* [66], where Duplicator has to match accepting states only in the limit, and not step-wise like in direct simulation: I.e., Duplicator is required to visit infinitely many accepting states only if Spoiler does so. Thus, fair simulation is coarser than direct simulation. Moreover, it still implies language inclusion between Büchi automata. Thus, it is natural to ask whether fair simulation is GFQ. This turns out to be false [67, 43] (cf. also Example 2.6.1).

Motivated by the quotienting problem, [43, 44] introduced *delayed simulation* as an intermediate notion between direct and fair simulation. Delayed simulation arises as a relaxation of direct simulation, where Duplicator can allow a *finite delay* to visit an accepting state after Spoiler has done so. Unlike fair simulation, delayed simulation is GFQ [43] (cf. also [20]).

Still motivated by state-space reduction, another variant of simulation that has been considered is *backward simulation* [99]. Backward simulation is similar to direct

Simulation	Language inclusion (Da)	GFQ (Db)	Efficiently computable (Dc)	Transitive (Dd)
Direct simulation	✓	✓	✓	✓
Delayed simulation	✓	✓	✓	✓
Fair simulation	✓	×	✓	✓
Backward simulation	×	✓	✓	✓

Table 1.1: Basic simulation preorders

simulation, except that pebbles are moved backwards, i.e., towards predecessors instead of successor states. Backward simulation is interesting since, while it does not imply language inclusion, it is GFQ [110] (where it is called "reverse simulation"). Moreover, backward simulation is in general incomparable with direct, delayed, or fair simulation The properties of the various simulations are summarized in Table 1.1.

1.5.3 Generalizations

Simulation preorders can be generalized in at least two perpendicular directions.

- *More general models of automata:* The ordinary notion of simulation can be extended to more general models of automata, like automata over trees, alternating automata, pushdown automata, etc.
- *Coarser simulation-like notions:* More general notions of simulation can be defined. This is usually obtained by modifying the rules of the simulation game; since one is usually interested in obtaining coarser notions, Duplicator is given "more power" in some form.

In this thesis, we consider generalizations of both kinds. In the literature, simulation for *alternating Büchi automata* (ABAs) [55], for *generalized Büchi automata* [73], and for *Büchi tree automata* [12] are instances of the first kind, while the *multipebble simulations* for nondeterministic Bchi automata (NBAs) of [42] is of the second kind. Simulations for ABAs and multipebble simulations play an important rôle in this thesis.

1.5.3.1 Simulations for alternating Büchi automata

Simulation preorders can be studied for more general models. One such model is obtained by considering alternation [22]. Simulations for *alternating Büchi automata*

Simulation	Language inclusion (Da)	GFQ (Db)	Efficiently computable (Dc)	Transitive (Dd)
Multipebble for NBAs	✓	✓*	×**	×***
Simulations for ABAs	✓	✓*	✓	✓

* except for the fair variant
** except for fixed number of pebbles
*** except for certain limit cases (1 pebble or high number of pebbles)

Table 1.2: Generalized simulations

(ABAs) have been studied in the fundamental work [55], where they show that simulations satisfying the four desiderata can be defined for alternating models; the most challenging point is to design suitable notions of quotient for alternating models.

1.5.3.2 Multipebble simulations

Another direction is to define more general simulation relations. *Multipebble simulations* have been introduced in the context of nondeterministic Büchi automata (NBAs) in [42], with the aim of obtaining coarser relations, at the expense of higher computational cost. The ordinary simulation game is generalized by allowing Duplicator, and only Duplicator, to use several pebbles. Duplicator can "hedge her bets" by splitting her pebbles to several successors during the game, in order to delay committing to particular decisions. Multipebble simulations can be defined in the direct, delayed and fair variant, in the spirit of [43]. They can be used for under-approximating language inclusion, and, for *fixed* number of pebbles, they are computable in polynomial time. Moreover, the direct and delayed variant can be used for quotienting automata (GFQ) [42]. In certain limit cases, they are also transitive.

Table 1.2 summarizes the properties of generalized simulations.

1.6 Contributions and thesis structure

In this thesis, we generalize the theory of simulation relations along several directions. These generalizations are motivated by tackling quotienting and language universality/inclusion problems.

In Chapter 2, we introduce notation and formal definitions for many notions used

Simulation	Ch.	Language inclusion (Da)	GFQ (Db)	Efficiently computable (Dc)	Transitive (Dd)
Fixed-word for NBAs	3	✓	✓*	×	✓
Jumping for NBAs	4	×	✓*	✓	✓
Multipebble for ABAs	5	✓	✓*	×**	×***

* except for the fair variant
** except for fixed number of pebbles
*** except for certain limit cases (1 pebble or high number of pebbles)

Table 1.3: Contributions

throughout the thesis, like automata, games and simulation preorders. Then, three main chapters follow, constituting the three main contributions of this thesis (cf. Table 1.3).

1. In Chapter 3, we introduce *fixed-word simulations* for NBAs, which generalize ordinary simulation. In fixed-word simulations we give more power to Duplicator by requiring Spoiler to declare the input word in advance; once the input is declared, both players have to play according to the prescribed symbols. Fixed-word simulations satisfy desiderata (Da), (Db) and (Dd), but they are not efficiently computable (PSPACE-complete) (Dc).

   We also consider a further generalization of fixed-word simulation, by giving Duplicator multiple pebbles. The main technical result of the chapter is Theorem 3.4.3, which states that multiple pebbles do not increase the power of Duplicator for fixed-word delayed simulation. In particular, this implies that fixed-word simulation subsumes multipebble simulation. The result is obtained via a non-trivial ranking construction.

2. In Chapter 4, we introduce *jumping simulations* for NBAs, which generalize simulations by allowing both players to "jump" during the game. Jumps are controlled by a binary relation given as input to the construction. Thus, jumping simulations act as a relation transformer. In the main result of the chapter we prove that jumping simulation transformers map GFQ preorders to GFQ preorders (cf. Theorems 4.5.3 and 4.5.4). Jumping and fixed-word simulations are incomparable.

3. In Chapter 5, we define *multipebble simulations for ABAs*, thus generalizing

both [42] and [55]. We establish that multipebble simulations do generalize from NBAs to ABAs in a satisfactory way, preserving all the expected properties.

In particular, direct and delayed multipebble simulations are GFQ even over ABAs; this requires a further tuning of the quotients defined in [55].

Finally, in Chapter 6, we draw some conclusions and we illustrate possible directions for further research. Moreover, we put simulation preorders in perspective, by presenting connections with exact (exponential) algorithms for Büchi automata universality and language-inclusion testing; in particular, we mention how simulation preorders have recently found applications in such optimized algorithms [2, 3].

Chapter 2

Background

Contents

2.1 Overview **16**

2.2 Basic notation **16**

2.3 Games **18**

2.4 Automata **20**

2.4.1 Nondeterministic Büchi automata 20

2.4.2 Alternating Büchi automata 21

2.4.3 Language inclusion and containment 22

2.4.4 Quotienting 23

2.5 Simulation preorders **25**

2.5.1 Forward simulations 25

2.5.2 Backward simulation 28

2.5.3 Complexity of simulations 30

2.6 Simulation-based automata simplification **30**

2.6.1 Quotienting 30

2.6.2 Transition simplification 31

2.6.3 Max quotients 33

2.1 Overview

In this chapter, we introduce notation and formal definitions for standard notions used throughout the thesis. In Section 2.2 we fix general notation, in Section 2.3 we introduce games, and in Section 2.4 we define nondeterministic and alternating Büchi automata, along with the language inclusion and quotienting problem. In Section 2.5, we present classical results on simulation preorders for Büchi automata. Finally, in Section 2.6 we present simulation-based techniques for quotienting and transition-simplification.

2.2 Basic notation

Let ω be the least infinite ordinal.

Iverson brackets [78] If E is a boolean condition, then $[E]$ is 0 if E is false and 1 otherwise.

Unordered cartesian product Fix a set X, and let $\mathcal{X} = \{X_0, X_1, \dots\} \subseteq 2^X$ be a collection of subsets of X. The unordered product $\bigotimes \mathcal{X}$ is defined as follows:

$$\bigotimes \mathcal{X} = \{\{x_0, x_1, \dots\} \mid x_0 \in X_0, x_1 \in X_1, \dots\}$$

That is, any $Y \in \bigotimes \mathcal{X}$ is obtained by fixing an element $x_i \in X_i$ from each set X_i. If any X_i is empty, then $\bigotimes \mathcal{X} = \emptyset$.

We now give an alternative formulation. A *choice function for* $\mathcal{X}$ is any function $f : \mathcal{X} \mapsto X$ s.t., given any set $X_i \in \mathcal{X}$, f selects a representative element $f(X_i) \in X_i$. If $\mathcal{X}' \subseteq \mathcal{X}$ is a sub-collection of $\mathcal{X}$ and f is a choice function for $\mathcal{X}$, then the *restriction of* f *to* $\mathcal{X}'$, written $f{\downarrow_{\mathcal{X}'}}$, is just f restricted to $\mathcal{X}'$.

There clearly is a bijection between elements in $\bigotimes \mathcal{X}$ and choice functions. In particular, any choice function f on $\mathcal{X}$ determines a set Y_f, which is the unique $Y_f \in \bigotimes \mathcal{X}$ s.t. $Y_f = \{f(X_i) \mid X_i \in \mathcal{X}\}$. We extend the restriction operation to sets: Given an element $Y_f \in \bigotimes \mathcal{X}$ and a sub-collection $\mathcal{X}' \subseteq \mathcal{X}$, the *restriction of* Y_f *to* $\mathcal{X}'$, written $Y_f{\downarrow_{\mathcal{X}'}}$, is defined as $Y_f{\downarrow_{\mathcal{X}'}} = Y_{f'} \in \bigotimes \mathcal{X}'$, where $f' = f{\downarrow_{\mathcal{X}'}}$ is the restriction of f to $\mathcal{X}'$. In other words,

$$Y_f{\downarrow_{\mathcal{X}'}} = \{f(X_i) \mid X_i \in \mathcal{X}'\}$$

Binary relations Fix sets A, B and C. A binary relation is a subset $R \subseteq A \times B$. The *transpose* of R (or the *reverse*), denoted $R^{-1} \subseteq B \times A$, is the binary relation $R^{-1} = \{(x, y) \mid (y, x) \in R\}$. For two binary relations $R_0 \subseteq A \times B$ and $R_1 \subseteq B \times C$, their composition, denoted $R_0 \circ R_1$, is defined as

$$R_0 \circ R_1 = \{(x, z) \mid \exists y \in B \ \cdot \ (x, y) \in R_0, (y, z) \in R_1\}$$

The identity on A, denoted $Id_A \subseteq A \times A$, is the binary relation $id_A = \{(x, x) \mid x \in A\}$. Consider a binary relation $R \subseteq A \times A$ on A. R is *reflexive* iff $id_A \subseteq R$, *transitive* iff $R \circ R \subseteq R$, and *symmetric* iff $R = R^{-1}$.

The *transitive and reflexive closure* of R, denoted R^*, is the smallest transitive and reflexive relation containing R. In other words, $R^* = \bigcup_{n \geq 0} R^n$, where $R^0 = Id_A$ and $R^{n+1} = R^n \circ R$. The *equivalence induced by* R, denoted $\approx_R$, is the largest equivalence contained in R^*, i.e., $\approx_R = R^* \cap (R^*)^{-1}$.

Subsets of bounded cardinality For a set Q, let $|Q|$ be its cardinality. For a finite set Q and an integer $k \geq 0$, let $2^{Q,k}$ be the set of subsets of Q of cardinality at most k, i.e.

$$2^{Q,k} = \{Q' \subseteq Q \mid |Q'| \leq k\}$$

For $Q' \in 2^{Q,k}$, we say that Q' is a k-subset of Q.

Finite and infinite sequences Fix a finite alphabet Σ. A finite sequence (or word) over Σ is a function from a (possibly empty) finite prefix of ω to Σ, while an infinite sequence (or word) is a function from ω to Σ. Let Σ^* and Σ^ω be the set of finite and infinite sequences, respectively. It is customary to denote sequences by listing its elements, e.g., $\pi = p_0 p_1 \cdots p_{k-1} \in \Sigma^*$ and $\pi' = p_0 p_1 \cdots \in \Sigma^\omega$. For a finite sequence $\pi = p_0 p_1 \cdots p_{k-1}$, let $|\pi| = k$ be its length, and let $\mathrm{last}(\pi) = e_{k-1}$ be its last element. If π is infinite, then $|\pi| = \omega$, and $\mathrm{last}(\pi)$ is undefined.

For two finite sequences $\pi_0 = p_0 p_1 \cdots p_{h-1}$ an $\pi_1 = q_0 q_1 \cdots q_{k-1}$, let their *concatenation* be the sequence $\pi_0 \cdot \pi_1 = p_0 p_1 \cdots p_{h-1} q_0 q_1 \cdots q_{k-1}$; a similar definition holds when π_1 is of length ω. We often denote concatenation just by juxtaposition $\pi_0 \pi_1$. Given two sequences of equal length $\pi_0 = p_0 p_1 \cdots p_{k-1}$ and $\pi_1 = q_0 q_1 \cdots q_{k-1}$, let $\pi_0 \times \pi_1 = \langle p_0, q_0 \rangle \langle p_1, q_1 \rangle \cdots \langle p_{h-1}, q_{k-1} \rangle$ be the sequence of the corresponding pairs; a similar definition can be given when both sequences are infinite.

2.3 Games

A *game* is a tuple $G = (P, P_0, P_1, p_I, \Gamma, \Gamma_0, \Gamma_1, W)$, where P is the set of positions, partitioned into disjoint sets P_0 and P_1, $p_I \in P_0$ is the initial position, $\Gamma = \Gamma_0 \cup \Gamma_1$ is the set of moves, where $\Gamma_0 \subseteq P_0 \times P_1$ and $\Gamma_1 \subseteq P_1 \times P_0$ are the set of moves of Player 0 and Player 1, respectively, and $W \subseteq P_0^\omega$ is the winning condition. A *dead end* is a position $p \in P_i$, for $i \in \{0, 1\}$, s.t. there is no $p' \in P_{1-i}$ with $(p, p') \in \Gamma_i$.

A *path* is a finite or infinite sequence of positions $\pi = p_0 p_1 \cdots \in (P_0 \cup P_1)^* \cup (P_0 \cup P_1)^\omega$, and it is a *valid path* iff it starts at $p_0 = p_I$, and, for all i s.t. $2i + 1 < |\pi|$, $(p_{2i}, p_{2i+1}) \in \Gamma_0$ and, if $2i + 2 < |\pi|$, then $(p_{2i+1}, p_{2i+2}) \in \Gamma_1$. Thus, a valid path is a sequence of alternating positions between the two players which is conform to the players' moves.

Partial plays and *plays* are finite and infinite valid paths, respectively. We assume that there are no dead ends in the game, therefore a partial play can be always extended to a longer one. A play is *winning* for Player 1 iff $p_0 p_2 p_4 \cdots \in W$; otherwise, is it winning for Player 0.

Strategies A *strategy* for Player 0 is a partial function $\sigma_0 : (P_0 P_1)^* P_0 \rightharpoonup P_1$ s.t., for any partial play $\pi \in (P_0 P_1)^* P_0$ ending in P_0, if σ_0 is defined on π and $\sigma(\pi) = p$, then πp is again a partial play. A play or partial play $\pi = p_0 p_1 \cdots$ is σ_0-*conform* iff, for every i s.t. $2i + 1 < |\pi|$, $p_{2i+1} = \sigma_0(p_0 p_1 \cdots p_{2i})$. Similarly, a strategy for Player 1 is a partial function $\sigma_1 : (P_0 P_1)^+ \rightharpoonup P_0$ s.t., for any partial play $\pi \in (P_0 P_1)^+$ ending in P_1, if σ_1 is defined on π and $\sigma_1(\pi) = p$, then πp is again a partial play. A play or partial play $\pi = p_0 p_1 \cdots$ is σ_1-*conform* iff, for every i s.t. $2i + 2 < |\pi|$, $p_{2i+2} = \sigma_1(p_0 p_1 \cdots p_{2i+1})$. While we do not require strategies to be total functions, we do require that strategies are defined on all conform partial plays.

A strategy σ_i is a *winning strategy* for Player i, for $i = 0, 1$, iff all σ_i-conform plays are winning for Player i. We say that Player i wins the game G if she has a winning strategy.

Determinacy A class of games is *determined* if, for every game in the class, at least one player has a winning strategy. A class of games is *zero-sum* if, for every game in the class, at most one player has a winning strategy. Games as defined here are zero-sum by definition. Moreover, for the very wide class of games where the winning set W is a *Borel set* (as it is in our case), a very general result ensures that they are determined

[88]. Since we only consider Borel winning objectives, all games presented in this thesis admit exactly one winner.

Reduced strategies Notice that, since the winning condition refers only to the projection of a play π onto its even positions, we can assume that strategies in fact do not depend on past odd positions (except for the current one). That is, if $\pi = p_0p_1p_2p_3p_4\cdots p_{2k-1}p_{2k}$ and $\pi' = p_0p'_1p_2p'_3p_4\cdots p'_{2k-1}p_{2k}$ are two plays differing only on odd positions, then $\sigma_0(\pi) = \sigma_0(\pi')$ and $\sigma_1(\pi \cdot p_{2k+1}) = \sigma_1(\pi' \cdot p_{2k+1})$ for any p_{2k+1}. This induces a natural projection operation on plays, that maps $\pi = p_0p_1p_2\cdots$ down to $p_0p_2p_4\cdots$. We call sequences of the latter form *reduced plays*. In practice, we will be mostly concerned with reduced plays, which are easier to manipulate. To do so, we define *reduced strategies* acting on reduced plays: A reduced strategy for Player 0 is a function $\sigma'_0 : P_0^+ \rightharpoonup P_1$, and a reduced strategy for Player 1 is a function $\sigma'_1 : P_0^+P_1 \rightharpoonup P_0$. A full strategy can be easily recovered from a reduced one, by letting $\sigma_0(p_0p_1\cdots p_{2k}) = \sigma'_0(p_0p_2\cdots p_{2k})$ and $\sigma_1(p_0p_1\cdots p_{2k-1}p_{2k}p_{2k+1}) = \sigma'_1(p_0p_2\cdots p_{2k-2}p_{2k}p_{2k+1})$. The notions of being conform and winning are easily lifted to reduced strategies. In the rest of this work, we just say strategy instead of reduced strategy.

For a set of positions $R \subseteq P_0$ and a strategy σ, we say that σ is *R-respecting* (or, *R-preserving*) if, for any σ-conform (partial) play $\pi = p_0p_2p_4\cdots$, we have $p_{2i} \in R$ for any $i < |\pi|$.

Finite-memory and memoryless strategies Sometimes we are interested in Player 1's strategies of a special form. Intuitively, a strategy has finite memory iff determining the next move requires only a finite amount of information about the partial play so far.

Definition 2.3.1. *A strategy $\sigma_1 : (P_0P_1)^+ \rightharpoonup P_0$ for Player 1 has* finite memory *iff there exists a finite pointed set (M, m), with $m \in M$, and two functions*

$$\mathsf{next} : P_0 \times P_1 \times M \rightharpoonup P_0, \text{ and}$$
$$\mathsf{up} : P_0 \times P_1 \times M \rightharpoonup M$$

s.t., for any partial play $\pi = p_0p_1\cdots p_{2i}p_{2i+1} \in (P_0P_1)^+$ in the domain of σ_1, $\sigma_1(\pi) = \mathsf{next}(p_{2i}, p_{2i+1}, m_i)$, where the sequence of memory states $m_0, m_1, \ldots, m_i$ is inductively defined as follows: $m_0 = m$, and, for $k \geq 0$, $m_{k+1} = \mathsf{up}(p_{2k}, p_{2k+1}, m_k)$. Sometimes we identify finite memory strategies and representation in terms of $\mathsf{next}, \mathsf{up}$.

σ_1 is *memoryless* if M is just a singleton; in this case we can discard the memory M, and we identify a memoryless strategy with the partial function $\sigma_1 : P_0 \times P_1 \rightharpoonup P_0$.

2.4 Automata

In this section we define nondeterministic Bchi automata, and we introduce the language inclusion and quotienting problem. We also define alternating Büchi automata, but their study is deferred to Chapter 5.

2.4.1 Nondeterministic Büchi automata

A *nondeterministic Büchi automaton* (NBA) is a tuple $\mathcal{Q} = (Q, \Sigma, I, \Delta, F)$, where Q is a finite set of states, Σ is a finite alphabet, $I \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of accepting states and $\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation. We also write both $q \xrightarrow{a} q'$ and $q' \xleftarrow{a} q$ instead of $(q, a, q') \in \Delta$, and $q \longrightarrow q'$ when $\exists a \in \Sigma \cdot q \xrightarrow{a} q'$. For a set of states $\mathbf{q}' \subseteq Q$, let $\mathrm{Pre}(\mathbf{q}') = \{q \mid q \longrightarrow q'\}$. We say that $\mathcal{Q}$ is *complete* iff, for any state $q \in Q$ and input symbol $a \in \Sigma$, there exists a successor $q' \in Q$ s.t. $q \xrightarrow{a} q'$, and that $\mathcal{Q}$ is *deterministic* iff q' is unique (when it exists). We say that a state q is accepting if $q \in F$; let $[q \in F] = 1$ if q is accepting, and 0 otherwise.

For a finite or infinite sequence of states $\rho = q_0 q_1 \cdots$ and an index $i \leq |\rho|$, let $\#F(\rho, i)$ be the number of accepting states occurring in ρ up to (and including) the $(i-1)$-th element. Formally, $\#F(\rho, i) = \sum_{0 \leq k < i} [q_k \in F]$, with $\#F(\rho, 0) = 0$. Let $\#F(\rho) = \#F(\rho, |\rho|)$; if ρ is infinite, then $\#F(\rho) = \omega$ iff ρ contains infinitely many accepting states.

Fix a finite word $w = a_0 a_1 \cdots a_{n-1} \in \Sigma^*$. A *path* π *over* w is a sequence of transitions $q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{n-1}} q_n$ of length n. When such a path exists, we also write $q_0 \xrightarrow{w} q_n$, and if a path exists passing through an accepting state $q_j \in F$, $0 \leq j \leq n$, then we write $q_0 \xrightarrow{w}_F q_n$.

Paths over infinite words are defined similarly. π is *initial* if it starts in an initial state $q_0 \in I$ and it is *fair* if $\#F(\pi) = \omega$ (which implies that the path is infinite). For a state $q \in Q$, its *language* $\mathcal{L}(q)$ is the set of infinite words admitting a fair path from q:

$$\mathcal{L}(q) = \{w \in \Sigma^\omega \mid \text{ there exists a fair path } \pi \text{ over } w \text{ from } q\} ,$$

and the language of an automaton $\mathcal{Q}$ is the set of words admitting an initial and fair path, i.e., $\mathcal{L}(\mathcal{Q}) = \bigcup_{q_I \in I} \mathcal{L}(q_I)$.

Multipaths When dealing with multipebble simulations, we manipulate the following generalization of transitions and paths.

Fix an automaton $\mathcal{Q} = (Q, \Sigma, I, \Delta, F)$. For two sets of states $\mathbf{q}, \mathbf{q}' \subseteq Q$ and an input symbol $a \in \Sigma$, let $\mathbf{q} \overset{a}{\Longrightarrow} \mathbf{q}'$ iff every state in $\mathbf{q}'$ is an a-successor of some state in $\mathbf{q}$. Formally[1], $\mathbf{q} \overset{a}{\Longrightarrow} \mathbf{q}'$ holds iff $\forall q' \in \mathbf{q}' \cdot \exists q \in \mathbf{q} \cdot q \xrightarrow{a} q'$. We call $\mathbf{q} \overset{a}{\Longrightarrow} \mathbf{q}'$ a *multi-transition*; we are interested in sequences of multi-transitions.

Definition 2.4.1 (Multipath)**.** *A* multipath *is a finite or infinite sequence* $\pi = \mathbf{s}_0 \overset{a_0}{\Longrightarrow} \mathbf{s}_1 \overset{a_1}{\Longrightarrow} \cdots$*, where, for all* $i < |\pi|$*,* $a_i \in \Sigma$*,* $\mathbf{s}_i \neq \emptyset$*, and* $\mathbf{s}_i \subseteq Q$*.*

Multipaths consisting solely of singletons are just ordinary paths.

Definition 2.4.2 (Submultipath)**.** *For two multipaths* $\pi = \mathbf{s}_0 \overset{a_0}{\Longrightarrow} \mathbf{s}_1 \overset{a_1}{\Longrightarrow} \cdots$ *and* $\pi' = \mathbf{s}'_0 \overset{a_0}{\Longrightarrow} \mathbf{s}'_1 \overset{a_1}{\Longrightarrow} \cdots$ *over the same finite or infinite word* $w = a_0 a_1 \cdots$ *(thence, of the same length),* π' *is a* submultipath of π*, written* $\pi' \subseteq \pi$*, iff, for any* $i < |\pi|$*,* $\mathbf{s}'_i \subseteq \mathbf{s}_i$*.*

For an ordinary path $\pi' = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots$ *and a multipath* π *as above, let* $\pi' \in \pi$ *iff, for any* $i < |\pi'|$*,* $s'_i \in \mathbf{s}_i$*.*

The following is an immediate property of submultipaths.

Lemma 2.4.3. *If* $\pi = \mathbf{s}_0 \overset{a_0}{\Longrightarrow} \mathbf{s}_1 \overset{a_1}{\Longrightarrow} \cdots \overset{a_{k-1}}{\Longrightarrow} \mathbf{s}_k$ *is a multipath, then, for any nonempty* $\mathbf{s}'_k \subseteq \mathbf{s}_k$*, there exists a submultipath* $\pi' \subseteq \pi$ *ending in* $\mathbf{s}'_k$*:*

$$\pi' = \mathbf{s}'_0 \overset{a_0}{\Longrightarrow} \mathbf{s}'_1 \overset{a_1}{\Longrightarrow} \cdots \overset{a_{k-1}}{\Longrightarrow} \mathbf{s}'_k$$

2.4.2 Alternating Büchi automata

An *alternating Büchi automaton* (ABA) $\mathcal{Q}$ is a tuple $(Q, \Sigma, q_I, \Delta, E, U, F)$, where Q is a finite set of states, Σ is a finite alphabet, q_I is the initial state, $\{E, U\}$ is a partition of Q into *existential* and *universal* states, $\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation and $F \subseteq Q$ is the set of accepting states. Like for NBAs, we assume that the transition relation is complete.

An ABA $\mathcal{Q}$ recognizes a language of infinite words $\mathcal{L}(\mathcal{Q}) \subseteq \Sigma^\omega$. The acceptance condition is best described in a game-theoretic way [61]. Given an input word $w = a_0 a_1 \cdots \in \Sigma^\omega$, the *acceptance game* $\mathbb{G}^\omega(\mathcal{Q}, w)$ is played by two players, Pathfinder and Automaton, where Pathfinder plays as Player 0 and Automaton as Player 1. Positions of the game are of the form $\langle q, k \rangle \in P = P_{\text{even}} \cup P_{\text{odd}}$, for a state $q \in Q$ and an integer

[1] This kind of backward-compatible transition had already appeared in [86].

$k \geq 0$, where Pathfinder owns configurations in P_{even} with k even, and Automaton owns configurations in P_{odd} with k odd. Intuitively, existential states are controlled by Automaton, while Pathfinder controls universal states; when a player plays on a state she owns but she does not control, nothing happens. Formally, the initial configuration is $\langle q_I, 0\rangle$, and, if the current configuration is $\langle q, h\rangle$, then the next configuration is determined as follows, depending on whether h is even or odd:[2]

- $h = 2k$ even:
  - If $q \in E$, then nothing happens and the next configuration is just $\langle q, 2k+1\rangle$.
  - If $q \in U$, then Pathfinder selects a successor $q' \in \Delta(q, a_k)$, and the next configuration is $\langle q', 2k+1\rangle$.
- $h = 2k+1$ odd:
  - If $q \in E$, then Automaton selects a successor $q' \in \Delta(q, a_k)$, and the next configuration is $\langle q', 2k+2\rangle$.
  - If $q \in U$, then nothing happens and the next configuration is just $\langle q, 2k+2\rangle$.

A play of the game is an infinite sequence of configurations $\langle q_0, 0\rangle\langle q_1, 1\rangle \cdots$ (with $q_0 = q_I$), and Automaton wins this play iff $q_i \in F$ for infinitely many i's. Thus, the winning set $W = (P^*(F \times \omega))^\omega$ is Borel; consequently, the acceptance game is determined [88]. The language recognized by $\mathcal{Q}$ is

$$\mathcal{L}(\mathcal{Q}) = \{w \in \Sigma^\omega \mid \text{ Automaton wins } \mathbb{G}^\omega(\mathcal{Q}, w)\}$$

Nondeterministic Büchi automata are just ABAs with no universal states, i.e., with $U = \emptyset$ (all choices are existential, i.e., of Automaton); dually, a universal Büchi automaton (UBA) is an ABA with $E = \emptyset$, where all choices are universal, i.e., of Pathfinder. When drawing pictures, we represent existential states with circles

and universal states with squares

.

2.4.3 Language inclusion and containment

For two automata $\mathcal{A}$ and $\mathcal{B}$, nondeterministic or alternating, the *language inclusion problem* asks whether $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$, which we abbreviate as $\mathcal{A} \subseteq \mathcal{B}$. The *language*

[2]The need of introducing even and odd positions comes from the fact that our games are strictly alternating, while in an ABA not necessarily existential and universal states alternate along a path. On the other side, strictly alternating games are more natural when defining simulation games.

universality problem is the subproblem where $\mathcal{L}(\mathcal{A}) = \Sigma^\omega$. Both language inclusion and universality of Büchi automata are PSPACE-complete problems (see, e.g., [80]), like in the finite-word case.

We also consider the language inclusion problem between states, which we call the *containment problem*. For two states q_0 and s_0, q_0 is contained in s_0, written $q_0 \subseteq s_0$, iff $\mathcal{L}(q_0) \subseteq \mathcal{L}(s_0)$. Clearly, inclusion and containment are inter-reducible in PTIME; consequently, also containment is PSPACE-complete.

Game-theoretic interpretation We give an alternative presentation of language containment, in the special case of nondeterministic automata. This is useful when contrasting it with simulation preorders (introduced in Section 2.5).

Language containment/inclusion can be seen as a game between two players, called Spoiler and Duplicator, where the latter wants to prove $q_0 \subseteq s_0$, and the former wants to disprove it. The game is played in only one round:

- Spoiler chooses a word $w = a_0 a_1 \cdots \in \Sigma^\omega$ and a w-path

$$\pi_0 = q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \cdots$$

- Duplicator replies by choosing a w-path

$$\pi_1 = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots$$

Duplicator wins this play of the game iff the following condition holds:

If π_0 is fair, then π_1 is fair.

Clearly, Duplicator has a winning strategy in the containment game iff $q_0 \subseteq s_0$.

2.4.4 Quotienting

In this section, we introduce the quotienting problem for nondeterministic Büchi automata. Quotienting for alternating automata is more complex; cf. Section 5.8.

Let $\mathcal{Q} = (Q, \Sigma, I, \Delta, F)$ be an NBA, and let $\approx$ be an equivalence relation on Q. Let the function $[\cdot]_\approx : Q \mapsto 2^Q$ map each element $q \in Q$ to the equivalence class $[q]_\approx \subseteq Q$ q belongs to; that is, $[q]_\approx := \{\widehat{q} \in Q \mid q \approx \widehat{q}\}$. We overload $[P]_\approx$ on sets $P \subseteq Q$ by taking the set of equivalence classes: $[P]_\approx = \{[q]_\approx \in 2^Q \mid q \in P\}$. When the equivalence relation is clear from the context, we omit it and we just write $[q]$, $[P]$.

The *naïve quotient* $\mathcal{Q}_\approx$ of $\mathcal{Q}$ w.r.t. $\approx$ is the automaton $\mathcal{Q}_\approx = (Q_\approx, \Sigma, I_\approx, \Delta_\approx, F_\approx)$ over the same alphabet Σ, where

- $Q_\approx = [Q]_\approx$: States are equivalence classes of states from $\mathcal{Q}$.
- $I_\approx = [I]_\approx$: Initial states are those classes containing some initial state.
- $F_\approx = [F]_\approx$: Final states are those classes containing some final state.
- Transitions are as follows: For states $q, q' \in Q$ and input symbol $a \in \Sigma$,

$$([q], a, [q']) \in \Delta_\approx \quad \text{iff} \quad (q, a, q') \in \Delta$$

This is called a *naïve* quotient since initial states, final states and transitions are induced representative-wise. In particular, each member of an equivalence class potentially contributes to the transition relation.

Remark 2.4.4. In [19], naïve quotients are called *existential quotients*, in contrast with *universal quotients*, where transitions are induced only if *every* member in an equivalence class contributes to it: I.e., in an universal quotient transitions are induced as follows:

$$([q], a, [q']) \in \Delta_\approx \quad \text{iff} \quad \forall(\widehat{q} \in [q]) \cdot \exists(\widehat{q}' \in [q']) \cdot (\widehat{q}, a, \widehat{q}') \in \Delta$$

Clearly, universal quotients have at most as many transitions as existential quotients (and possibly less), and thus they are preferable. We compare them in more detail in Section 2.6.2, where we show that, in the context of simulation-based quotienting, by deleting certain transitions in Q, existential and universal quotients actually coincide (see Remark 2.6.12).

Remark 2.4.5. Notice that the quotient of a deterministic automaton is also a deterministic automaton. Indeed, if two deterministic states are simulation equivalent, then, for every a, also their a-successors are simulation equivalent.

When quotienting, we are interested in equivalences preserving the language of the automaton. They are called Good for Quotienting equivalences; the design of Good for Quotienting equivalences is the central topic of this thesis.

Definition 2.4.6 (Good for Quotienting equivalences)**.** *We say that* $\approx$ *is* good for quotienting *(GFQ) if quotienting* $\mathcal{Q}$ *w.r.t.* $\approx$ *preserves the language:*

$$\approx \text{ is GFQ} \quad \textit{iff} \quad \mathcal{L}(\mathcal{Q}) = \mathcal{L}(\mathcal{Q}_\approx)$$

Sometimes we extend the notion of quotienting and being GFQ to arbitrary binary relations $R \subseteq Q \times Q$, by applying it to the induced equivalence $\approx_R = R^* \cap (R^*)^{-1}$.

The following is an immediate property of naïve quotients for nondeterministic models. It says that coarser relations give rise to quotients recognizing larger languages.

Lemma 2.4.7. *For two equivalences $\approx_0$ and $\approx_1$ on Q, if $\approx_0 \subseteq \approx_1$, then $\mathcal{L}(\mathcal{Q}_{\approx_0}) \subseteq \mathcal{L}(\mathcal{Q}_{\approx_1})$. In particular, by letting $\approx_0$ be the identity, $\mathcal{L}(\mathcal{Q}) \subseteq \mathcal{L}(\mathcal{Q}_{\approx_1})$.*

Intuitively, this holds since a path in $Q_{\approx_0}$ immediately induces a corresponding path in $Q_{\approx_1}$, and initial/accepting states are preserved. Thus, quotienting w.r.t. finer relations (partially quotienting), preserves correctness.

Corollary 2.4.8. *Let $\approx_0$ and $\approx_1$ be two equivalences, with $\approx_1$ coarser than $\approx_0$. If $\approx_1$ is GFQ, then $\approx_0$ is GFQ.*

Lemma 2.4.7 and Corollary 2.4.8 fail for alternating models, as we remark in Section 5.8.

2.5 Simulation preorders

In this section, we present classic results on simulation preorders for nondeterministic Büchi automata. Analogous results for alternating automata are presented in Section 5.2.

2.5.1 Forward simulations

In automata theory, nondeterministic machines are usually compared with respect to the language they describe, e.g., language inclusion/containment. In simulation, instead, the internal behavior of automata is exposed, and the sequence of transitions an automaton performs for accepting a word becomes relevant. Thus, simulation preorders can be seen as a refinement of containment which take into account the implementation details of the automaton, therefore allowing us to compare automata at a finer granularity.

Like language inclusion, also simulation can be described game-theoretically, with the difference that the players do not build infinite paths in only one round, but they interleave their moves as to build them in an incremental fashion.

Fix a nondeterministic automaton $\mathcal{Q}$ and two distinguished states q_0 and s_0 therein. In the simulation game between q_0 and s_0, Duplicator and Spoiler play in rounds, where, at each round, the two players extend by one more transition a finite path from the respective starting state. Formally, Spoiler's set of positions is $P_{\text{Sp}} := Q \times Q$, and Duplicator's set of positions is $P_{\text{Dup}} := Q \times Q \times \Sigma \times Q$. The interaction of the two players is best described with the metaphor of *pebbles*. There are two pebbles, one controlled by Spoiler and one controlled by Duplicator. In the initial configuration $\langle q_0, s_0 \rangle$, Spoiler's pebble is on q_0, and Duplicator's one is on s_0. At any round of the game, if Spoiler's pebble is on q_i and Duplicator's one is on s_i, then

- Spoiler chooses an input symbol $a_i \in \Sigma$, and she moves her pebble from q_i to an a_i-successor q_{i+1}, thus selecting a transition

$$q_i \xrightarrow{a_i} q_{i+1}$$

  This determines a move in the game $\langle (q_i, s_i), (q_i, s_i, a_i, q_{i+1}) \rangle \in \Gamma^{\mathrm{Sp}}$.

- Duplicator replies by moving her pebble from s_i to some a_i-successor s_{i+1}, thus selecting a transition

$$s_i \xrightarrow{a_i} s_{i+1}$$

  This determines a move in the game $\langle (q_i, s_i, a_i, q_{i+1}), (q_{i+1}, s_{i+1}) \rangle \in \Gamma^{\mathrm{Dup}}$.

Remark 2.5.1. Technically, we are describing an *autosimulation game*, since states q_0 and s_0 belong to the same automaton. Simulations between different automata $\mathcal{A}$ and $\mathcal{B}$ can be described as autosimulations in the disjoint union. Therefore, we do not distinguish between simulations *within* an automaton or *between* automata.

Remark 2.5.2 (Bisimulation equivalence). The related notion of *bisimulation equivalence* is obtained from the game above by allowing Spoiler to choose, at each round, whether to play a transition on the left or on the right. Bisimulation equivalence is finer than simulation equivalence (and, therefore, simulation preorder), thus it induces larger quotient automata. We do not consider bisimulation equivalences in this thesis.

If at any round Duplicator cannot reply to Spoiler's move, then she loses. Otherwise, the two players can keep playing for ω rounds (until "doomsday", to paraphrase [17]), at the end of which two infinite paths $\pi_0 = q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \cdots$ and $\pi_1 = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots$ are built, as in language containment. Let $\pi = \pi_0 \times \pi_1$ be the resulting play of the game. Then, the winner depends on the specific kind of simulation we are considering. In the literature, the notions of *direct*, *delayed* and *fair* simulation have been considered. For $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$, let W^x be the corresponding winning set, which is defined as follows.

1. *Direct simulation* [34]: This is the counterpart to containment over *finite words*. Duplicator has to visit accepting states at the same time when Spoiler does so:

$$\pi \in W^{\mathrm{di}} \iff \forall i \cdot (q_i \in F \implies s_i \in F)$$

2. *Fair simulation* [66]: This is the counterpart to language containment over infinite words. Duplicator has to match accepting states only in the limit:

$$\pi \in W^{\mathrm{f}} \iff (\pi_0 \text{ fair } \implies \pi_1 \text{ fair })$$

3. *Delayed simulation* [43]: This winning condition is intermediate between direct and fair simulation. Duplicator has to visit accepting states with at most a finite delay since Spoiler has done so:

$$\pi \in W^{\mathrm{de}} \iff \forall i \cdot (q_i \in F \implies \exists j \geq i \cdot s_j \in F)$$

If Duplicator has a winning strategy in the x-simulation game, then we say that q_0 *x-simulates* s_0, written $q_0 \sqsubseteq^x s_0$.

Remark 2.5.3. If the automaton is deterministic, then fair simulation and language containment coincide, since the choice of the input word uniquely determines a path.

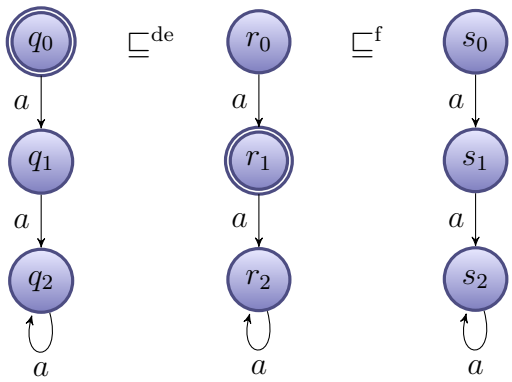
It follows immediately from the definition that direct, delayed and fair simulation are naturally ordered by inclusion.

Lemma 2.5.4. *For any NBA $\mathcal{Q}$ and $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$, let $\sqsubseteq^x$ be x-simulation. Then,*

$$\sqsubseteq^{\mathrm{di}} \subseteq \sqsubseteq^{\mathrm{de}} \subseteq \sqsubseteq^{\mathrm{f}}$$

The inclusions are strict by the following example.

Example 2.5.1 - The inclusion between direct, delayed and fair simulation is strict



We clearly have

$$q_0 \sqsubseteq^{\mathrm{de}} r_0 \sqsubseteq^{\mathrm{f}} s_0$$

Indeed, r_0 delayed-simulates q_0 since Duplicator is accepting on r_1, and s_0 trivially fair-simulates r_0 since Spoiler cannot build a fair path from r_0. However,

$$q_0 \not\sqsubseteq^{\mathrm{di}} r_0 \not\sqsubseteq^{\mathrm{de}} s_0$$

Simulation game for $\sqsubseteq^x$, with $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$

Current configuration $\langle q_i, s_i \rangle$ - - - - - - → Next configuration $\langle q_{i+1}, s_{i+1} \rangle$

Spoiler chooses $q_i \xrightarrow{a_i} q_{i+1}$

Duplicator replies with $s_i \xrightarrow{a_i} s_{i+1}$

Winning condition

$x = \mathrm{di}: \quad \forall (i \geq 0) \cdot q_i \in F \implies s_i \in F$

$x = \mathrm{de}: \quad \forall (i \geq 0) \cdot q_i \in F \implies \exists (j \geq i) \cdot s_j \in F$

$x = \mathrm{f}: \quad \pi_0 \text{ fair} \implies \pi_1 \text{ fair}$

2.5.2 Backward simulation

Another notion of simulation has been considered in the literature, namely, *backward simulation*, which is just a simulation for the reverse automaton (where all transitions are reversed). The definition is as in forward simulations, except that Spoiler and Duplicator select "backward transitions": From configuration $\langle q_i, s_i \rangle$, Spoiler plays some transition $q_i \xleftarrow{a_i} q_{i+1}$, and Duplicator replies with some other transition $s_i \xleftarrow{a_i} s_{i+1}$, and the game goes to configuration $\langle q_{i+1}, s_{i+1} \rangle$. The winning condition is as follows:

4. *Backward simulation* [99, 110]: Like in direct simulation, Duplicator has to match final states immediately; additionally, the same is required for initial states:

$$\pi \in W^{\mathrm{bw}} \iff \forall i \cdot (q_i \in F \implies s_i \in F) \wedge (q_i \in I \implies s_i \in I)$$

Let $\sqsubseteq^{\mathrm{bw}}$ be the induced backward simulation relation.

Simulation game for $\sqsubseteq^{\mathrm{bw}}$

Current configuration $\langle q_i, s_i \rangle$ $\dashrightarrow$ Next configuration $\langle q_{i+1}, s_{i+1} \rangle$

Spoiler chooses $q_i \xleftarrow{a_i} q_{i+1}$

Duplicator replies with $s_i \xleftarrow{a_i} s_{i+1}$

Winning condition

$$1)\ \forall (i \geq 0)\ \cdot\ q_i \in F \implies s_i \in F$$
$$2)\ \forall (i \geq 0)\ \cdot\ q_i \in I \implies s_i \in I$$

Simulation desiderata

It is a classic result that forward simulations satisfy desiderata (Da), (Dc), and (Dd). Backward simulation fails (Da), since its backward nature is incompatible with language containment between *states*; nonetheless, backward simulation can be used in a more sophisticated test for language inclusion between *automata* (instead of single states; cf. Chapter 4). Quotienting (Db) is dealt with in the next section.

Theorem 2.5.5 (Simulation desiderata [34, 66, 43, 110])**.** *For any NBA $\mathcal{Q}$ and for $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}, \mathrm{bw}\}$,*

- $\sqsubseteq^x$, $x \neq \mathrm{bw}$, *implies language inclusion (Da).*
- $\sqsubseteq^x$ *can be computed in deterministic polynomial time (Dc).*
- $\sqsubseteq^x$ *is transitive (Dd).*

Proofs of these facts are given in Chapter 4, in the more general context of jumping simulations.

2.5.3 Complexity of simulations

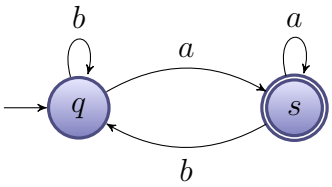
In the thesis we analyze the complexity of generalized simulations, and we give upper and lower bounds for computing them. When we speak about the complexity of a simulation R, we take the view of R as a mere set of pairs, and the associated decision problem is just membership in R. (Of course, computing a simulation relation R and checking membership in R are PTIME inter-reducible problems.)

2.6 Simulation-based automata simplification

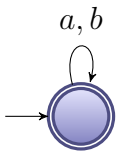
2.6.1 Quotienting

We have already mentioned in the Section 1.5.2 that fair simulation is not GFQ. This is shown in the example below. (Incidentally, since the automaton in the example is deterministic, this also shows that fair *bi*simulation is not GFQ [67].)

Example 2.6.1 - Fair (bi)simulation is not GFQ



a) The original automaton $\mathcal{Q}$



b) The quotient automaton $\mathcal{Q}_{\approx}$

Consider the deterministic automaton $\mathcal{Q}$ on the left above. States q and s both recognize the language $\mathcal{L}(q) = \mathcal{L}(s) = (\Sigma^* a)^\omega$ consisting of words with infinitely many a's. Since the automaton is deterministic, simulation and containment coincide, thus q and s are fair simulation equivalent. However, identifying together q and s is incorrect as the resulting automaton $\mathcal{Q}_{\approx}$ on the right above would recognize Σ^ω.

Remark 2.6.1. Although fair simulation quotienting is not guaranteed to preserve the language of the automaton, it has been shown that fair simulation can be nonetheless used in a non-trivial state-space reduction algorithm [62]. We do not explore this possibility further in this thesis.

However, the finer direct and delayed simulation are GFQ, as well as (the incomparable) backward simulation.

Theorem 2.6.2 (cf. [10, 43, 110])**.** *For $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{bw}\}$, x-simulation equivalence is GFQ (Db).*

In fact, simulation-based quotienting has the stronger property that the quotient automaton not only has the same language as the original automaton, but it is even simulation-equivalent to it [19, 110, 55].

Lemma 2.6.3. *For $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{bw}\}$ and a state q in $\mathcal{Q}$ with quotient class $[q]$ in $\mathcal{Q}_{\approx^x}$,*

$$q \approx^x [q]$$

Consequently, simulation preorder on $\mathcal{Q}$ completely characterizes simulation preorder on the quotient automaton $\mathcal{Q}_{\approx}$. In particular, it is not necessary to recompute simulation preorder on $\mathcal{Q}_{\approx}$. Moreover, simulation preorder on $\mathcal{Q}_{\approx}$ is actually a *partial order*, i.e., it is antisymmetric.

Lemma 2.6.4. *For $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{bw}\}$ and for two states q and s in $\mathcal{Q}$, with quotient classes $[q]$ and $[s]$ in $\mathcal{Q}_{\approx}$, respectively,*

$$q \sqsubseteq^x s \quad \textit{iff} \quad [q] \sqsubseteq^x [s]$$

Consequently, $\sqsubseteq^x$ is a partial order on $\mathcal{Q}_{\approx}$, i.e.,

$$[q] \sqsubseteq^x [s] \ \textit{and} \ [s] \sqsubseteq^x [q] \quad \textit{implies} \quad [q] = [s]$$

Proof. Immediate since $q \approx^x [q]$ and $s \approx^x [s]$ by Lemma 2.6.3 (and transitivity). □

2.6.2 Transition simplification

Sometimes it is possible to simplify the transition structure of the automaton if a simulation is given. The idea is that not every successor of a state is necessary, and it suffices to keep only certain successors. Fix an automaton $\mathcal{Q}$, together with a preorder $\preceq \subseteq Q \times Q$. We introduce the notion of maximal successor.

Definition 2.6.5 (Maximal successors)**.** *For a symbol $a \in \Sigma$ and states q and q', q'* is a $\preceq$-maximal a-successor of q *iff $q \xrightarrow{a} q'$ and, for any other $q'' \in Q$ s.t. $q \xrightarrow{a} q''$, if $q' \preceq q''$, then $q'' \preceq q'$. Let $\max_a^{\preceq}(q)$ be the set of $\preceq$-maximal a-successors of q.*

In other words, maximal a-successors are those a-successors with no strictly $\preceq$-larger a-successor. Depending on $\preceq$, the language of $\mathcal{Q}$ does not change if we prune away transitions to non-maximal successors. Formally, we let $Q^{\preceq}$ be any automaton

which is the same as $\mathcal{Q}$, except that $q \xrightarrow{a} q'$ is in $\mathcal{Q}^{\preceq}$ only if $q' \in \max_a^{\preceq}(q)$. ($Q^{\preceq}$ is not uniquely determined since there might be several maximal successors; correctness holds for all such $Q^{\preceq}$'s.) This operation is sound for direct simulation.

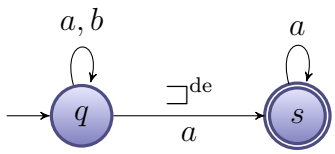
Lemma 2.6.6. *For* $\preceq = \sqsubseteq^{\mathrm{di}}$, $\mathcal{L}(\mathcal{Q}) = \mathcal{L}(\mathcal{Q}^{\preceq})$.

Remark 2.6.7. Non-maximal successors are called *little-brothers* in [19]. Sometimes we follow this terminology, and we say that, if transitions to non-maximal successors are removed, then there are no little-brothers.

Remark 2.6.8. It is possible to define also the set of maximal predecessors, and eliminate all non-maximal ones. This can be used with backward simulation. We do not investigate this possibility further.

For delayed simulation, the lemma above fails, and we need all transitions, even those to little-brothers. This is shown in the next example.

Example 2.6.2 - Little-brothers are necessary for delayed simulation



Consider the automaton above, where $s \sqsubseteq^{\mathrm{de}} q \not\sqsubseteq^{\mathrm{de}} s$. Therefore $s \notin \max_a^{\sqsubseteq^{\mathrm{de}}}(q)$, but removing the transition $q \xrightarrow{a} s$ would make the language empty.

The following simple but important lemma is useful for future developments. It states that simulation preorder can be preserved along transitions, and that simulation equivalence can be preserved along maximal transitions (cf. Corollary 3 of [55]; for a generalization to alternating automata, see also Lemma 5.8.5).

Lemma 2.6.9. *For an NBA* $\mathcal{Q}$ *and* $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$, *let* $\sqsubseteq$ *be* x*-simulation (with induced equivalence* $\approx$*), and let* q *and* s *be two states in* $\mathcal{Q}$. *Then,*

1. *If* $q \sqsubseteq s$ *and* $q \xrightarrow{a} q'$, *then there exists* $s \xrightarrow{a} s'$ *s.t.* $q' \sqsubseteq s'$.
2. *If* $q \approx s$ *and* $q \xrightarrow{a} q' \in \max_a^{\sqsubseteq}(q)$, *then there exists* $s \xrightarrow{a} s' \in \max_a^{\sqsubseteq}(s)$ *s.t.* $q' \approx s'$.

Proof. Point 1. holds immediately from the definition of simulation, and Point 2. follows from maximality. For completeness, we prove the latter. Assume $q \approx s$ and let $q \xrightarrow{a} q' \in \max_a^{\sqsubseteq}(q)$. Since $q \sqsubseteq s$, by Point 1. there exists $s \xrightarrow{a} s'$ s.t. $q' \sqsubseteq s'$; w.l.o.g., s' can be taken in $\max_a^{\sqsubseteq}(s)$. Since $s \sqsubseteq q$, by Point 1. there exists $q \xrightarrow{a} q''$ s.t. $s' \sqsubseteq q''$. By transitivity, $q' \sqsubseteq q''$, and, by maximality of q', $q'' \sqsubseteq q'$. Therefore, by transitivity, $s' \sqsubseteq q'$, hence $q' \approx s'$. □

While we do not study simulation-based simplification of transitions in this thesis *per se*, maximal successors play a rôle in the following notion of max quotients (which generalizes to *minimax quotients* [55] for alternating models—see Section 5.8).

2.6.3 Max quotients

In max quotients we refine the transition structure of naïve quotients, in order to consider, in the quotient automaton, only those transitions induced by maximal successors in the original automaton. Formally, given an NBA $\mathcal{Q} = (Q, \Sigma, I, \Delta, F)$ with a preorder $\preceq \subseteq Q \times Q$ and induced equivalence $\approx$, the $\preceq$-*max quotient* of $\mathcal{Q}$ is the automaton $\mathcal{Q}^{\mathrm{m}}_{\preceq} = ([Q], \Sigma, [I], \Delta^{\mathrm{m}}_{\preceq}, [F])$ which is the same as the naïve quotient, except for the modified transition relation $\Delta^{\mathrm{m}}_{\approx}$, which is defined as follows:

$$([q], a, [q']) \in \Delta^{\mathrm{m}}_{\preceq} \quad \text{iff} \quad \exists(\widehat{q} \in [q], \widehat{q}' \in [q']) \cdot (\widehat{q}, a, \widehat{q}') \in \Delta \wedge \widehat{q}' \in \max_a^{\preceq}(\widehat{q})$$

That is, a transition in $\mathcal{Q}$ induces a transition in $\mathcal{Q}^{\mathrm{m}}_{\preceq}$ only if it is maximal (in $\mathcal{Q}$). The connection between max quotients and transition simplification is the following.

Lemma 2.6.10. *For an automaton $\mathcal{Q}$, let $\sqsubseteq$ be x-simulation preorder and $\approx$ the induced equivalence, for $x \in \{\mathrm{di}, \mathrm{de}\}$. Consider the following three constructions:*

a) Build the $\sqsubseteq$-max quotient $\mathcal{Q}^{m}_{\sqsubseteq}$.

b) Remove all $\sqsubseteq$-little-brothers in $\mathcal{Q}$ to obtain an automaton $\mathcal{R} := \mathcal{Q}^{\sqsubseteq}$, and then build the naïve quotient w.r.t. $\approx$ $\mathcal{R}_{\approx}$.

c) Build the naïve quotient $\mathcal{S} = \mathcal{Q}_{\approx}$, and then remove all little-brothers in $\mathcal{Q}_{\approx}$ to obtain the automaton $\mathcal{S}^{\sqsubseteq}$.

Then, the constructions above are equivalent, i.e., $\mathcal{Q}^{m}_{\sqsubseteq} = \mathcal{R}_{\approx} = \mathcal{S}^{\sqsubseteq}$.

Proof. Point a) and b) are equivalent since a transition is in the max quotient iff it is induced by a non-little-brother transition in Q. To see that a) and c) are equivalent too,

it suffices to show that max quotients don't have little-brothers (since only the transition structure is different between naïve and max quotients).

Indeed, we show that a little-brother in the quotient can only arise from a little-brother in the original automaton. Therefore, no little-brother is possible in max quotients (by definition). To this end, let $[q] \xrightarrow{a} [q'_0]$ and $[q] \xrightarrow{a} [q'_1]$ be two transitions with $[q'_0] \sqsubset [q'_1]$ (i.e., $[q'_0]$ is a little-brother in the quotient), and let $\widehat{q}_0 \in [q]$ and $\widehat{q}'_0 \in [q'_0]$ be any two states s.t. $\widehat{q}_0 \xrightarrow{a} \widehat{q}'_0$. We show that $\widehat{q}'_0$ is a little brother in $\mathcal{Q}$, i.e., $\widehat{q}'_0 \notin \max_a^{\sqsubseteq}(\widehat{q}_0)$. Since $[q] \xrightarrow{a} [q'_1]$, by definition there exist $\widehat{q}_1 \in [q]$ and $\widehat{q}'_1 \in [q'_1]$ s.t. $\widehat{q}_1 \xrightarrow{a} \widehat{q}'_1$. Since $\widehat{q}_0 \approx \widehat{q}_1$, by the definition of simulation, there exists $\widehat{q}_0 \xrightarrow{a} \widehat{q}''_0$ s.t. $\widehat{q}'_1 \sqsubseteq \widehat{q}''_0$. Since $[q'_0] \sqsubset [q'_1]$ and $q'_0 \approx \hat{q}'_0$ and $q'_1 \approx \hat{q}'_1$, clearly $[\widehat{q}'_0] \sqsubset [\widehat{q}'_1]$; by Lemma 2.6.4, $\widehat{q}'_0 \sqsubseteq \widehat{q}'_1$. But $\widehat{q}'_0$ and $\widehat{q}'_1$ are in different equivalence classes (hence inequivalent), therefore $\widehat{q}'_0 \sqsubset \widehat{q}'_1$. From $\widehat{q}'_1 \sqsubseteq \widehat{q}''_0$, by transitivity, $\widehat{q}'_0 \sqsubset \widehat{q}''_0$. Therefore $\widehat{q}'_0 \notin \max_a^{\sqsubseteq}(\widehat{q}_0)$. □

Since pruning little-brothers is sound w.r.t. direct simulation, direct simulation is GFQ w.r.t. max quotients; this fails for delayed simulation (cf. Example 2.6.2).

Corollary 2.6.11. *For a NBA $\mathcal{Q}$, let $\sqsubseteq^{\mathrm{di}}$ be direct simulation on $\mathcal{Q}$. Then, $\mathcal{L}(\mathcal{Q}) = \mathcal{Q}^m_{\sqsubseteq^{\mathrm{di}}}$.*

Remark 2.6.12 (Existential vs. universal vs. max quotients). If a transition $q \xrightarrow{a} q'$ does not induce a transition in the universal quotient (cf. Remark 2.4.4), by definition there exists $\widehat{q}$ s.t. $q \approx \widehat{q}$ and, for no $\widehat{q}'$ with $q' \approx \widehat{q}'$, $\widehat{q} \xrightarrow{a} \widehat{q}'$. By Lemma 2.6.9, q' is a little-brother. Therefore, universal quotients prune *only* those transitions induced by little-brothers. But not all of them, like max quotients do (by Lemma 2.6.10). Consider Example 2.6.2, where the universal quotienting leaves the automaton unchanged, while the max quotient prunes away transition $q \xrightarrow{a} s$. (This holds w.r.t. delayed simulation; by making q accepting instead of s, the same holds also w.r.t. direct simulation.) Thus, with respect to the transition relation, we have the following inclusions (which are strict in general):

$$\textit{max quotients} \subseteq \textit{universal quotients} \subseteq \textit{existential quotients.}$$

Chapter 3

Fixed-word simulations

Contents

3.1 Overview **35**

3.2 Multipebble simulations **37**

3.3 Containment preorders **42**

3.4 Fixed-word simulations **45**

3.4.1 Intuition 46

3.4.2 Definition and basic properties 47

3.4.3 Pebbles vs. fixed-words vs. containment 50

3.4.4 Complexity 53

3.5 Fixed-word multipebble delayed simulation **56**

3.6 Fixed-words and ranks **57**

3.6.1 Overview 58

3.6.2 Preliminaries 58

3.6.3 The ranking construction 59

3.6.4 Important properties of ranks $< \omega^2$ 64

3.6.5 The two directions of Theorem 3.5.1 66

3.1 Overview

We study novel simulation-like preorders for quotienting nondeterministic Büchi automata. Our main contribution is the introduction of *fixed-word delayed simulation*,

which is a novel simulation-like preorder included in containment. Our main result is that fixed-word delayed simulation can be used to quotient NBA (i.e., it is GFQ), and that it is coarser than previously studied GFQ preorders. In particular, we prove that it is coarser than multipebble simulations [42]. Therefore, quotienting w.r.t. fixed-word simulation equivalence results in automata that are smaller than previously possible. In turn, having smaller automata makes existing model-checking algorithms manipulating these automata faster.

The material presented in this chapter is an extension of material that has been published in [28]. The main technical results are:

- Theorem 3.4.2: Fixed-word delayed simulation is GFQ.
- Theorem 3.4.5: Computing fixed-word delayed simulation is PSPACE-complete.
- Theorem 3.5.1: Fixed-word delayed simulation cannot be further enlarged by generalizing it to multiple pebbles. This result is technically challenging and it relies on a ranking argument over ordinals. As a consequence, fixed-word delayed simulation is coarser than ordinary multipebble simulation.

The development of this chapter is as follows. In Section 2.5 we have presented several GFQ simulation preorders. An immediate question is whether there are other natural GFQ preorders coarser than simulation. A classic idea is to generalize simulation to *multipebble simulation* [42], by giving to Duplicator the ability of using several pebbles during the simulation game. We describe multipebble simulations in Section 3.2, where we recall that in the direct and delayed variant they are GFQ.

Another feature of multipebble simulations is that they are included in containment. In Section 3.3 we introduce direct, delayed, fair and backward containment, and we notice that multipebble simulation is included in the respective containment. In the delayed case, the inclusion is strict, and we consider whether *delayed containment* is a GFQ preorder.

This turns out to be false: We give a counterexample showing that delayed containment is not GFQ. Thus, in Section 3.4 we turn our attention to finer preorders: We introduce *fixed-word simulations*, which naturally fall between simulation and containment. We show that fixed-word delayed simulation is a GFQ preorder, and that computing fixed-word simulations is in general a PSPACE-complete problem. The complexity and GFQ results of this chapter are summarized in the gray area of Table 3.1.

In Section 3.5 we consider even coarser preorders. We generalize fixed-word simulation by allowing Duplicator to use *multiple pebbles* (in the style of [42]), and

Type	1-pebble	Multipebble	Fixed-word	Containment
Direct	✓	✓	✓	✓
Delayed	✓	✓	✓	×
Fair	×	×	×	×
Complexity	PTIME	between PSPACE and EXPTIME (cf. Sec. 5.10)	PSPACE-c. (cf. Sec. 3.4.4)	PSPACE-c. [111]

Table 3.1: Main contributions (gray area; ✓= GFQ, ×= not GFQ)

we prove that *fixed-word multipebble delayed simulation* coincides with ordinary (i.e., 1-pebble) fixed-word delayed simulation. The proof of this fact is non-trivial and relies on a ranking argument over ordinals; we present the proof in great detail in Section 3.6, which concludes the chapter.

3.2 Multipebble simulations

Multipebble simulations are best defined in a game-theoretic way. The game is played by two players, Spoiler and Duplicator. The game starts with two pebbles placed on two states of the automaton, say on q_0 and s_0. Assume Spoiler controls the pebble on q_0 and Duplicator the one on s_0. The objective of Duplicator is to show that s_0 can simulate q_0, while Spoiler has the complementary objective.

The k-pebble simulation game proceeds in rounds, Spoiler starting first and Duplicator following. Suppose that, in the current round, Spoiler's pebble is on state q and that Duplicator's pebbles are on states in $\mathbf{s}$, with $1 \leq |\mathbf{s}| \leq k$. Spoiler selects an input symbol $a \in \Sigma$ and moves her pebble from q to an a-successor q'. Then, it is Duplicator's turn to reply. In ordinary simulation, Duplicator has only one pebble, which is moved to some a-successor. In multipebble simulation, Duplicator controls several pebbles. Each pebble can be moved to some a-successor, or it can be even "split" to several a-successors. Moreover, Duplicator always has the possibility of "reclaiming" extra pebbles when they are needed elsewhere, with the proviso that at least one pebble

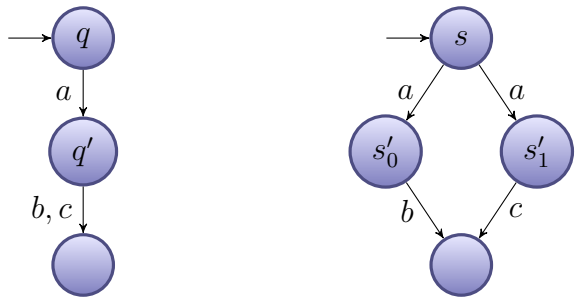


Figure 3.1: More pebbles help Duplicator

remains on the automaton. Thus, Duplicator replies to Spoiler by moving pebbles from states in s to states in s', s.t. any pebble in s' is an a-successor of some pebble in s (written $\mathrm{s} \overset{a}{\Longrightarrow} \mathrm{s}'$), and there are at most k pebbles in s'.

In a k-simulation game, Duplicator can put *up to* k pebbles on the automaton. Since not all of them have to be used, $k+1$ pebbles are always at least as good as k. Moreover, since putting more than one pebble on the same state is not meaningful, more than n pebbles are actually superfluous, where n is the number of states of the automaton.

Having multiple pebbles allows Duplicator to delay committing to any particular choice. Consider Figure 3.1. It is a classic example: q clearly simulates s, but s does not ordinarily (i.e., with $k = 1$) simulate q. Indeed, Spoiler can choose input symbol a and move her pebble from q to q'. Then, Duplicator has to face a choice of whether to move her pebble from s to either s'_0 or to s'_1: In any case, Spoiler can play input symbol c or b, respectively, and Duplicator cannot reply. However, if Duplicator can control up to $k = 2$ pebbles, then she splits pebbles to both successors $\mathrm{s}' = \{s'_0, s'_1\}$. From this configuration, if Spoiler plays symbol b, then Duplicator reclaims the pebble on s'_1 and responds from s'_0, and vice versa if Spoiler plays symbol c. Thus, Duplicator does not resolve the nondeterminism immediately, but she procrastinates it to the next round: This allows her to "buy time", waiting for Spoiler to reveal her intents. We proceed to give a formal presentation of multipebble simulations.

Forward multipebble simulations We summarize results from [42]. Let $\mathcal{Q}$ be a NBA. We define the basic k-simulation game as follows. Spoiler's set of positions is $P_{\text{Sp}} := Q \times 2^{Q,k}$, and Duplicator's set of positions is $P_{\text{Dup}} := Q \times 2^{Q,k} \times \Sigma \times Q$. The initial configuration $\langle q_0, \mathrm{s}_0 \rangle$. If the current configuration in round i is $\langle q_i, \mathrm{s}_i \rangle$, then the next configuration $\langle q_{i+1}, \mathrm{s}_{i+1} \rangle$ is determined as follows:

- First, Spoiler chooses an input symbol a_i and a transition $q_i \xrightarrow{a_i} q_{i+1}$.

- Then, Duplicator chooses a matching multipebble transition $\mathbf{s}_i \overset{a_i}{\Longrightarrow} \mathbf{s}_{i+1}$ s.t. $|\mathbf{s}_{i+1}| \leq k$.

The game can halt prematurely, for pebbles may get stuck. In this case, the winning condition is as follows: If the left pebble cannot be moved, then Duplicator wins. Otherwise, if no right pebble can be moved, then Spoiler wins.

In all other cases, the game goes on forever. Let $\pi = \langle q_0, \mathbf{s}_0 \rangle \langle q_1, \mathbf{s}_1 \rangle \cdots \in P^{\omega}_{\mathrm{Sp}}$ be the resulting play, and let $\pi_0 = q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \cdots$ be the infinite path built by Spoiler, and $\pi_1 = \mathbf{s}_0 \overset{a_0}{\Longrightarrow} \mathbf{s}_1 \overset{a_1}{\Longrightarrow} \cdots$ the infinite multipath built by Duplicator.

The winning condition for delayed and fair simulation requires some technical preparation (cf. [42]). Fix an infinite multipath $\pi_1 = \mathbf{s}_0 \overset{a_0}{\Longrightarrow} \mathbf{s}_1 \overset{a_1}{\Longrightarrow} \cdots$. Given a source state $s \in \mathbf{s}_i$, a destination state $s' \in \mathbf{s}_k$, and an intermediate state $\hat{s} \in \mathbf{s}_j$, with $i \leq j \leq k$, we write $s \longrightarrow^*_{\hat{s}} s'$ iff there exists a path $\pi \in \pi_1$ from s to s' passing through $\hat{s}$, i.e., $\pi = s \xrightarrow{a_i} \cdots \hat{s} \cdots \xrightarrow{a_{k-1}} s'$. For readability, we do not note the dependency on π_1. The notation for intermediate states is overloaded over the set of accepting states: We write $s \longrightarrow^*_F s'$ to mean $s \longrightarrow^*_{\hat{s}} s'$ for some $\hat{s} \in F$. Finally, we write $\mathbf{s}_i \Longrightarrow^*_F \mathbf{s}_k$ iff, for any $s' \in \mathbf{s}_k$, there exists $s \in \mathbf{s}_i$ s.t. $s \longrightarrow^*_F s'$. In the last case, we also say that $\mathbf{s}_i$ *is good at round* k. Notice that the universal quantification is on destination states, which adds a sort of backward flavor to multipebble simulation [86]. For a multipath π_1, we say that it is *fair* iff $\mathbf{s}_i$ is good at some later round for infinitely many i's (thus "fair" is overloaded on multipaths).

Classically, the following winning conditions $W^x \subseteq P^{\omega}_{\mathrm{Sp}}$ have been considered [42], corresponding to different notions of simulation.

1. *Direct k-simulation*. Duplicator wins if, whenever q_i is accepting, then *every* pebble $s \in \mathbf{s}_i$ is accepting:

$$\pi \in W^{\mathrm{di}} \iff (\forall i \cdot q_i \in F \implies \mathbf{s}_i \subseteq F)$$

2. *Delayed k-simulation*. Duplicator wins if, whenever q_i is accepting, then there exists $j \geq i$ s.t. $\mathbf{s}_i$ is good at round j:

$$\pi \in W^{\mathrm{de}} \iff (\forall i \cdot q_i \in F \implies \exists (j \geq i) \cdot \mathbf{s}_i \Longrightarrow^*_F \mathbf{s}_j)$$

3. *Fair k-simulation*. Duplicator wins if, whenever π_0 is fair, then π_1 is fair.

$$\pi \in W^{\mathrm{f}} \iff (\pi_0 \text{ fair} \implies \pi_1 \text{ fair })$$

We summarize these winning condition below.

Simulation game for $\sqsubseteq^{k\text{-}x}$, with $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$

Current configuration		Next configuration
$\langle q_i, \mathbf{s}_i \rangle$	$\dashrightarrow$	$\langle q_{i+1}, \mathbf{s}_{i+1} \rangle$

Spoiler chooses $q_i \xrightarrow{a_i} q_{i+1}$

Duplicator replies with $\mathbf{s}_i \overset{a_i}{\Longrightarrow} \mathbf{s}_{i+1}$, where $|\mathbf{s}_{i+1}| \leq k$

Winning condition

$x = \mathrm{di}: \quad \forall (i \geq 0) \cdot q_i \in F \implies \mathbf{s}_i \subseteq F$

$x = \mathrm{de}: \quad \forall (i \geq 0) \cdot q_i \in F \implies \exists (j \geq i) \cdot \mathbf{s}_i \Longrightarrow_F^* \mathbf{s}_j$

$x = \mathrm{f}: \quad \pi_0$ fair $\implies \pi_1$ fair

Backward multipebble simulation We extend the multipebble approach to backward simulation in a natural way (which has not been considered in [42]). The interaction between players is analogous to the forward case, except that now transitions are taken backward. For two sets of pebbles $\mathbf{s}$ and $\mathbf{s}'$, we write $\mathbf{s} \overset{a}{\Longleftarrow} \mathbf{s}'$ iff $\forall s' \in \mathbf{s}' \cdot \exists s \in \mathbf{s} \cdot s \overset{a}{\longleftarrow} s'$. ($\overset{a}{\Longleftarrow}$ is different from $(\overset{a}{\Longrightarrow})^{-1}$.)

4. *Backward k-simulation*. Duplicator wins if both the conditions below are satisfied:

   - Whenever q_i is accepting, then *every* pebble $s \in \mathbf{s}_i$ is accepting, and
   - Whenever q_i is initial, then *some* pebble $s \in \mathbf{s}_i$ is initial. I.e.,

$$\pi \in W^{\mathrm{bw}} \iff \forall i \, (q_i \in F \implies \mathbf{s}_i \subseteq F) \wedge (q_i \in I \implies \mathbf{s}_i \cap I \neq \emptyset)$$

Simulation game for $\sqsubseteq^{k\text{-bw}}$

Current configuration $\langle q_i, \mathbf{s}_i \rangle$ $\dashrightarrow$ Next configuration $\langle q_{i+1}, \mathbf{s}_{i+1} \rangle$

Spoiler chooses $q_i \xleftarrow{a_i} q_{i+1}$

Duplicator replies with $\mathbf{s}_i \xleftarrow{a_i} \mathbf{s}_{i+1}$, where $|\mathbf{s}_{i+1}| \leq k$

Winning condition

1) $\forall (i \geq 0) \cdot q_i \in F \implies \mathbf{s}_i \subseteq F$, and
2) $\forall (i \geq 0) \cdot q_i \in I \implies \mathbf{s}_i \cap I \neq \emptyset$

We denote the previous acceptance conditions with $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}, \mathrm{bw}\}$. We are now ready to define the simulation relation $\sqsubseteq^{k\text{-}x}$. We say that a k-set $\mathbf{s}$ x-simulates a state q, written $q \sqsubseteq^{k\text{-}x} \mathbf{s}$, iff Duplicator has a winning strategy in the corresponding k, x-simulation game starting from configuration $\langle q, \mathbf{s} \rangle$. We overload the simulation relation on singletons, and we write $q \sqsubseteq^{k\text{-}x} s$ iff $q \sqsubseteq^{k\text{-}x} \{s\}$. Note that, in general, $\sqsubseteq^{k\text{-}x}$ is not a transitive relation. However, for sufficiently large n, $\sqsubseteq^{n\text{-}x}$ is transitive (since Duplicator never runs out of pebbles when composing simulation games); when we speak of multipebble simulation, we mean such a maximal transitive relation.

Having more pebbles gives more power to Duplicator. For any $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}, \mathrm{bw}\}$, we obtain the following hierarchy:

$$\sqsubseteq^{1\text{-}x} \subseteq \sqsubseteq^{2\text{-}x} \subseteq \sqsubseteq^{3\text{-}x} \subseteq \cdots$$

Moreover, for any fixed $k > 0$, $\sqsubseteq^{k\text{-}x}$ is computable in polynomial time (Dc) and it implies containment (Da) (shown in the next section). Most importantly, it is GFQ (Db).

Theorem 3.2.1. *For any $k > 0$ and $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{bw}\}$, k-pebble x-simulation is GFQ.*

For forward simulations, these properties have been established in [42]; we observe them also in the backward case. That work also gives algorithms for computing multipebble

simulations; we treat these in Section 5.9 in the more general context of alternating automata.

3.3 Containment preorders

Multipebble x-simulation naturally relates to a corresponding notion of *x-containment*. We define x-containments, and show that multipebble x-simulation under-approximates x-containment. We also establish which containment is GFQ.

A containment preorder is obtained as a modification of the usual step-wise interaction between players: In the *containment game* between q and s there are only two rounds. Spoiler moves first and selects both an infinite word $w = a_0 a_1 \cdots$ and an infinite path $\pi_0 = q \xrightarrow{a_0} q_1 \xrightarrow{a_1} \cdots$ over w. Then, Duplicator replies with a corresponding infinite path $\pi_1 = s \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots$ over w. The winning condition depends on the kind x of containment, and it is exactly the same as in simulation; the only difference is about *how* π_0 and π_1 are built. If Duplicator wins the x-containment game between q and s, then we write $q \subseteq^x s$. The rules for forward containments are summarized below. Clearly, $\subseteq^{\mathrm{di}}$ is included in $\subseteq^{\mathrm{de}}$, and the latter is included in $\subseteq^{\mathrm{f}}$.

Simulation game for $\subseteq^x$, with $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$

Spoiler chooses $\pi_0 = q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \cdots$

Duplicator replies with $\pi_1 = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots$

Winning condition

$x = \mathrm{di}: \quad \forall (i \geq 0) \cdot q_i \in F \implies s_i \in F$

$x = \mathrm{de}: \quad \forall (i \geq 0) \cdot q_i \in F \implies \exists (j \geq i) \cdot s_j \in F$

$x = \mathrm{f}: \quad \pi_0 \text{ fair} \implies \pi_1 \text{ fair}$

Backward containment is slightly different, since it needs to be about *finite paths*, which, of course, go backwards.

Simulation game for $\subseteq^{\mathrm{bw}}$

Spoiler chooses $\pi_0 = q_0 \xleftarrow{a_0} q_1 \xleftarrow{a_1} \cdots \xleftarrow{a_{n-1}} q_n$

Duplicator replies with $\pi_1 = s_0 \xleftarrow{a_0} s_1 \xleftarrow{a_1} \cdots \xleftarrow{a_{n-1}} s_n$

Winning condition

1) $\forall (i \geq 0) \cdot q_i \in F \implies s_i \in F$
2) $q_n \in I \implies s_n \in I$

Remark 3.3.1. In the definition of backward containment, we could have required the following weaker winning condition for Duplicator: If Spoiler produces a path s.t. $q_n \in I$, then Duplicator has to produce a path s.t. $s_n \in I$, and final states have to be matched as in condition 1). In general, the latter definition would produce a coarser preorder: In fact, in the case $q_n \notin I$ Duplicator does not have any obligation, whereas in the current notion she always has to match final states as in condition 1). However, if we assume that every state is reachable from some initial state, then the two notions coincide, since Spoiler can always extend any path to an initial one. The current definition matches multipebble backward simulation.

While language inclusion is a property of two automata, containment is a property of two *states*. In the fair case, the two are tightly related. Fix an automaton $\mathcal{Q}$. On the one hand, for two states q and s, $q \subseteq^{\mathrm{f}} s$ iff $\mathcal{L}(\mathcal{Q}_q) \subseteq \mathcal{L}(\mathcal{Q}_s)$, where $\mathcal{Q}_q$ and $\mathcal{Q}_s$ are two automata equal to $\mathcal{Q}$, except that the initial state has been changed to q and s, respectively. On the other hand, for two different automata $\mathcal{Q}$ and $\mathcal{S}$, $\mathcal{L}(\mathcal{Q}) \subseteq \mathcal{L}(\mathcal{S})$ if, for any initial q state in $\mathcal{Q}$, there exists an initial state s in $\mathcal{S}$ s.t. $q \subseteq^{\mathrm{f}} s$. (Notice that the latter condition is only sufficient for language inclusion, and not necessary in general.)

The reason why we have introduced containment preorders is that multipebble simulations under-approximate them. This might hint at why multipebble simulations are GFQ, and possibly suggest coarser GFQ preorders.

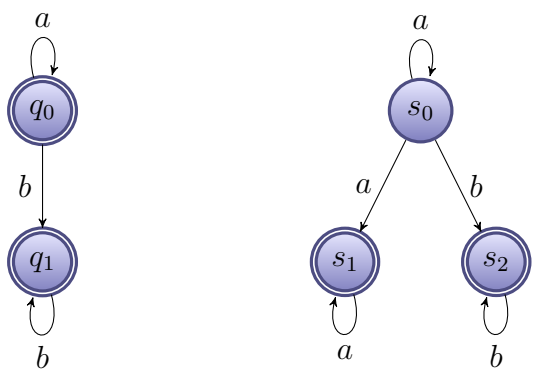
Lemma 3.3.2. *For $k > 0$ and $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}, \mathrm{bw}\}$, k-pebble x-simulation implies x-containment. I.e., for any NBA $\mathcal{Q}$ and states q and s therein, if $q \sqsubseteq^{k\text{-}x} s$, then $q \subseteq^{x} s$.*

One might wonder whether multipebble x-simulation "reaches" the corresponding x-containment in the limit. Since for any given automaton, $\sqsubseteq^{n\text{-}x} = \sqsubseteq^{(n+1)\text{-}x}$ where n is the size of the automaton (as number of states), we actually ask whether such a maximal simulation $\sqsubseteq^{n\text{-}x}$ equals $\subseteq^x$. This holds only for "finitary" winning conditions.

Lemma 3.3.3. *Fix an automaton $\mathcal{Q}$ with n states. For $x \in \{\mathrm{di}, \mathrm{bw}\}$, x-containment equals n-pebble x-simulation.*

We give a counterexample showing that the lemma above fails for both delayed and fair containment.

Example 3.3.1 - Multipebble simulations "do not reach" containment



We have $q_0 \subseteq^x s_0$ for both $x \in \{\mathrm{de}, \mathrm{f}\}$. Since the automaton on the left is deterministic, the input word uniquely determines a path through it. There are two cases to consider: If Spoiler plays $w = a^\omega$, then Duplicator plays transition $s_0 \xrightarrow{a} s_1$ and then stays in s_1 forever. If Spoiler plays $w = a^*b^\omega$, then Duplicator stays in s_0 until the first b, at which time she takes transition $s_0 \xrightarrow{b} s_2$ and then she remains in s_2 forever. In both cases, Duplicator wins by constructing a fair path.

However, $q_0 \not\sqsubseteq^{k\text{-}x} s_0$ for any $k > 0$ and both $x \in \{\mathrm{de}, \mathrm{f}\}$. To see this, suppose Spoiler plays letter a for a very high number of rounds. What Duplicator can do is to split pebbles at s_0, by taking the sequence of transitions

$$\pi = \{s_0\} \overset{a}{\Longrightarrow} \{s_0, s_1\} \overset{a}{\Longrightarrow} \{s_0, s_1\} \overset{a}{\Longrightarrow} \cdots$$

Since the automaton on the left is everywhere accepting, Spoiler will always build a fair path. The issue for Duplicator is that s_0 is not accepting and it has no accepting

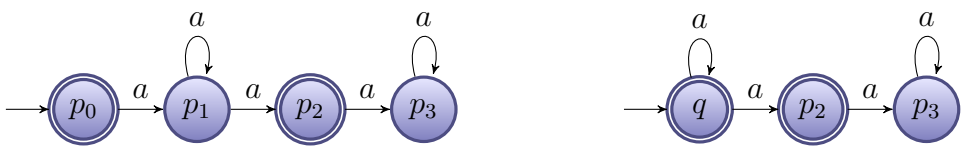
predecessor in the multipath π as above. Therefore, in order to build a fair multipath, Duplicator is eventually forced to drop the pebble on s_0, and to take transition $\{s_0, s_1\} \overset{a}{\Longrightarrow} \{s_1\}$. At this point, Spoiler plays b and wins, since Duplicator gets stuck.

One might wonder which containment is GFQ. Certainly, fair containment is not GFQ, since it is coarser than fair simulation which is not GFQ. On the other hand, direct and backward containment are GFQ, since they coincide with the corresponding maximal multipebble simulations, which are GFQ (cf. Theorem 3.2.1).

Theorem 3.3.4. *For $x \in \{\mathrm{di}, \mathrm{bw}\}$, x-containment is GFQ.*

For delayed containment, the situation is different, since multipebble simulation (which is GFQ) does not reach delayed containment (as we have seen above). The gap turns out to be substantial, since *delayed containment is not GFQ*.

Example 3.3.2 - Delayed containment is not GFQ



a) The original automaton $\mathcal{Q}$ b) The quotient automaton $\mathcal{Q}_{\approx}$

Consider automaton $\mathcal{Q}$ above. We have that p_0 is delayed containment equivalent to p_1. Notice that $\mathcal{Q}$ does not accept a^ω. However, the quotient automaton $\mathcal{Q}_{\approx}$ above, obtained by identifying p_0 and p_1, does accept a^ω.

The relationship between simulations, multipebble simulations and containment is shown in Figure 3.2. An arrow indicates inclusion, which is always strict except when marked with "=". The red area denotes non-GFQ preorders; in particular, $\subseteq^{\mathrm{de}}$ is too coarse for quotienting. In the next section, we look at a finer GFQ preorder.

3.4 Fixed-word simulations

In the previous section we have seen that delayed containment is not GFQ. A natural question is whether there are other GFQ preorders intermediate between simulation and containment. We answer this question positively: We introduce *fixed-word delayed simulation*, which is a natural GFQ generalization of delayed simulation. While the

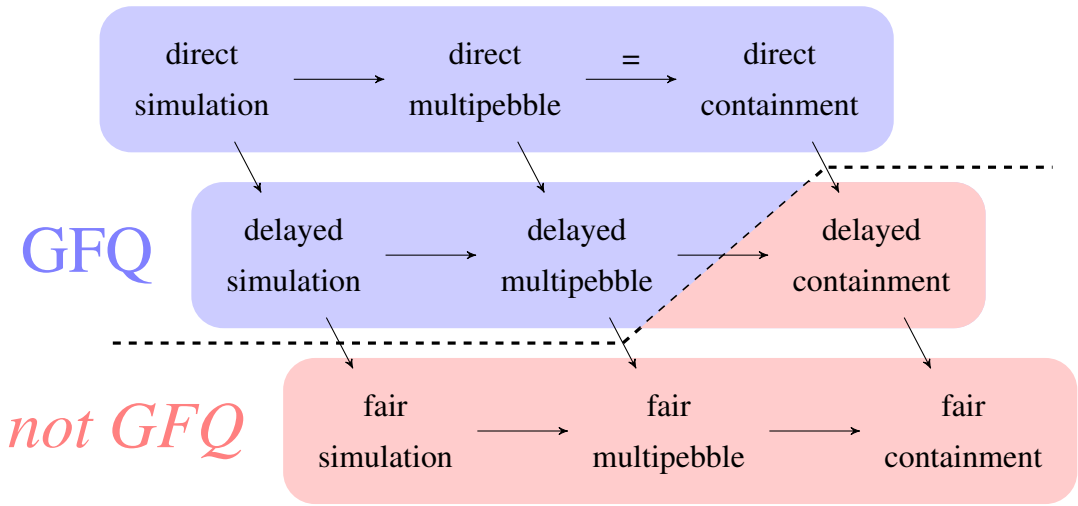


Figure 3.2: Simulation, pebbles, and containment

main results are about the delayed case, for completeness we also treat fixed-word direct, fair and backward simulations.

Outline In Section 3.4.1 we introduce fixed-word simulations informally. Formal definitions are given in Section 3.4.2. The exact relationship between multipebble simulations, fixed-word simulations, and containment is examined in Section 3.4.3. In particular, we show that fixed-word delayed simulation is coarser than multipebble delayed simulation (cf. Theorem 3.4.3). This is a non-trivial fact, and it is proved separately in Section 3.5. Finally, in Section 3.4.4 we establish that computing fixed-word simulations is PSPACE-complete, via mutual reductions to the universality problem for alternating automata.

3.4.1 Intuition

We have already seen that Spoiler and Duplicator can build infinite paths using different interaction modes. In the previous sections we have considered simulation interaction, multipebble interaction, and containment interaction. The conceptually simplest one is containment interaction, where the game is played in just two rounds: Spoiler chooses a path π_0 over a certain word, and Duplicator replies by choosing a path π_1 over the same word. The winner depends on the winning condition, which is a predicate on the two paths π_0 and π_1. On the other hand, in simulation interaction the two players build π_0 and π_1 incrementally, with Spoiler starting first. After ω turns, π_0 and π_1 are finally constructed, and the winner depends again on a predicate on π_0 and π_1. Finally,

multipebble simulations generalize the usual simulation game, and allow Duplicator to build a multipath instead of a path.

Fixed-word simulations are a mixture of simulation and containment. Like in simulation, transitions are chosen in an interleaving manner. But, unlike simulation, the input word $w = a_0 a_1 \cdots$ is chosen beforehand by Spoiler: w is *fixed* during the rest of the game. This latter feature makes fixed-word simulation akin to containment, since Spoiler reveals an unbounded amount of information in only one move. We show that, even by revealing this unbounded amount of information, Spoiler does not get "too weak", in the sense that the resulting preorder is still GFQ.

3.4.2 Definition and basic properties

We first discuss forward fixed-word simulations. Let $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$. Spoiler starts off the game by declaring an infinite input word $w = a_0 a_1 \cdots \in \Sigma^\omega$. Then, the game proceeds in rounds like in simulation, except that now, at round i, the next input symbol a_i is fixed. In the w-simulation game $G_w^x(q, s)$ (which is parameterized by w), the set of positions of Spoiler is $P_0 = Q \times Q \times \omega$, the set of positions of Duplicator is $P_1 = Q \times Q \times Q \times \omega$ and $\langle q, s, 0\rangle$ is the initial position. If the current position is $\langle q_i, s_i, i\rangle$, then then next one $\langle q_{i+1}, s_{i+1}, i+1\rangle$ is determined as follows:

- Spoiler selects a transition $q_i \xrightarrow{a_i} q_{i+1}$.

- Duplicator selects a transition $s_i \xrightarrow{a_i} s_{i+1}$.

In both cases, a_i is no longer under the control of Spoiler at this point, since it is fixed in advance and has to match the corresponding symbol in w. The winning conditions are the usual one as in direct, delayed and fair simulation.

For the backward case $x = \mathrm{bw}$, Spoiler declares a *finite* word $w = a_0 a_1 \cdots a_{n-1} \in \Sigma^*$. The w-simulation game lasts n rounds, during which transitions are taken backward.

w-simulation game for $\sqsubseteq_w^x$, with $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$

Current configuration $\langle q_i, s_i, i\rangle$ $\dashrightarrow$ Next configuration $\langle q_{i+1}, s_{i+1}, i+1\rangle$

Spoiler chooses $q_i \xrightarrow{a_i} q_{i+1}$

Duplicator replies with $s_i \xrightarrow{a_i} s_{i+1}$

Winning condition

$x = \mathrm{di}: \quad \forall (i \geq 0) \cdot q_i \in F \implies s_i \in F$

$x = \mathrm{de}: \quad \forall (i \geq 0) \cdot q_i \in F \implies \exists (j \geq i) \cdot s_j \in F$

$x = \mathrm{f}: \quad \pi \text{ fair} \implies \pi' \text{ fair}$

w-simulation game for $\sqsubseteq_w^{\mathrm{bw}}$

Current configuration $\langle q_i, s_i, i\rangle$ $\dashrightarrow$ Next configuration $\langle q_{i+1}, s_{i+1}, i+1\rangle$

Spoiler chooses $q_i \xleftarrow{a_i} q_{i+1}$

Duplicator replies with $s_i \xleftarrow{a_i} s_{i+1}$

Winning condition

1) $\forall (i \geq 0) \cdot q_i \in F \implies s_i \in F$

2) $\forall (i \geq 0) \cdot q_i \in I \implies s_i \in I$

If Duplicator wins the (w, x)-game starting from position $\langle q, s, 0\rangle$, then we write $q \sqsubseteq_w^x s$. If $q \sqsubseteq_w^x s$ holds for every w, then s *fixed-word simulates* q, written as $q \sqsubseteq_{\mathrm{fx}}^x s$.

Clearly, fixed-word simulation is a preorder between simulation and containment.

Lemma 3.4.1. *For any $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}, \mathrm{bw}\}$ and states q and s,*

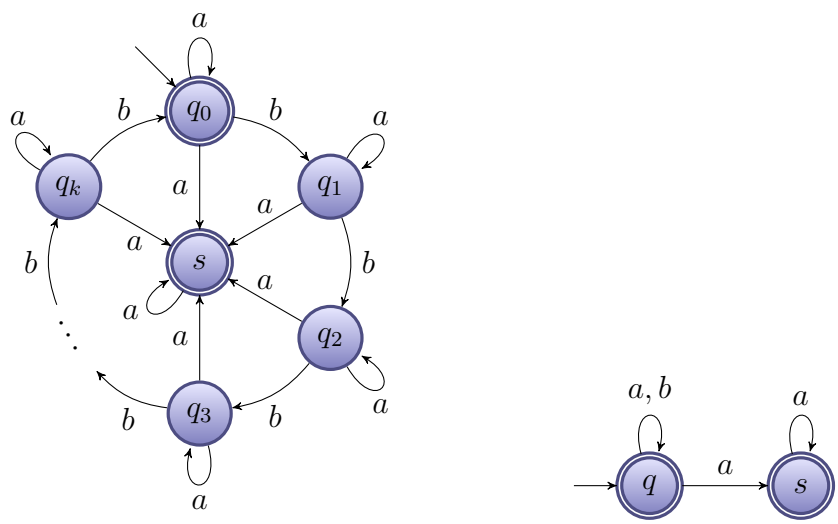
$$q \sqsubseteq^x s \text{ implies } q \sqsubseteq_{fx}^x s \text{ implies } q \subseteq^x s$$

Moreover, since direct and backward containment are GFQ, fixed-word direct and backward simulations are GFQ as well. More importantly, *fixed-word delayed simulation is GFQ*. This follows directly from the very same proof that ordinary delayed simulation is GFQ [43], which we do not replicate here; it relies on the observation that the input word is always fixed in advance when reasoning about paths in the quotient automaton.

Theorem 3.4.2. *Fixed-word delayed simulation is a GFQ preorder.*[1]

Fixed-word delayed simulation quotients can be more succinct than (multipebble) delayed simulation quotients by an arbitrarily large factor, as shown in the next example.

Example 3.4.1 - Fixed-word delayed simulation quotients can achieve arbitrary large compression ratios



a) Family of automata $\mathcal{Q}_k$ b) The quotients $\mathcal{Q}_k/{\approx}_{\mathrm{fx}}^{\mathrm{de}}$

[1] It even holds that $\sqsupseteq_{\mathrm{fx}}^{\mathrm{de}}$ (the transpose of $\sqsubseteq_{\mathrm{fx}}^{\mathrm{de}}$) is *jumping-safe*, in the terminology of Chapter 4.

Consider the family of automata $\mathcal{Q}_k$ above, over the alphabet $\Sigma = \{a, b\}$. Each $\mathcal{Q}_k$ has $k+2$ states. Any state q_i in the ring is universal (and so it is $\mathcal{Q}_k$): Indeed, each q_i accepts either a^ω via state s, or it reads the prefix a^*b until the first b, and the rest of the word is accepted from the next state $q_{i+1 \mod k+1}$. If there are only finitely many b's, then s is eventually reached. Otherwise, q_0 is visited infinitely often. In both cases, an accepting run is produced. Therefore, q_i is universal. Except for the fact that s is simulated by every state, no two different states q_i, q_j are $\sqsubseteq^{\mathrm{de}}$-comparable: For example, Spoiler wins from configuration $\langle q_0, q_1\rangle$ by playing b and going to $\langle q_1, q_2\rangle$, and then playing a^* until at some point, in order to be accepting, Duplicator has to move her pebble from q_2 to s. From configuration $\langle q_1, s\rangle$, Spoiler plays b and wins. Notice that multiple pebbles don't help Duplicator because q_2 is not accepting, and when Spoiler plays a^* any non-accepting pebble would have to be eventually dropped. Also, no two different states are in backward simulation, since each state has a word w to backward-reach the initial state q_0 s.t. no other state can backward-reach q_0 via the same word w: For example, the word for s is just a, and that for q_j is b^j.

However, all states except s are fixed-word delayed simulation equivalent:

$$s \sqsubseteq^{\mathrm{de}}_{\mathrm{fx}} q_0 \approx^{\mathrm{de}}_{\mathrm{fx}} q_1 \approx^{\mathrm{de}}_{\mathrm{fx}} \cdots \approx^{\mathrm{de}}_{\mathrm{fx}} q_k$$

Indeed, when the input word w is known in advance, Duplicator can directly build from the states in the ring an accepting run over w, thus winning the game. Therefore, we obtain the 2-states quotient automaton $\mathcal{Q}_k / \approx^{\mathrm{de}}_{\mathrm{fx}}$ on the right above. Thus, the compression ratio achieved by fixed-word delayed simulation can be arbitrarily large.

The relationships between multipebble and fixed-word simulations and containments are non-trivial. We explore them in the next section.

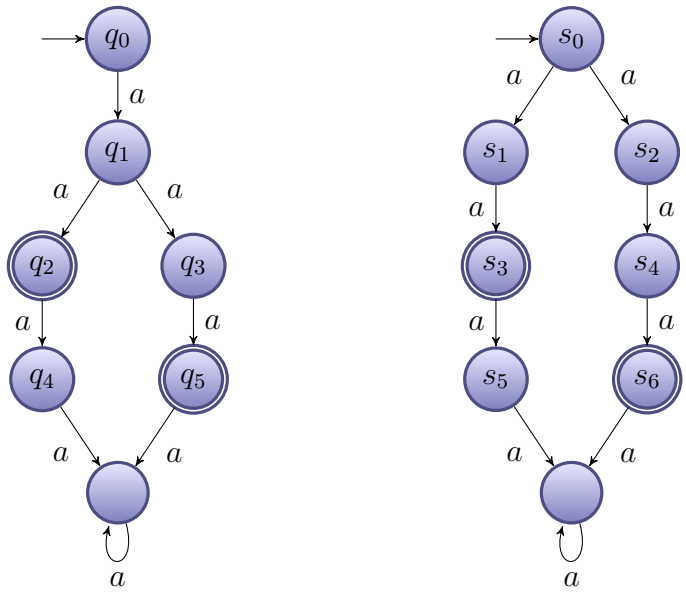
3.4.3 Pebbles vs. fixed-words vs. containment

The direct and backward case Since multipebble direct and backward simulation coincide with the respective containments (cf. Lemma 3.3.3), it trivially follows that fixed-word direct and backward simulations are included in the respective multipebble simulations. I.e., for $k > 0$, $x \in \{\mathrm{di}, \mathrm{bw}\}$, and states q and s,

$$q \sqsubseteq^{x}_{\mathrm{fx}} s \quad \textit{implies} \quad q \sqsubseteq^{k\text{-}x} s \quad \textit{iff} \quad q \subseteq^{x} s$$

The next example shows that the containment is strict.

Example 3.4.2 - Fixed-word direct and backward simulations are strictly included in containment



The reason is simple. It relies on the fact that, even if Spoiler declares the input word in advance, the exact time she is accepting is not known beforehand by Duplicator.

Consider the two automata above. Since they are unary, Spoiler can only choose the unique infinite word $w = a^\omega$. Direct containment holds between q_0 and s_0. Indeed, there are only two distinct infinite paths from q_0 over a^ω: The first $\pi_0 = q_0q_1q_2q_4\cdots$ is accepting in q_2 at round 2, and the second $\pi_1 = q_0q_1q_3q_5\cdots$ is accepting in q_5 at round 3. Clearly, each π_i's can be matched by Duplicator to a path from s_0 with identical properties.

However, $q_0 \not\sqsubseteq^{\text{di}}_{\text{fx}} s_0$. Indeed, Spoiler takes transition $q_0 \xrightarrow{a} q_1$, and Duplicator has to face a choice. Suppose she takes the left transition $s_0 \xrightarrow{a} s_1$. From configuration $\langle q_1, s_1\rangle$, Spoiler takes the right transitions $q_1 \xrightarrow{a} q_3 \xrightarrow{a} q_5$. On the other hand, Duplicator is forced to take transitions $s_1 \xrightarrow{a} s_3 \xrightarrow{a} s_5$. Configuration $\langle q_5, s_5\rangle$ is winning for Spoiler, since q_5 is accepting and s_5 is not. An analogous reasoning applies if Duplicator took the right transition $s_0 \xrightarrow{a} s_2$.

A similar argument holds for backward containment and fixed-word simulation.

The fair case For fixed-word fair simulation, the situation is symmetric, since fixed-word fair simulation coincides with fair containment: Indeed, once Spoiler reveals the

input word w, both players can proceed in playing the game independently, trying to build a fair path over w, if any. Duplicator wins when either Spoiler is not able to build such a fair path, or when she is herself able to build a fair path. This is exactly what happens in fair containment. Consequently, multipebble fair simulation is strictly contained in fixed-word simulation. I.e., for $k > 0$ and states q and s,

$$q \sqsubseteq^{k\text{-f}} s \quad \textit{implies} \quad q \sqsubseteq^{\text{f}}_{\text{fx}} s \quad \textit{iff} \quad q \subseteq^{\text{f}} s$$

The delayed case The most interesting case is the delayed one, where multipebble simulation, fixed-word simulation and containment are all distinct preorders. Like in the direct and backward cases and unlike in the fair case, fixed-word delayed simulation is strictly included in the corresponding containment.

Example 3.4.3 - Fixed-word delayed simulation is strictly included in delayed containment

The obvious reason is that, in the delayed case, fixed-word simulation is GFQ, while containment is not. We also give a concrete example, in order to shed some light on why Duplicator gains more power in containment than in fixed-word simulation. Consider again automaton $\mathcal{Q}$ from Example 3.3.2. We have that p_1 delayed contains p_0, i.e., $p_0 \subseteq^{\text{de}} p_1$. However, $p_0 \not\sqsubseteq^{\text{de}}_{\text{fx}} p_1$: To see this, notice that, since the automaton is unary, revealing the unique input word a^ω does not help Duplicator. Indeed, starting from configuration $\langle p_0, p_1, 0\rangle$, even when Duplicator knows the input word in advance, she still does not know exactly at which time Spoiler will be accepting. And this is crucial: Spoiler, which is initially accepting, takes transition $p_0 \xrightarrow{a} p_1$ and remains in p_1 indefinitely. Spoiler can postpone the next time she can be accepting in p_2 by an unbounded amount of time. Now, Duplicator, which has a pending obligation to satisfy, has to eventually leave state p_1 and take transition $p_1 \xrightarrow{a} p_2$. Thus, at some round $k > 0$, the game is in configuration $\langle p_1, p_2, k\rangle$, from which Spoiler clearly wins by taking transition $p_1 \xrightarrow{p}_2$, being accepting once more and preventing Duplicator from visiting an accepting state anymore in the future. Thus, $p_0 \not\sqsubseteq^{\text{de}}_{\text{fx}} p_1$. The reason why $p_0 \subseteq^{\text{de}} p_1$ holds, however, is that Spoiler has to reveal in advance at which time k she will go to p_2 ($k = \omega$ if she stays in p_1 for ever). Once k is known, Duplicator reacts by staying the first k steps in p_1, and only then she goes to p_2 (if $k = \omega$, then Duplicator can go to p_2 immediately).

On the other hand, like in the fair case and unlike in the direct and backward cases, multipebble simulation is included in fixed-word delayed simulation.

Theorem 3.4.3. *Multipebble delayed simulation is included in fixed-word delayed simulation. I.e., for $k > 0$ and states q and s,*

$$q \sqsubseteq^{k\text{-de}} s \quad \textit{implies} \quad q \sqsubseteq^{\text{de}}_{fx} s \quad \textit{implies} \quad q \subseteq^{\text{de}} s$$

This is a nontrivial result; its proof is postponed until Section 3.5, where we discuss *fixed-word multipebble delayed simulation*. As a consequence, the multipebble delayed simulation hierarchy of [42] (cf. Section 3.2) is entirely contained in fixed-word delayed simulation:

$$\sqsubseteq^{1\text{-de}} \subseteq \sqsubseteq^{2\text{-de}} \subseteq \sqsubseteq^{3\text{-de}} \subseteq \cdots \subseteq \sqsubseteq^{\text{de}}_{\text{fx}}$$

By the next example, the inclusion is strict.

Example 3.4.4 - Multipebble delayed simulation is strictly included in fixed-word delayed simulation

We have shown in Example 3.3.1 that $q_0 \not\sqsubseteq^{k\text{-de}} s_0$ for any $k > 0$, although $q_0 \subseteq^{\text{de}} s_0$. In fact, since the automaton on the left is deterministic, even if Spoiler reveals just the input word, this in fact uncovers the unique induced path as well. Therefore, $q_0 \sqsubseteq^{\text{de}}_{\text{fx}} s_0$.

The "big picture" about the relationship between ordinary simulation, multipebble simulation, fixed-word simulation and containment is shown in Figure 3.3. All inclusions follow immediately from the definitions, except for the inclusion of multipebble delayed simulation in fixed-word delayed simulation, which is marked by "!" (cf. Theorem 3.4.3). All containments are strict, unless otherwise noted by "=".

Remark 3.4.4. As another consequence of the inclusions above, notice that direct containment is included in fixed-word delayed simulation. We find this interesting, since their definitions are incomparable: while in direct containment Duplicator is stronger than in delayed fixed-word simulation w.r.t. path quantification, it is the other way around w.r.t. the winning condition.

3.4.4 Complexity

We examine the computational complexity of computing fixed-word simulation. Let q and s be two states in a given NBA $\mathcal{Q}$. We reduce the problem of checking $q \sqsubseteq^{x}_{\text{fx}} s$ to

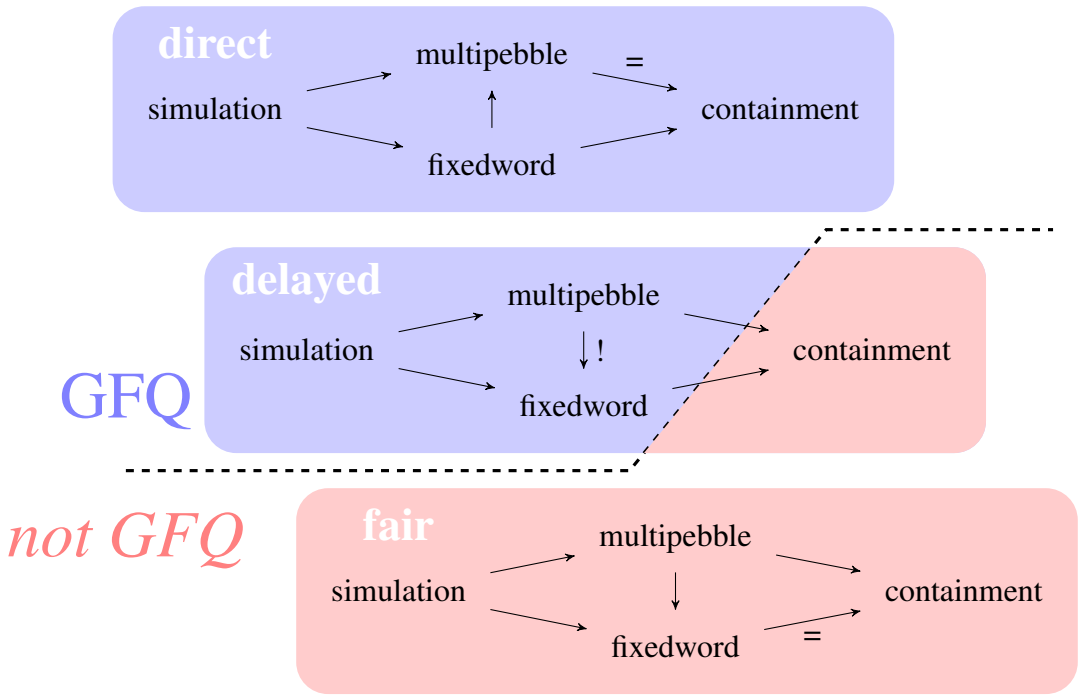


Figure 3.3: Pebbles, fixed-words, and containment

the universality problem of a certain *alternating automaton* $\mathcal{A}$, obtained as a suitable product automaton from $\mathcal{Q}$. The idea is to design $\mathcal{A}$ in order to accept exactly those words w s.t. $q \sqsubseteq^x_w s$ (except for a minor modification). Then, by the definition of $\sqsubseteq^x_{\text{fx}}$, it suffices to check whether $\mathcal{A}$ is universal—i.e., $\mathcal{L}(\mathcal{A}) = \Sigma^\omega$ (or $\mathcal{L}(\mathcal{A}) = \Sigma^*$ in the case of fixed-word backward simulation).

Since the universality problem for the models we consider is computable in PSPACE, it follows that fixed-word simulations are in PSPACE as well. Moreover, very simple reductions from well-known (and very similar) PSPACE-hard problems can be established for the converse direction, giving the following result.

Theorem 3.4.5. *Checking Fixed-word simulations is PSPACE-complete.*

We study the complexity of multipebble simulations in Section 5.10 (in the more general context of alternating automata).

The details of the product automaton $\mathcal{A}^x_{q,s}$ depend on x. Generally, $\mathcal{A}^x_{q,s}$ is designed such that

$$\mathcal{L}(\mathcal{A}^x_{q,s}) = \{a_0\Sigma a_1\Sigma a_2 \cdots \mid w = a_0a_1a_2\cdots \text{ and } q \sqsubseteq^x_w s\} \tag{3.1}$$

The presence of the extra padding symbols in Σ at the odd positions is due to the kind of alternating automata that we use. (It is possible to use alternating automata with

a more compact transition relation in the style of [115], which would eliminate those padding symbols. We choose a different alternation model in coherence with the second part of this thesis.)

Remark 3.4.6 (Ultimately periodic counterexamples). As a by-product of the characterization of Equation (3.1), the set of words w s.t. $q \sqsubseteq^x_w s$ holds is a regular language. Since regular languages are closed under complement, also the set of words w s.t. $q \not\sqsubseteq^x_w s$ is regular. This means that, if there exists a word w s.t. Spoiler wins the w-simulation game, then there exists an ultimately periodic word $w' = uv^\omega$ with the same property.

Below, we instantiate the construction for the different $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}, \mathrm{bw}\}$.

Fixed-word fair simulation $\mathcal{A}^{\mathrm{f}}_{q,s}$ is an alternating parity automaton. Universal states in $\mathcal{A}^x_{q,s}$ are of the form $\langle p, r\rangle$, while existential states are of the form $\langle p, r, a, p'\rangle$. The initial state is $\langle q, s\rangle$, and transitions are as follows: $\langle p, r\rangle \xrightarrow{a} \langle p, r, a, p'\rangle$ iff $p \xrightarrow{a} p'$, and $\langle p, r, a, p'\rangle \xrightarrow{\Sigma} \langle p', r'\rangle$ iff $r \xrightarrow{a} r'$. The acceptance condition is a three color min-parity condition: $\langle p, r\rangle$ has color 0 if $r \in F$, and has color 1 if $p \in F$ and $r \notin F$; every other state has color 2.

The lower bound is trivial: As we have seen, fixed-word fair simulation coincides with fair containment, which is already known to be PSPACE-hard [80]. The upper bound above gives an alternative proof of membership in PSPACE for fair containment.

Fixed-word direct and backward simulations For $x \in \{\mathrm{di}, \mathrm{bw}\}$, the automaton $\mathcal{A}^x_{q,s}$ is an alternating safety automaton. States are the same as in the fair case, including the initial state. The acceptance condition requires that only *safe* states are visited.

For the direct case, $\mathcal{A}^{\mathrm{di}}_{q,s}$ is an automaton over infinite words. Transitions are as in the fair case. State $\langle p, r\rangle$ is not safe if $p \in F$ and $r \notin F$; every other state is safe.

For the backward case, $\mathcal{A}^{\mathrm{bw}}_{q,s}$ is an automaton over finite words. Transitions are now taken backwards: For example, $\langle p, r\rangle \xrightarrow{a} \langle p, r, a, p'\rangle$ iff $p \xleftarrow{a} p'$. State $\langle p, r\rangle$ is not safe if $p \in F$ and $r \notin F$, or $p \in I$ and $r \notin I$; every other state is safe.

For the lower bound, notice that we could also make unsafe states unreachable, by removing all incoming transitions, and maintain that accepting runs are exactly the infinite ones. In this case, we are interested in the *trace language* of $\mathcal{A}^x_{q,s}$, which can be obtained by making every state accepting. It is well known that trace universality is PSPACE-hard.

Fixed-word delayed simulation The idea is to add an obligation bit recording whether Duplicator has any pending constraint to visit an accepting state [43]. Initially the bit is 0, and it is set to 1 whenever Spoiler is accepting; a reset to 0 can occur afterwards, if and when Duplicator visits an accepting state.

$\mathcal{A}^{\text{de}}_{q,s}$ is an alternating Büchi automaton. Universal states are of the form $\langle p, r, b\rangle$, and existential states are of the form $\langle p, r, b, a, p'\rangle$, where $b \in \{0, 1\}$ is the obligation bit. The initial state is $\langle q, s, 0\rangle$, and transitions are as follows: $\langle p, r, b\rangle \xrightarrow{a} \langle p, r, b, a, p'\rangle$ iff $p \xrightarrow{a} p'$, and $\langle p, r, b, a, p'\rangle \xrightarrow{\Sigma} \langle p', r', b'\rangle$ iff $r \xrightarrow{a} r'$ and b' is 0 if $r \in F$, is 1 if $p \in F$ and $r \notin F$, and is b otherwise. Accepting states are those of the form $\langle p, r, 0\rangle$, where no obligation is pending.

For the lower bound, notice that an automaton $\mathcal{Q}$ is universal iff $\mathcal{U} \sqsubseteq^{\text{de}}_{\text{fx}} \mathcal{Q}$, where $\mathcal{U}$ is the trivial, universal one-state automaton with an accepting Σ-loop. Since universality is PSPACE-hard already for NBAs [80], the same complexity applies to $\sqsubseteq^{\text{de}}_{\text{fx}}$.

3.5 Fixed-word multipebble delayed simulation

With Theorem 3.4.2 we have established that fixed-word delayed simulation is GFQ. An immediate question is whether we can find other natural GFQ preorders between fixed-word delayed simulation and containment. A natural idea is to add a multipebble facility on top of fixed-word simulation: In *fixed-word multipebble simulation* Duplicator has multiple pebbles *and* knows the input word in advance. We show that, perhaps surprisingly, fixed-word multipebble delayed simulation is contained in ordinary (i.e., 1-pebble) fixed-word delayed simulation (cf. Theorem 3.5.1). Since fixed-word multipebble simulation is, by definition, coarser than both multipebble and fixed-word simulations, this also proves Theorem 3.4.3 from Section 3.4—i.e, it shows that multipebble delayed simulation is included in fixed-word delayed simulation.

We now formally define fixed-word multipebble delayed simulation. Let $k > 0$ and $w = a_0 a_1 \cdots \in \Sigma^\omega$. In the k-multipebble delayed w-simulation game $G^{k\text{-de}}_w(q, s)$ the set of positions of Spoiler is $Q \times 2^Q \times \omega$, the set of positions of Duplicator is $Q \times 2^Q \times Q \times \omega$, the initial position is $\langle q, \{s\}, 0\rangle$, and transitions are as follows: $(\langle q, \mathrm{s}, i\rangle, \langle q, \mathrm{s}, q', i\rangle) \in \Gamma_0$ iff $q \xrightarrow{a_i} q'$, and $(\langle q, \mathrm{s}, q', i\rangle, \langle q', \mathrm{s}', i+1\rangle) \in \Gamma_1$ iff $\mathrm{s} \overset{a_i}{\Longrightarrow} \mathrm{s}'$ and $|\mathrm{s}'| \leq k$. Given an infinite play $\pi = \langle q_0, \mathrm{s}_0, 0\rangle\langle q_1, \mathrm{s}_1, 1\rangle \cdots$, Duplicator is winning iff, whenever $q_i \in F$, there exists $j \geq i$ s.t. $\mathrm{s}_i \Longrightarrow^*_F \mathrm{s}_j$. We write $q \sqsubseteq^{k\text{-de}}_w s$ iff Duplicator wins $G^{k\text{-de}}_w(q, s)$, and we write $q \sqsubseteq^{k\text{-de}}_{\text{fx}} s$ iff $\forall w \in \Sigma^\omega \cdot q \sqsubseteq^{k\text{-de}}_w s$.

Clearly, pebble simulations induce a non-decreasing hierarchy: $\sqsubseteq^{1\text{-de}}_{\text{fx}} \subseteq \sqsubseteq^{2\text{-de}}_{\text{fx}} \subseteq \cdots$.

We establish that the hierarchy actually collapses to the first level, which is just $\sqsubseteq^{\mathrm{de}}_{\mathrm{fx}}$.

Theorem 3.5.1. *For $k > 0$, k-pebble fixed-word delayed simulation equals 1-pebble fixed-word delayed simulation.*

This result is non-trivial. The delayed winning condition requires reasoning not only about the *possibility* to visit accepting states in the future, but also about exactly *when* such a visit occurs. The proof uses a natural ranking argument inspired by [81], with the notable difference that our ranks are *ordinals* ($\leq \omega^2$), instead of just natural numbers. We need ordinals to represent how long a player can delay visiting accepting states, and how this events nest with each other.

Pebbles, lookahead, and containment We give a brief interpretation of Theorem 3.5.1 in terms of *degree of lookahead*. We say that Duplicator has lookahead $l \in \omega \cup \{\omega\}$ iff during the simulation game she has the ability of waiting up to l steps of Spoiler before making a move. Higher lookahead clearly gives more power to Duplicator.

Lookahead is related to multiple pebbles in a simple way. When Duplicator uses multiple pebbles, she can "hedge her bets" by moving pebbles to several successors. This allows Duplicator to delay committing to any particular choice by arbitrarily many steps: Multiple pebbles subsume any *finite* lookahead l. By Theorem 3.5.1, pebbles are redundant in fixed-word delayed simulation. This means that, once Duplicator knows the input word in advance, there is no difference between knowing only the next step by Spoiler, or the next l steps for any finite $l > 1$. Therefore, the finite lookahead hierarchy collapses to the $l = 1$ level at the GFQ preorder $\sqsubseteq^{\mathrm{de}}_{\mathrm{fx}}$.

However, in the case of infinite $l = \omega$ lookahead, Duplicator can wait until Spoiler has completed her infinite path. In this case, we recover delayed containment $\subseteq^{\mathrm{de}}$, which is not GFQ by Example 3.3.2. Thus,

$\sqsubseteq^{\mathrm{de}}_{fx}$ *is the coarsest GFQ relation included in* $\subseteq^{\mathrm{de}}$ *w.r.t. to the degree of lookahead.*

The rest of the chapter is devoted to the proof of Theorem 3.5.1.

3.6 Fixed-words and ranks

In this section we prove Theorem 3.5.1 from Section 3.5.

3.6.1 Overview

Fix an infinite word $w = a_0 a_1 \cdots \in \Sigma^\omega$. We associate to any state q an ordinal number $\mathsf{rank}(q)$ (depending on w), which, in some sense, measures the "power" of a player in a delayed w-simulation game starting from q. Then we show that, if Duplicator has a better rank than Spoiler's, then she wins the 1-pebble game:

Lemma 3.6.1. *Let $w \in \Sigma^\omega$. If $\mathsf{rank}(q) \leq \mathsf{rank}(s)$, then $q \sqsubseteq_w^{\mathrm{de}} s$.*

Therefore, ranks are sufficient witnesses for showing that Duplicator is winning. Ranks are also *necessary*, in a rather strong sense: We show that, if Duplicator wins the w-simulation game, *even using $k > 1$ pebbles*, then she has a better rank:

Lemma 3.6.2. *Let $w \in \Sigma^\omega$ and $k > 0$. If $q \sqsubseteq_w^{k\text{-de}} s$, then $\mathsf{rank}(q) \leq \mathsf{rank}(s)$.*

By combining these two results, we are able to show that Duplicator does not need more than one pebble.

Proof of Theorem 3.5.1. By the previous two lemmas, we get

$$q \sqsubseteq_{\mathrm{fx}}^{\mathrm{de}} s \implies q \sqsubseteq_{\mathrm{fx}}^{k\text{-de}} s \implies \forall w \in \Sigma^\omega \cdot \mathsf{rank}(q) \leq \mathsf{rank}(s) \implies q \sqsubseteq_{\mathrm{fx}}^{\mathrm{de}} s\,,$$

where the first implication holds since fixed-word simulation (which, by definition, is the same as 1-pebble fixed-word simulation), obviously implies k-pebble fixed-word simulation. □

3.6.2 Preliminaries

Ordinals Let ω be the least infinite ordinal, and let ω_1 be the set of all countable ordinals. We denote arbitrary ordinals by α or β, and limit ordinals by λ or μ. We consider 0 *not* as a limit ordinal.

We define a predecessor and a floor operation on ordinals. For an ordinal α, its *predecessor* $\alpha - 1$ is either α itself if α is a limit ordinal, or β if $\alpha = \beta + 1$ for some β; its *floor*

$$\lfloor \alpha \rfloor := \sup_{\substack{\text{limit ordinal } \lambda: \\ \lambda < \alpha}} \lambda$$

is the largest limit ordinal strictly smaller than α if it exists, and zero otherwise. Notice that, for $0 < \alpha < \omega^\omega$, $\lfloor \alpha \rfloor < \alpha$.

Trees Let $[n] = \{0, 1, \ldots, n-1\}$. A tree domain is a non-empty, prefix-closed subset V of $[n]^*$. With $<_{\text{prf}}$ we denote the prefix order on words; if $u <_{\text{prf}} u'$, then u' is called a descendant of u, and u is an ancestor of u'. In particular, if $u' = uc$ for some $c \in \omega$, then u' is a child of u. A (labelled) L-tree is a pair (V, t), where V is a tree domain and $t : V \mapsto L$ is a mapping which assigns a label from L to any node in t.

3.6.3 The ranking construction

We now show how to assign ranks to states. The following presentation follows [81]. Let $\mathcal{Q} = (Q, \Sigma, I, \Delta, F)$ be an automaton, and let n be the cardinality of Q. Given an infinite word $w = a_0 a_1 \cdots \in \Sigma^\omega$, we associate to any state $p \in Q$ a tree domain T_p^w and a $\mathcal{Q}$-tree (T_p^w, t_p^w), the *unravelling* of $\mathcal{Q}$ from p while reading w, by applying the following two rules:

- $\varepsilon \in T_p^w$ and $t_p^w(\epsilon) = p$.
- If u has length i, $u \in T_p^w$, $t_p^w(u) = q$ and $\Delta(q, a_i) = \{q'_0, q'_1, \ldots, q'_{k-1}\}$, then, for any j s.t. $0 \leq j < k$, $uj \in T_p^w$ and $t_p^w(uj) = q'_j$.

It is easy to see that if two nodes at the same level have the same label, then they generate isomorphic subtrees. Therefore, we can "compress" (T_p^w, t_p^w) into an infinite DAG $G_p^w = (V, E)$, where $V \subseteq Q \times \omega$ is such that $\langle q, l \rangle \in V$ iff there exists a node in (T_p^w, t_p^w) at level l with label q, and $(\langle q, l \rangle, \langle q', l+1 \rangle) \in E$ iff there exist two nodes u and u', labelled with q and q', respectively, s.t. u' is a child of u in (T_p^w, t_p^w). We say that a vertex $\langle q, l \rangle$ is *accepting* iff $q \in F$.

For any sub-DAG $G \subseteq G_p^w$, we say that a vertex $\langle q, l \rangle$ is a *dead end* in G iff it has no successor in G, and we say that it is *inert* in G iff no accepting vertex can be reached from $\langle q, l \rangle$ in G. In particular, an inert vertex is not accepting. For example, if the automaton is complete, then G_p^w has no dead end, and, if w accepted from p, then there exists an infinite branch with no inert vertex in it. The *girth* of G at level l is the maximal number of vertices of the form $\langle q, l \rangle$ in G, and the *width* of G is the maximal girth over infinitely many levels.

We build a non-increasing transfinite sequence of DAGs $\{G_\alpha \mid \alpha < \omega_1\}$ as follows:

$$\begin{aligned} G_0 &= G_p^w \\ G_{\alpha+1} &= G_\alpha \setminus \{\langle q, l\rangle \mid \langle q, l\rangle \text{ is a dead end in } G_\alpha\} \\ G_\lambda &= H_\lambda \setminus \{\langle q, l\rangle \mid \langle q, l\rangle \text{ is inert in } H_\lambda\} \\ &\text{where, for any ordinal } \beta > 0,\ H_\beta = \bigcap_{\alpha<\beta} G_\alpha \end{aligned}$$

For example, H_ω is obtained from G_0 by removing all possible dead ends. In general, H_λ does not have any dead end, and G_λ is obtained from the latter by removing inert vertices therein. This operation possibly introduces new dead ends, and the whole construction is iterated over higher ordinals. Notice that, for any ordinal α, $H_{\alpha+1} = G_\alpha$. We get an chain of smaller and smaller DAGs:

$$\alpha \leq \beta \quad \text{implies} \quad G_\alpha \supseteq G_\beta$$

Running example

Consider the automaton $\mathcal{Q}$ in Figure 3.4(a). (It is the same automaton $\mathcal{Q}$ as in Example 3.3.2.) Let $w = a^\omega$ be the unique input word, and consider the run tree from state p_0. The resulting DAG G_0 is shown in Figure 3.4(b). Accepting vertices are underlined. Since G_0 has no dead ends, $G_0 = G_1 = G_2 = \cdots = H_\omega$.

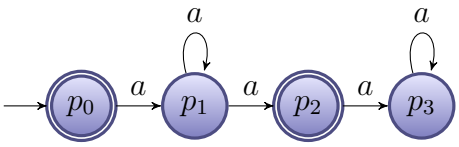
From level three on, vertices labelled with p_3 have no accepting descendant, and are thus inert. By removing those, we get the DAG G_ω in Figure 3.4(c). This pruning operation introduces dead ends in G_ω, which are removed in $G_{\omega+1}$. Since the latter does not have dead ends, we get

$$G_\omega \supset G_{\omega+1} = G_{\omega+2} = \cdots = H_{\omega\cdot 2}$$

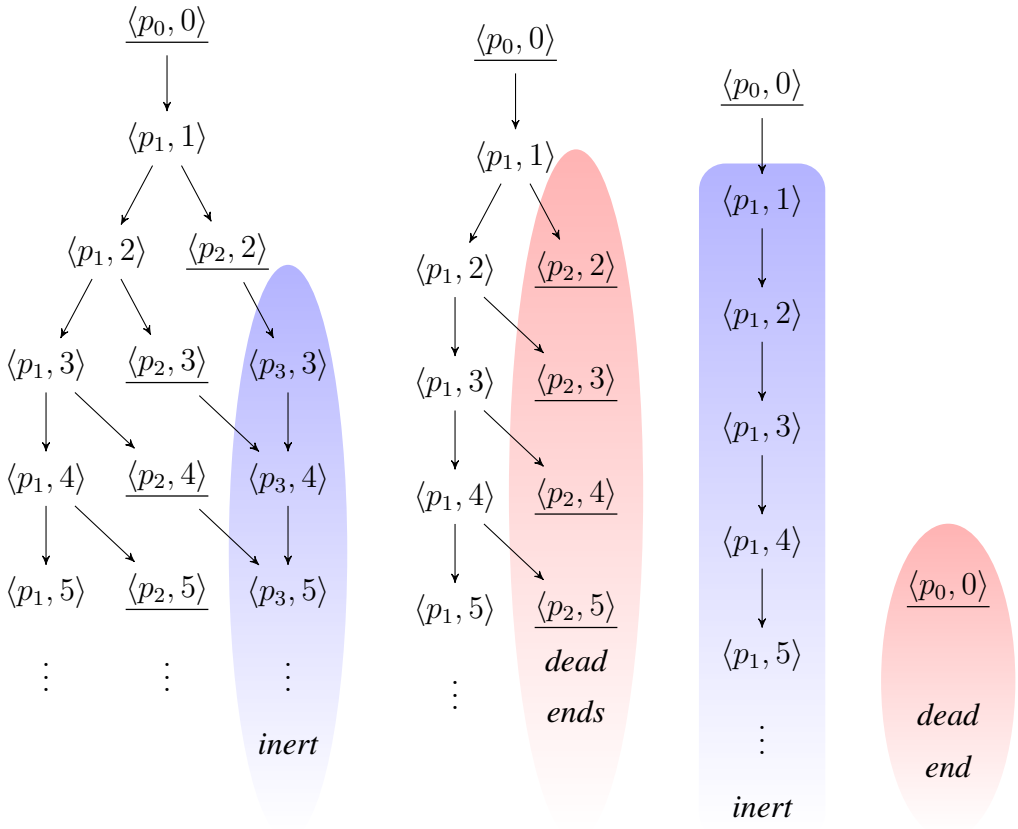
The limit DAG $H_{\omega\cdot 2}$ is shown in Figure 3.4(d).

Finally, all vertices in $H_{\omega\cdot 2}$ are inert, except the root. Therefore, only the root survives in $G_{\omega\cdot 2}$; see Figure 3.4(e). Finally, even this last vertex gets removed next, ending the hierarchy:

$$G_{\omega\cdot 2} \supset G_{\omega\cdot 2+1} = G_{\omega\cdot 2+2} = \cdots = H_{\omega\cdot 3} = \emptyset$$



(a) Automaton $\mathcal{Q}$ (from Example 3.3.2)



(b) The DAG G_0 (c) The DAG G_ω (d) $H_{\omega\cdot 2}$ (e) $G_{\omega\cdot 2}$

Figure 3.4: A running example

We argue that the chain eventually stabilizes, and the rank of a vertex is defined to be the stabilization level. For a limit ordinal λ, when going from H_λ to G_λ there are two possible cases:

a) No vertex in H_λ is inert and $G_\lambda = H_\lambda$. (H_λ might even be empty in this case.)

b) Some vertex in H_λ is inert and $G_\lambda \supset H_\lambda$.

In the first case, the chain stabilizes at ordinal λ. In the second case, one can show that, by König's Lemma, an infinite path is removed from the graph. (This path can start at an arbitrarily deep level down the tree.) Intuitively, one reasons as follows: Let $\langle q, l\rangle$ be an inert vertex in H_λ (which exists by assumption), and recall that H_λ has no dead end by construction. Therefore, $\langle q, l\rangle$ has at least one successor $\langle q', l+1\rangle$. The latter is inert by definition, and the procedure can be iterated in order to build longer and longer paths of inert vertices. The limit of these paths is an infinite path of inert vertices rooted at $\langle q, l\rangle$, all of which are removed in G_λ.

Therefore, in the second case, the width of G_λ is strictly less than the width of H_λ. As a consequence, since the width of G_0 is bounded by n (and thus by ω), the hierarchy stabilizes at most at ordinal ω^2:

$$G_0 \supset G_1 \supset \cdots \supset H_\omega \supset G_\omega \supset G_{\omega+1} \supset \cdots \supset H_{\omega^2} = G_{\omega^2} = G_{\omega^2+1} = \cdots$$

Thus, each vertex not in H_{ω^2} is either a dead end in $G_\alpha = H_{\alpha+1}$ or inert in H_λ. In the former case $\langle q, l\rangle$ is in $H_{\alpha+1} \setminus G_{\alpha+1}$, whereas in the latter case $\langle q, l\rangle$ is in $H_\lambda \setminus G_\lambda$: In both cases, there exists a unique ordinal $\alpha < \omega^2$ s.t. $\langle q, l\rangle$ is in $H_\alpha \setminus G_\alpha$, and vertex $\langle q, l\rangle$ receives[2] *rank* α.Otherwise, if a vertex is in H_{ω^2}, then it will never be removed from the chain. In this latter case, we assign rank ω^2:

$$\mathsf{rank}(q, l) := \begin{cases} \text{the unique } \alpha \text{ s.t. } \langle q, l\rangle \in H_\alpha \setminus G_\alpha & \text{if } \langle q, l\rangle \notin H_{\omega^2} \\ \omega^2 & \text{if } \langle q, l\rangle \in H_{\omega^2} \end{cases}$$

When $\langle q, l\rangle$ has rank α, we also write $\langle q, l\rangle : \alpha$.

Running example

Vertices of level $l \geq 3$ of the form $\langle p_3, l\rangle$ are in $H_\omega \setminus G_\omega$, therefore they get rank ω, since

[2] Alternatively, one could define α to be the maximal ordinal in H_α.

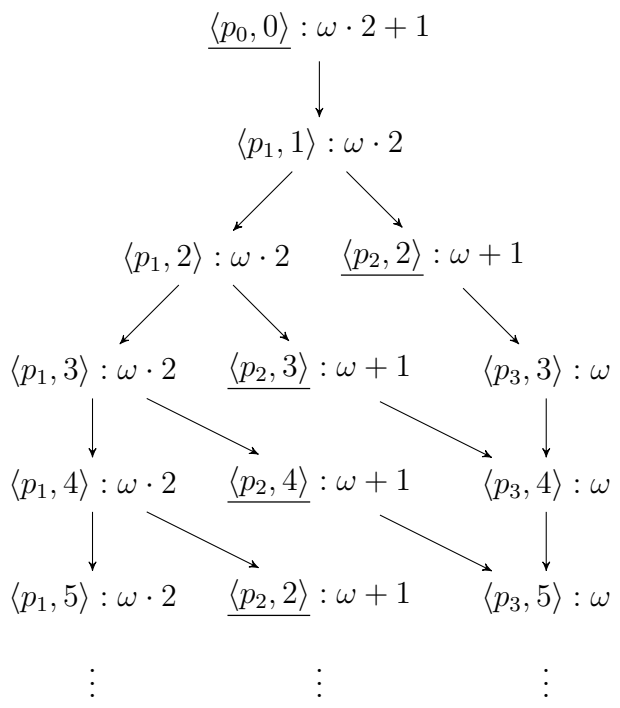


Figure 3.5: Ranks of the running example

this is the largest ordinal α s.t. $\langle p_3, l\rangle \in H_\alpha$. Vertices of level $l \geq 2$ of the form $\langle p_2, l\rangle$ are in $G_\omega \setminus G_{\omega+1}$, that is, in $H_{\omega+1} \setminus G_{\omega+1}$, therefore they get rank $\omega + 1$. Similarly, vertices of level $l \geq 1$ of the form $\langle p_1, l\rangle$ are in $H_{\omega\cdot 2} \setminus G_{\omega\cdot 2}$ and thus they get rank $\omega \cdot 2$. Finally, the root vertex $\langle p_0, 0\rangle$ is in $G_{\omega\cdot 2} \setminus G_{\omega\cdot 2+1}$ and consequently gets rank $\omega \cdot 2 + 1$. The ranks of all vertices in $G^w_{p_0}$ are shown in Figure 3.5.

Vertices of rank ω^2 are not dead ends, and they can always reach an accepting vertex of the same rank ω^2. Therefore, vertices of rank ω^2 are on fair paths in G^w_p.

Lemma 3.6.3. *A vertex $\langle q, l\rangle$ has rank ω^2 iff there exists a fair path from $\langle q, l\rangle$ in G^w_p.*

Remark 3.6.4. In the construction of [81], all dead ends and all vertices that can only reach dead ends are removed in just one step, and these vertices are collectively called *endangered*. Here, we need to remove endangered vertices from the bottom-up. This is necessary since the rank needs to take into account the maximal number of steps that a player can perform before visiting an accepting state. H_λ is obtained by removing all endangered vertices, while in [81] this information is not needed and is abstracted away.

3.6.4 Important properties of ranks $< \omega^2$

In this section we state and prove several properties of ranks, that will be used later in the proofs of Lemmas 3.6.1 and 3.6.2. All the properties in this section are stated under the global assumption that *no vertex gets rank* ω^2. In this case, we recall that the rank of a vertex $\langle q, l\rangle$ is the unique α s.t. $\langle q, l\rangle \in H_\alpha \setminus G_\alpha$. (The case where vertices have rank ω^2 is simple, and it is treated separately in the two main Lemmas 3.6.1 and 3.6.2.)

The following lemma relates accepting vertices and successor ranks.

Lemma 3.6.5. *If a vertex $\langle q, l\rangle$ is accepting, then it has successor ordinal rank $\alpha + 1$. Furthermore, if it has rank $\lambda + 1$, for λ a limit ordinal, then it is accepting.*

Proof. If $\langle q, l\rangle$ is accepting, then it is never inert. Consequently, if $\langle q, l\rangle \in H_\lambda$, then $\langle q, l\rangle \in G_\lambda$. The only possibility left is that $\langle q, l\rangle$ is a dead end in G_α, for some α: $\langle q, l\rangle \in H_{\alpha+1} \setminus G_{\alpha+1}$ (since $H_{\alpha+1} = G_\alpha$) and $\mathsf{rank}(q, l) = \alpha + 1$.

For the second part, assume $\langle q, l\rangle : \lambda+1$, i.e., $\langle q, l\rangle \in G_\lambda \setminus G_{\lambda+1}$ (since $H_{\lambda+1} = G_\lambda$). $\langle q, l\rangle$ is a dead end in G_λ. Since $G_\lambda \subseteq H_\lambda$, $\langle q, l\rangle$ is in H_λ as well. But H_λ has no dead ends by construction, therefore $\langle q, l\rangle$ has at least one successor $\langle q', l + 1\rangle$ in H_λ. Consider any such successor $\langle q', l + 1\rangle$. Since $\langle q, l\rangle$ is a dead end in G_λ, $\langle q', l + 1\rangle$ is not in G_λ. That is, $\langle q', l + 1\rangle \in H_\lambda \setminus G_\lambda$. By definition, $\langle q', l + 1\rangle$ is inert in H_λ. Therefore, $\langle q, l\rangle$ has only inert successors in H_λ.

By contradiction, assume that $\langle q, l\rangle$ is not accepting. Since it has only inert successors in H_λ, it is itself inert in H_λ, contradicting $\langle q, l\rangle \in G_\lambda$. Thus, $\langle q, l\rangle$ is accepting. □

We say that a vertex $\langle q', l + 1\rangle$ is a *maximal successor* of $\langle q, l\rangle$ if its rank is maximal amongst all successors of $\langle q, l\rangle$. The following lemma states that ranks are non-increasing along paths, and that intermediate ranks cannot be skipped on every successor.

Lemma 3.6.6. *Let vertex $\langle q, l\rangle$ have rank α. Then,*

a) Every successor $\langle q', l + 1\rangle$ has rank at most $\alpha - 1$.

b) There exists a maximal successor attaining rank $\alpha - 1$.

As a direct consequence,

c) Every node $\langle q', l'\rangle$ reachable from $\langle q, l\rangle$ has a smaller rank $\alpha' \leq \alpha$.

Proof. We split the proof into two cases, depending on whether α is a successor or limit ordinal.

- *First case:* Let α be a successor ordinal $\beta + 1$. Consequently, $\alpha - 1 = \beta$. Then, $\langle q, l\rangle \in G_\beta \setminus G_{\beta+1}$. $\langle q, l\rangle$ is a dead end in G_β and no successor $\langle q', l+1\rangle$ is in G_β. Thus, any successor $\langle q', l+1\rangle$ has rank $\leq \beta$. This shows Point a). For Point b), we show that at least one successor has rank exactly equal to β. Let $\beta^* \leq \beta$ be the maximum rank amongst $\langle q, l\rangle$'s successors. (It exists since there are finitely many successors.) We show $\beta^* = \beta$. By maximality, no successor $\langle q', l+1\rangle$ is in G_{β^*}. Since $G_\beta \subseteq G_{\beta^*}$, $\langle q, l\rangle$ is a dead end in G_{β^*}. Thus, $\langle q, l\rangle \in G_{\beta^*} \setminus G_{\beta^*+1}$, and $\langle q, l\rangle : \beta^* + 1$. By assumption, $\langle q, l\rangle : \beta + 1$ Therefore, $\beta^* = \beta$.

- *Second case:* Let α be a limit ordinal λ. Consequently, $\alpha - 1 = \lambda$. $\langle q, l\rangle$ is inert in H_λ. Let $\langle q', l+1\rangle$ be any successor of $\langle q, l\rangle$ (in G_p^w). If $\langle q', l+1\rangle$ is not in H_λ, then, since $G_\lambda \subseteq H_\lambda$, $\langle q', l+1\rangle$ is not in G_λ either. Thus, $\langle q', l+1\rangle :\leq \lambda$ in this case. Otherwise, let $\langle q', l+1\rangle$ be in H_λ. Since $\langle q, l\rangle$ is inert in H_λ, $\langle q', l+1\rangle$ is inert in H_λ. Therefore, $\langle q', l+1\rangle : \lambda$. This proves Point a). For Point b), a successor of rank λ always exists, since, by construction, H_λ does not contain dead ends. □

A sequence $\langle q_0, l\rangle\langle q_1, l+1\rangle \cdots \langle q_h, l+h\rangle$ is a *maximal path* if, for any $0 \leq k < h$, $\langle q_{k+1}, l+k+1\rangle$ is a maximal successor of $\langle q_k, l+k\rangle$.

Lemma 3.6.7. *If a vertex $\langle q_0, l\rangle$ has a successor ordinal rank $\alpha + 1$, then there exists a maximal path $\langle q_0, l\rangle\langle q_1, l+1\rangle \cdots \langle q_h, l+h\rangle$ ending in $\langle q_h, l+h\rangle : \lambda + 1$ with $\lfloor \alpha + 1\rfloor \leq \lambda$.*

Proof. We proceed by ordinal induction. If α is a limit ordinal λ or zero, the claim holds immediately: Take $h = 0$; clearly, $\lambda = \lfloor \lambda + 1\rfloor$.

Otherwise, let α be a successor ordinal $\beta + 1$. $\langle q_0, l\rangle$ has rank $\alpha + 1 = (\beta + 1) + 1$. By Lemma 3.6.6(b), $\langle q_0, l\rangle$ has a maximal successor $\langle q_1, l+1\rangle : \beta + 1 = \alpha$. By induction hypothesis, there exists a maximal path $\langle q_1, l+1\rangle \cdots \langle q_h, l+h\rangle$ with $h > 0$, ending in $\langle q_h, l+h\rangle : \lambda + 1$ with $\lfloor \beta + 1\rfloor \leq \lambda$. But $\lfloor (\beta + 1) + 1\rfloor = \lfloor \beta + 1\rfloor$, since $\{$ limit ordinal $\lambda \mid \lambda < (\beta + 1) + 1\} = \{$ limit ordinal $\lambda \mid \lambda < \beta + 1\}$. Therefore, $\lfloor \alpha + 1\rfloor = \lfloor (\beta + 1) + 1\rfloor = \lfloor \beta + 1\rfloor \leq \lambda$. □

The following lemma says that vertices with a limit rank have some descendant (not necessarily a direct successor) of strictly smaller rank.

Lemma 3.6.8. *If a vertex $\langle q_0, l\rangle$ has a limit ordinal rank λ s.t. $\omega < \lambda < \omega^2$, then there exists a (non-maximal) path $\langle q_0, l\rangle\langle q_1, l+1\rangle \cdots \langle q_h, l+h\rangle$ with $h > 0$ ending in $\langle q_h, l+h\rangle : \alpha + 1$ with $\lfloor \lambda\rfloor \leq \alpha$.*

Proof. Let $\langle q_0, l\rangle : \lambda$ with $\omega < \lambda < \omega^2$. By contradiction, assume $\langle q_0, l\rangle$ has no descendant $\langle q', l'\rangle : \alpha + 1$ with $\lfloor\lambda\rfloor \leq \alpha$. Then, all descendants $\langle q', l'\rangle : \alpha + 1$ have $\alpha < \lfloor\lambda\rfloor$. Since $\lfloor\lambda\rfloor$ is a limit ordinal by definition, $\alpha + 1 < \lfloor\lambda\rfloor$. By definition, $\langle q_0, l\rangle$ is inert in H_λ. Since $\lfloor\lambda\rfloor \leq \lambda$ by definition of the floor operation, $H_\lambda \subseteq H_{\lfloor\lambda\rfloor}$. ($H_{\lfloor\lambda\rfloor}$ is defined since $\lambda > \omega$ by assumption, thus, $\lfloor\lambda\rfloor \geq \omega$.) Therefore, $\langle q_0, l\rangle$ is in $H_{\lfloor\lambda\rfloor}$. We show that $\langle q_0, l\rangle$ is inert in $H_{\lfloor\lambda\rfloor}$ as well. To this end, we show that any vertex reachable from $\langle q_0, l\rangle$ in $H_{\lfloor\lambda\rfloor}$ is non-accepting. Let $\langle q', l'\rangle$ be a descendant of $\langle q_0, l\rangle$ in $H_{\lfloor\lambda\rfloor}$. If $\langle q', l'\rangle$ were accepting, then, by Lemma 3.6.5, it would have successor ordinal rank $\alpha + 1$. By assumption, $\alpha + 1 < \lfloor\lambda\rfloor$, thus $\langle q', l'\rangle$ is not in $H_{\lfloor\lambda\rfloor}$.

Therefore, $\langle q_0, l\rangle$ is inert in $H_{\lfloor\lambda\rfloor}$. But this is a contradiction, since $\langle q_0, l\rangle$ would get rank $\lfloor\lambda\rfloor < \lambda$. □

3.6.5 The two directions of Theorem 3.5.1

The first direction is easier. It says that if Duplicator has a better rank than Spoiler, then Duplicator wins the w-simulation game with just one pebble. Thus, ranks are sufficient for simulation.

Lemma 3.6.1. Let $w \in \Sigma^\omega$. If $\mathsf{rank}(q_0, 0) \leq \mathsf{rank}(s_0, 0)$, then $q_0 \sqsubseteq^{\mathrm{de}}_w s_0$.

Proof. Assume $\mathsf{rank}_{q_0}(q_0, 0) \leq \mathsf{rank}_{s_0}(s_0, 0)$. We show that Duplicator has a winning strategy in $G^{\mathrm{de}}_w(q_0, s_0)$. For any round i, let $\langle q_i, s_i\rangle$ be the current configuration of the simulation game (we omit the third component for simplicity), and let the rank of Spoiler and Duplicator at round i be $\mathsf{rank}_{q_0}(q_i, i)$ and $\mathsf{rank}_{s_0}(s_i, i)$, respectively. If Duplicator's rank is ω^2, then she will just play a fair path, which exists by Lemma 3.6.3. Otherwise, both ranks are $< \omega^2$, and we proceed by applying the properties from Section 3.6.4.

Intuitively, Duplicator wins by ensuring both a *safety* and a *liveness* condition. The safety condition requires Duplicator to always preserve the ordering between ranks: For any round i,

$$\mathsf{rank}_{q_0}(q_i, i) \leq \mathsf{rank}_{s_0}(s_i, i)$$

When this happens, we say that Duplicator *plays safely*. Duplicator can alway play safely by selecting maximal successors; in this case, we say that Duplicator *plays maximally*. Sometimes, Duplicator has to play safely, by sub-maximally: The liveness condition forces Duplicator to eventually visit an accepting state whenever Spoiler does so, and this might require to take a sub-maximal successor.

Duplicator plays in three modes: *start-up mode*, *normal mode* and *obligation mode*. In start-up and normal mode Duplicator plays maximally, while in obligation mode Duplicator needs to satisfy the liveness condition, while still playing safely. Start-up mode happens only at the beginning, to prepare for normal mode. Normal and obligation mode alternate during the game, as new obligations are raised by Spoiler and met by Duplicator.

In start-up mode, Duplicator plays maximally until she has a limit ordinal. Then the game goes to normal mode. There is no pending obligation at the beginning of normal mode, since at the end of startup-mode Duplicator has rank $\lambda + 1$, and, by Lemma 3.6.5, she is accepting.

In normal mode, Duplicator is guaranteed to always have a limit ordinal rank, which is preserved by playing maximally. The game stays in normal mode as long as Spoiler is not accepting. Whenever $q_i \in F$ at round i, the game switches to obligation mode.

At the beginning of obligation mode, Duplicator always has a limit ordinal rank λ. Since $q_i \in F$, by Lemma 3.6.5, Spoiler's rank at the beginning of obligation mode is a successor ordinal $\alpha + 1 < \lambda$. W.l.o.g. we assume that Spoiler plays maximally during obligation mode. By Lemma 3.6.7, there exists a maximal path $\langle q_i, i\rangle\langle q_{i+1}, i+1\rangle \cdots \langle q_j, j\rangle$ s.t. Spoiler's rank at round $j \geq i$ is $\lambda' + 1$. A further maximal move by Spoiler extends the previous path to $\langle q_{j+1}, j+1\rangle$. By Lemma 3.6.6(b), Spoiler's rank at round $j+1$ is now λ', and, by Lemma 3.6.6(c), $\lambda' \leq \alpha + 1$. From now on, we assume w.l.o.g. that Spoiler plays maximally as to preserve rank λ'. We show that Duplicator can match the pending obligation while still playing safely.

By induction from Lemma 3.6.6(b), Duplicator can play a maximal path

$$\langle s_i, i\rangle\langle s_{i+1}, i+1\rangle \cdots \langle s_{j+1}, j+1\rangle$$

s.t. Duplicator's rank at round $j+1$ is still λ. But $\lambda' \leq \alpha + 1 < \lambda$, thus $\lambda' < \lambda$. Consequently, $\lambda' \leq \lfloor\lambda\rfloor$ by the definition of floor operation. So, let $\langle q_{j+1}, s_{j+1}\rangle$ be the configuration at round $j+1$. By Lemma 3.6.8, Duplicator can play a path

$$\langle s_{j+1}, j+1\rangle\langle s_{j+2}, j+2\rangle \cdots \langle s_k, k\rangle$$

with $k > j+1$ and s.t. Duplicator's rank at round k is $\alpha' + 1$ with $\lfloor\lambda\rfloor \leq \alpha'$. Since $\lambda' \leq \lfloor\lambda\rfloor$, we have $\lambda' \leq \alpha'$. By Lemma 3.6.7, Duplicator can extend the previous path with a maximal path

$$\langle s_k, k\rangle\langle s_{k+1}, k+1\rangle \cdots \langle s_h, h\rangle$$

s.t. Duplicator's rank at round $h > k$ is $\lambda'' + 1$ with $\lfloor \alpha' + 1 \rfloor \leq \lambda''$. By Lemma 3.6.5, $s_h \in F$, thus Duplicator has satisfied the pending obligation. In the next round $h + 1$, the game switches back to normal mode. By Lemma 3.6.6(b), Duplicator's rank is the limit ordinal λ'', thus satisfying the precondition of normal mode. Notice that $\lambda' \leq \alpha' < \alpha' + 1$ implies $\lambda' \leq \lfloor \alpha + 1 \rfloor$. Therefore, $\lambda' \leq \lambda''$ and the safety condition is satisfied. □

The second direction is more interesting. Its contrapositive says that if Spoiler has a better rank than Duplicator, then Duplicator loses the w-simulation game, *even if she uses multiple pebbles*. Thus, ranks are necessary for simulation.

The following lemma, and its proof, conclude the chapter.

Lemma 3.6.2. Let $w \in \Sigma^\omega$ and $k > 0$. If $q_0 \sqsubseteq_w^{k\text{-de}} s_0$, then $\mathsf{rank}(q_0, 0) \leq \mathsf{rank}(s_0, 0)$

Proof. We prove the contrapositive. Assume $\mathsf{rank}_{q_0}(q_0, 0) \not\leq \mathsf{rank}_{s_0}(s_0, 0)$. Since ordinals are linearly ordered, this means $\mathsf{rank}_{q_0}(q_0, 0) > \mathsf{rank}_{s_0}(s_0, 0)$. We have to show $q_0 \not\sqsubseteq_w^{k\text{-de}} s_0$, for arbitrary $k > 0$. Take n to be the size of the automaton. We actually prove that Duplicator does not win even with n pebbles, i.e., $q_0 \not\sqsubseteq_w^{n\text{-de}} s_0$.

For any round i, let $\langle q_i, \mathbf{s}_i \rangle$ be the current configuration of the simulation game $G_w^{n\text{-de}}(q_0, s_0)$. (For simplicity, we omit the third component.) $\mathbf{s}_i$ identifies a *subset* of vertices at level i in $G_{s_0}^w$: $\mathbf{s}_i \subseteq \{s \mid \langle s, i \rangle \in G_{s_0}^w\}$. We extend the notion of rank from vertices to *sets of vertices*, by taking the maximal rank amongst all members: That is, the rank of Duplicator at round i is $\sup_{s \in \mathbf{s}_i} \mathsf{rank}_{s_0}(s, i)$. As before, Spoiler's rank is just $\mathsf{rank}_{q_0}(q_i, i)$.

If Spoiler has rank ω^2, then she will just play a fair path, which exists by Lem. 3.6.3. Duplicator loses, since she has rank $< \omega^2$ by assumption, and, by Lemma 3.6.3, there exists no fair path in this case. Otherwise, assume Spoiler's rank is $< \omega^2$, and we proceed by applying the properties from Section 3.6.4.

We assume that, at round 0, every pebble has limit rank. If not, Spoiler can enforce such a situation by waiting a suitable number of rounds. (I.e., by playing maximally according to Lemma 3.6.6(b).) So, let Spoiler have limit rank λ, and let Duplicator have limit rank μ, with $\lambda > \mu$. Thus, $\lfloor \lambda \rfloor \geq \mu$. We assume that Duplicator always plays maximally, unless she is forced to act differently. By Lemma 3.6.8, Spoiler can play a path $\langle q_0, 0 \rangle \langle q_1, 1 \rangle \cdots \langle q_i, i \rangle$ with $i > 0$, s.t. her rank at round i is $\alpha + 1$ and $\alpha \geq \lfloor \lambda \rfloor$. Since $\lfloor \lambda \rfloor \geq \mu$, we have $\alpha \geq \mu$. By Lemma 3.6.7, Spoiler can extend the previous path with a maximal path $\langle q_i, i \rangle \langle q_{i+1}, i + 1 \rangle \cdots \langle q_j, j \rangle$ with $j > i$, s.t. her rank

at round j is $\lambda' + 1$ and $\lambda' \geq \lfloor \alpha + 1 \rfloor$. By Lemma 3.6.5, $q_j \in F$. From $\alpha + 1 > \alpha \geq \mu$ we have $\lfloor \alpha + 1 \rfloor \geq \mu$. By $\lambda' \geq \lfloor \alpha + 1 \rfloor$, we have $\lambda' \geq \mu$. By performing a further maximal step, Spoiler reaches state $\langle q_{j+1}, j+1 \rangle$ with rank λ'. From now on, Spoiler plays maximally.

Since Duplicator was supposed to play maximally, in the meantime she replied to Spoiler with a sequence $\langle \mathbf{s}_0, 0 \rangle \langle \mathbf{s}_1, 1 \rangle \cdots \langle \mathbf{s}_{j+1}, j+1 \rangle$ maintaining rank μ. Now, let $\langle q_{j+1}, \mathbf{s}_{j+1} \rangle$ be the current configuration, and remember that Duplicator has a pending obligation to satisfy. That is, Duplicator has to ensure that at some future round all pebbles are good since round $j+1$. Let $k > j+1$ be such a round, at which point Duplicator is in position $\langle \mathbf{s}_k, k \rangle$ and $\mathbf{s}_{j+1} \Longrightarrow_F^* \mathbf{s}_k$. Every state in $\mathbf{s}_k$ has seen an accepting state since round $j+1$. By Lemma 3.6.5, accepting states receive successor ranks $\alpha + 1$. Since ranks are non-increasing along paths in $G^w_{s_0}$ (by Lemma 3.6.6(c)), at each visit of a final state, $\alpha + 1 < \mu$. Therefore, all states in $\mathbf{s}_k$ have rank $< \mu$. In particular, Duplicator's rank at round k is $< \mu$. Since Duplicator has now satisfied the pending obligation, she will again play maximally, from round k on. By Lemma 3.6.6, all pebbles eventually stabilize to a limit rank. Since there are only finitely many pebbles, this stabilization happens in a finite number of rounds. At some round $h \geq k$, Duplicator's rank is $\mu' < \mu$. Let $\mathbf{s}_h$ be the position of Duplicator's pebbles at round h.

Meanwhile, Spoiler replied with a maximal path $\langle q_{j+1}, j+1 \rangle \cdots \langle q_h, h \rangle$, preserving rank $\lambda' \geq \mu > \mu'$ until round h. Therefore, $\lambda' > \mu'$, and the situation at round h is the same as at round 0, with the crucial difference that Duplicator's rank at round h is strictly smaller than it was at round 0, $\mu' < \mu$.

Since ordinals are well-founded, Spoiler can iterate the whole procedure and after a finitely many repetitions Duplicator hits the trap rank ω, from which no more accepting states can be visited. At that point, Spoiler would have a limit rank $\lambda'' > \omega$, so she will just force one more obligation, which would remain unmet. Thus, Spoiler wins. $\square$

Chapter 4

Jumping simulations

Contents

4.1 Overview **72**

4.2 Preliminaries **74**

4.2.1 Jumping containments 75

4.2.2 Jumping-safe preorders 76

4.3 Jumping simulation transformers **76**

4.3.1 Definitions 77

4.3.2 Basic properties 80

4.3.3 Composing strategies and transitivity 82

4.4 Language containment and inclusion **84**

4.4.1 Jumping simulation implies jumping containment 85

4.4.2 Jumping containment implies jumping inclusion 86

4.4.3 Deciding ordinary language inclusion 87

4.4.4 Coherent sequences of jumping paths 88

4.5 Reflexivity and jumping-safety **91**

4.5.1 Reflexivity 92

4.5.2 Preserving jumping-safe preorders 93

4.5.3 Proof of reflexivity 94

4.6 Proxy simulation hierarchies **98**

4.6.1 Idempotence 99

4.6.2 Proxy simulations 100

4.6.3 Iterated quotienting 106
4.7 Proxy simulations vs Mediated preorder **109**
4.7.1 Mediated preorder 109
4.7.2 Comparison 110
4.7.3 Mediated preorder as a jumping simulation game 115
4.8 Computing jumping simulations **116**
4.8.1 Jumping predecessor operators 116
4.8.2 Direct and backward simulations 117
4.8.3 Fair simulation 118
4.8.4 Delayed simulation 118

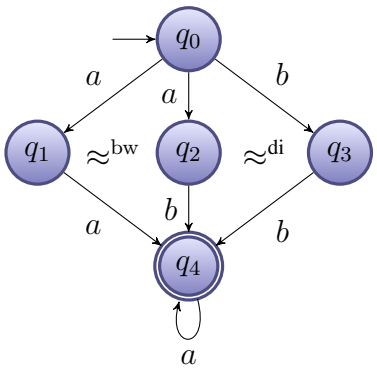
4.1 Overview

In this chapter, we study simulation-like preorders for quotienting NBA coarser than ordinary simulation. The main difference with the previous chapter is that here we concentrate on preorders which are *efficiently computable*. Indeed, a major drawback for practical applications of fixed-word simulations from Chapter 3 is that they are expensive to compute (i.e., PSPACE-complete). In this chapter, we define *jumping simulations*, which are computable in PTIME. Under certain assumptions, we show that jumping simulations can be used for quotienting NBA (i.e., they are GFQ). While in general jumping simulations are incomparable w.r.t. fixed-word simulations, they are still coarser than ordinary simulation. Therefore, quotienting w.r.t. jumping simulations yields automata that are smaller than what was previously possible w.r.t. practical (i.e., PTIME) methods. Being able to obtain smaller automata is beneficial in any application involving automata, since smaller automata are more efficiently manipulated and analyzed, e.g., in model-checking (cf. Chapter 1). Another prominent application of simulations is as an efficient test for language inclusion between automata, and we show that jumping simulations can be used also in this context. Therefore, the theory of jumping simulations offers a comprehensive generalization of ordinary simulation, and, at the same time, it retains the same good properties w.r.t. quotienting, language inclusion checking, and complexity.

We aim at defining simulation-like preorders coarser than either forward or backward simulation. We begin by noticing that preorders coarser than *both* forward and backward

simulation are in general not GFQ.

Example 4.1.1 - Simultaneous quotienting w.r.t. both forward and backward simulations is incorrect



States q_1 and q_2 are backward simulation equivalent, while q_2 and q_3 are direct simulation equivalent:

$$q_1 \approx^{\text{bw}} q_2 \approx^{\text{di}} q_3$$

If we simultaneously "glue together" q_1, q_2 and q_3—i.e., by quotienting w.r.t. the transitive closure of $\approx^{\text{bw}} \cup \approx^{\text{di}}$—, then we would also identify q_1 and q_3, which are otherwise unrelated. This results in a quotient automaton incorrectly accepting the extraneous word ba^ω, originally not in the language of the automaton.

Therefore, we restrict our attention to preorders coarser than either forward or backward simulation, but not coarser than both.

We introduce *jumping simulations*, which generalize ordinary simulation by allowing players to "jump" to other states before taking a transition. Jumps are taken w.r.t. a given input preorder $\sqsubseteq$. Consequently, jumping simulations act as *transformers*, mapping preorders $\sqsubseteq$ to a corresponding $\sqsubseteq$-jumping simulation preorder. We study the properties of jumping simulations transformers, and, in particular, we investigate under which conditions jumping simulations map GFQ preorders to GFQ preorders. The material presented in this chapter is a significant extension and improvement of material that has been published in [28].

The chapter is structured as follows. In Section 4.2, we fix notation. In Section 4.3, jumping simulations are introduced and their basic properties are investigated. In analogy to ordinary simulations, we define direct, delayed, fair, and backward jump-

ing simulation. We establish their transitivity (as relations) and non-decreasingness (as transformers). Subsequently, in Section 4.4, we explain the relationship between jumping simulations and various forms of containment. Moreover, we show that jumping simulation can be used as an efficient under-approximation to language inclusion between automata (and this approximation is provably better than ordinary simulation).

In Section 4.5, we present the main result of the chapter, which can be approximately stated as follows: *Jumping simulations map GFQ preorders to coarser GFQ preorders*; see Theorems 4.5.3 and 4.5.4. This allows us to nest transformers in order to produce hierarchies of coarser and coarser GFQ preorders. In Section 4.6 we explore this possibility, and we consider *proxy simulation hierarchies*, which are obtained by iterated application of forward and backward transformers, starting from the identity relation (which is trivially GFQ).

In Section 4.7, we compare jumping simulations with *mediated preorder*. Mediated preorder is a related GFQ relation that has been studied in the literature [4], and we show that it is in general incomparable w.r.t. jumping simulations.

Finally, in Section 4.8, we give an algorithm for computing jumping simulations; it arises as a classical fixpoint characterization of the winning regions in the corresponding simulation game. As a consequence, jumping simulation are computable in PTIME.

4.2 Preliminaries

We adopt the following convention: States that "can do more" simulation-wise are always written as the larger ones w.r.t. the preorder; i.e., in both $p \preceq q$ and $q \succeq p$, q simulates p. If $\preceq$ is a preorder, then we write $\succeq$ for its transpose.

Definition 4.2.1 (Jumping transitions)**.** *For a preorder $\sqsubseteq$, we write*

- $p \sqsubseteq^F q$ *iff* $p \sqsubseteq q$ *and, moreover, there exists* $r \in F$ *s.t.* $p \sqsubseteq r \sqsubseteq q$*;*
- $q \sqsubseteq I$ *iff there exists* $\hat{q} \in I$ *s.t.* $q \sqsubseteq \hat{q}$*;*
- $q \sqsubseteq \xrightarrow{a} q'$ *iff there exists* $\hat{q}$ *s.t.* $q \sqsubseteq \hat{q} \xrightarrow{a} q'$*;*
- $q \sqsubseteq \xrightarrow{a}_F q'$ *iff there exists* $\hat{q}$ *s.t.* $q \sqsubseteq^F \hat{q} \xrightarrow{a} q'$*.*

Similar definitions hold for $\sqsubseteq \xleftarrow{a}$ *and* $\sqsubseteq \xleftarrow{a}_F$*.*

4.2.1 Jumping containments

Let $\sqsubseteq$ be a preorder, and fix an infinite word $w = a_0 a_1 \cdots \in \Sigma^\omega$. A $\sqsubseteq$*-jumping* w*-path* is an infinite sequence of the form

$$\pi = q_0 \sqsubseteq \xrightarrow{a_0} q_1 \sqsubseteq \xrightarrow{a_1} q_2 \sqsubseteq \xrightarrow{a_2} \cdots$$

We say that π *starts at* q_0. π is *initial* if $q_0 \in I$, and is *fair* if $q_i \sqsubseteq \xrightarrow{a_i}_F q_{i+1}$ for infinitely many i's. Finite $\sqsubseteq$-jumping paths over finite words w are defined similarly. If w has length n, then π *ends at* q_n.

Definition 4.2.2 (Jumping language)**.** *The* $\sqsubseteq$-language $\mathcal{L}^{\sqsubseteq}(q)$ *of a state* q *is the set of* ω*-words which induce fair* $\sqsubseteq$*-jumping paths from* q*:*

$$w = a_0 a_1 \cdots \in \mathcal{L}^{\sqsubseteq}(q) \iff \exists \textit{ fair } \sqsubseteq\textit{-jumping } w\textit{-path } \pi \textit{ starting at } q$$

As usual, for a set of states $P \subseteq Q$*, let* $\mathcal{L}^{\sqsubseteq}(P) = \bigcup_{q \in P} \mathcal{L}^{\sqsubseteq}(q)$*, and, for an automaton* $\mathcal{Q}$*, let* $\mathcal{L}^{\sqsubseteq}(\mathcal{Q}) = \mathcal{L}^{\sqsubseteq}(I)$*, where* I *is the set of initial states of* $\mathcal{Q}$*.*

Jumping forward containment is the jumping analogue of fair containment from Section 3.3. For simplicity, we do not consider other forward containments in this chapter.

Definition 4.2.3 (Jumping forward containment)**.** *For two preorders* $\sqsubseteq_0$ *and* $\sqsubseteq_1$*, and states* q *and* s*, we say that* q is $(\sqsubseteq_0, \sqsubseteq_1)$-contained in s*, written* $q \subseteq^{\mathrm{fw}}(\sqsubseteq_0, \sqsubseteq_1)\ s$*, iff the* $\sqsubseteq_0$*-language of* q *is contained in the* $\sqsubseteq_1$*-language of* s*:*

$$q \subseteq^{\mathrm{fw}}(\sqsubseteq_0, \sqsubseteq_1)\ s \quad \textit{iff} \quad \mathcal{L}^{\sqsubseteq_0}(q) \subseteq \mathcal{L}^{\sqsubseteq_1}(s)$$

Jumping backward containment is the jumping analogue of backward containment from Section 3.3. Intuitively, two states q and s are in *jumping backward containment* $\subseteq^{\mathrm{bw}}(\sqsubseteq_0, \sqsubseteq_1)$ iff initial $\sqsubseteq_0$-jumping paths ending in q are matched by corresponding initial $\sqsubseteq_1$-paths ending in s, and, if the first path takes an accepting transition, then so does the second.

Definition 4.2.4 (Jumping backward containment)**.** *For two states* q *and* s*, let* $q \subseteq^{\mathrm{bw}} (\sqsubseteq_0, \sqsubseteq_1)\ s$ *iff, for any initial* $\sqsupseteq_0$*-jumping path* $q_0 \sqsupseteq_0 \hat{q}_0 \xrightarrow{a_0} \cdots \xrightarrow{a_{n-1}} q_n \sqsupseteq_0 \hat{q}_n$ *ending at* $\hat{q}_n = q$*, there exists an initial* $\sqsupseteq_1$*-jumping path* $s_0 \sqsupseteq_1 \hat{s}_0 \xrightarrow{a_0} \cdots \xrightarrow{a_{n-1}} s_n \sqsupseteq_1 \hat{s}_n$ *ending at* $\hat{s}_n = s$ *s.t., for any* $0 \leq i < n$*,*

$$q_i \sqsupseteq_0^F \hat{q}_i \textit{ implies } s_i \sqsupseteq_1^F \hat{s}_i$$

4.2.2 Jumping-safe preorders

Jumping-safe preorders are central in this chapter. Intuitively, a preorder is jumping-safe if fair jumping paths do not introduce new words into the language of the automaton.

Definition 4.2.5 (Jumping-safe preorder)**.** *For an automaton $\mathcal{Q}$, a preorder $\sqsubseteq \subseteq Q \times Q$ is* jumping-safe *(w.r.t. Q) iff*

$$\mathcal{L}^{\sqsubseteq}(\mathcal{Q}) = \mathcal{L}(\mathcal{Q})$$

Therefore, if a preorder is jumping-safe, then fair and initial jumping paths can always be replaced by fair and initial non-jumping paths.

Remark 4.2.6. Jumping-safe preorders and GFQ equivalences are strongly related. If a preorder is jumping-safe, then the induced equivalence is GFQ, and, vice versa, if an equivalence is GFQ, then it is also jumping-safe.

In the next section we introduce several PTIME preorder transformers. The main result of this chapter says that those transformers map jumping-safe preorders into jumping-safe preorders. (cf. Theorems 4.5.3 and 4.5.4).

4.3 Jumping simulation transformers

In this section we introduce several simulation transformers τ, which map a given preorder $\sqsubseteq$ into a new preorder $\tau(\sqsubseteq)$. The transformer τ is defined game-theoretically via a variant of the usual simulation game.

Fix a preorder $\sqsubseteq$. We introduce a novel class of simulation games where both Spoiler and Duplicator can *jump* during the game. The new position a player can jump to depends on the preorder $\sqsubseteq$: If a player's pebble is on q, then the pebble can instantaneously jump to any state $\hat{q}$ s.t. $q \sqsubseteq \hat{q}$. We call state $\hat{q}$ a *proxy*, which acts as a dynamic mediator for taking jumping transitions. From the proxy, the pebble can then take an ordinary a-transition to some state q'. Overall, we have a jumping transition $q \sqsubseteq \hat{q} \xrightarrow{a} q'$, which we usually abbreviate as $q \sqsubseteq \xrightarrow{a} q'$ by keeping the proxy implicit.

The winning condition depends on the specific transformer that we consider. We study four transformers τ^x, for $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}, \mathrm{bw}\}$, in analogy with direct, delayed, fair and backward simulation.

Intuitively, the acceptance condition is shifted from states to transitions. A pebble is no longer statically accepting in a given position, but can instead dynamically take an

accepting $\sqsubseteq$-transition. A $\sqsubseteq$-transition is accepting if the pebble can "transit through" an accepting state before reaching the proxy. We write an accepting a-transition from q to q' as $q \sqsubseteq \xrightarrow{a}_F q'$: Expanding the definition, this means that there exist states $q^F \in F$ and $\hat{q}$ s.t. $q \sqsubseteq q^F \sqsubseteq \hat{q} \xrightarrow{a} q'$.

In the general definition, we allow Spoiler and Duplicator to jump w.r.t. distinct preorders. For obtaining transitivity, we then restrict ourselves to the case where they jump w.r.t. the same preorder (cf. Section 4.3.3).

4.3.1 Definitions

Fix two preorders $\sqsubseteq_0$ and $\sqsubseteq_1$, controlling the jumping capability of Spoiler and Duplicator, respectively. Formally, the configurations of the basic $(\sqsubseteq_0, \sqsubseteq_1)$-simulation game between state q and state s, $G_{q,s}(\sqsubseteq_0, \sqsubseteq_1)$, are the same as in ordinary simulation. That is, they are pairs of states $\langle q_i, s_i \rangle$. The difference lies in the fact that more transitions are available in the $(\sqsubseteq_0, \sqsubseteq_1)$-game than in the ordinary game. We first discuss forward transformers.

Forward transformers Let $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$. Initially, the game is in configuration $\langle q_0, s_0 \rangle$. Subsequently, if in round i the current configuration is $\langle q_i, s_i \rangle$, then the configuration for the next round $i+1$ is determined as follows:

- First, Spoiler chooses an input symbol a_i and a $\sqsubseteq_0$-jumping transition

$$q_i \sqsubseteq_0 \xrightarrow{a_i} q_{i+1}$$

- Then, Duplicator replies with a matching $\sqsubseteq_1$-jumping transition

$$s_i \sqsubseteq_1 \xrightarrow{a_i} s_{i+1}$$

The next configuration is $\langle q_{i+1}, s_{i+1} \rangle$. If in any round a player cannot move because no transition is available, then she loses. Otherwise, the game goes on "for ever" (for ω rounds), and the two player jointly build two infinite jumping paths,

$$\pi_0 = q_0 \sqsubseteq_0 \xrightarrow{a_0} q_1 \sqsubseteq_0 \xrightarrow{a_1} \cdots \qquad \text{and} \qquad \pi_1 = s_0 \sqsubseteq_1 \xrightarrow{a_0} s_1 \sqsubseteq_1 \xrightarrow{a_1} \cdots$$

The winner is established by a predicate on π_0 and π_1, which depends on the kind x of simulation we are considering. Intuitively, Duplicator has to match accepting transitions in the same way accepting states are matched in the corresponding ordinary game. This translates to the the following acceptance conditions:

1. *Direct* $(\sqsubset_0, \sqsubseteq_1)$*-simulation*, $x = \mathrm{di}$. Duplicator wins if, whenever Spoiler takes an accepting $\sqsubseteq_0$-transition, then she takes an accepting $\sqsubseteq_1$-transition:

$$\forall (i \geq 0) \ \cdot \ q_i \sqsubseteq_0 \xrightarrow{a_i}_F q_{i+1} \implies s_i \sqsubseteq_1 \xrightarrow{a_i}_F s_{i+1}$$

2. *Delayed* $(\sqsubset_0, \sqsubseteq_1)$*-simulation*, $x = \mathrm{de}$. Duplicator wins if, whenever Spoiler takes an accepting $\sqsubseteq_0$-transition in round i, then she takes an accepting $\sqsubseteq_1$-transition in some later round $j \geq i$:

$$\forall (i \geq 0) \ \cdot \ q_i \sqsubseteq_0 \xrightarrow{a_i}_F q_{i+1} \implies \exists (j \geq i) \ \cdot \ s_j \sqsubseteq_1 \xrightarrow{a_j}_F s_{j+1}$$

3. *Fair* $(\sqsubset_0, \sqsubseteq_1)$*-simulation*, $x = \mathrm{f}$. Duplicator wins if, whenever Spoiler takes infinitely many accepting $\sqsubseteq_0$-transitions, then she takes infinitely many accepting $\sqsubseteq_1$-transitions:

$$\pi \text{ fair} \implies \pi' \text{ fair}$$

The rules for forward jumping simulations are summarized below.

Simulation game for $\tau^x(\sqsubseteq_0, \sqsubseteq_1)$, with $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$

Current configuration		Next configuration
$\langle q_i, s_i \rangle$	- - - - - - →	$\langle q_{i+1}, s_{i+1} \rangle$

Spoiler chooses $q_i \sqsubseteq_0 \xrightarrow{a_i} q_{i+1}$

Duplicator replies with $s_i \sqsubseteq_1 \xrightarrow{a_i} s_{i+1}$

Winning condition

$x = \mathrm{di}$: $\forall (i \geq 0) \ \cdot \ q_i \sqsubseteq_0 \xrightarrow{a_i}_F q_{i+1} \implies s_i \sqsubseteq_1 \xrightarrow{a_i}_F s_{i+1}$

$x = \mathrm{de}$: $\forall (i \geq 0) \ \cdot \ q_i \sqsubseteq_0 \xrightarrow{a_i}_F q_{i+1} \implies \exists (j \geq i) \ \cdot \ s_j \sqsubseteq_1 \xrightarrow{a_j}_F s_{j+1}$

$x = \mathrm{f}$: π fair $\implies \pi'$ fair

Backward transformer For $x = \mathrm{bw}$ the rules of the game are very similar, except for the fact that transitions are taken backwards. Suppose in round i the current configuration is $\langle q_i, s_i \rangle$. The next configuration $\langle q_{i+1}, s_{i+1} \rangle$ is determined as follows:

- First, Spoiler chooses an input symbol a_i and a backward $\sqsubseteq_0$-jumping transition

$$q_i \sqsubseteq_0 \xleftarrow{a_i} q_{i+1}$$

- Then, Duplicator replies with a matching backward $\sqsubseteq_1$-jumping transition

$$s_i \sqsubseteq_1 \xleftarrow{a_i} s_{i+1}$$

If in any round a player cannot move because no transition is available, then she loses. Otherwise, the game never stops and the players produce two infinite jumping backward paths:

$$\pi_0 = q_0 \sqsubseteq_0 \xleftarrow{a_0} q_1 \sqsubseteq_0 \xleftarrow{a_1} \cdots \qquad \text{and} \qquad \pi_1 = s_0 \sqsubseteq_1 \xleftarrow{a_0} s_1 \sqsubseteq_1 \xleftarrow{a_1} \cdots$$

The winning condition in this case is as follows.

4. *Backward* $(\sqsubseteq_0, \sqsubseteq_1)$*-simulation*, $x = \mathrm{bw}$. Duplicator wins if the two conditions below are satisfied, the first regarding accepting transitions and the second initial states.

   a) Whenever Spoiler takes an accepting backward $\sqsubseteq_0$-transition, Duplicator takes an accepting backward $\sqsubseteq_1$-transition (in the same round):

$$\forall (i \geq 0) \ \cdot \ q_i \sqsubseteq_0 \xleftarrow{a_i}_F q_{i+1} \implies s_i \sqsubseteq_1 \xleftarrow{a_i}_F s_{i+1}$$

   b) Initial states are treated differently, since they do not need to be part of transitions: Whenever Spoiler could possibly $\sqsubseteq_0$-jump to an initial state, Duplicator can $\sqsubseteq_1$-jump to an initial state:

$$\forall (i \geq 0) \ \cdot \ q_i \sqsubseteq_0 I \implies s_i \sqsubseteq_1 I$$

The rules for backward jumping simulation are summarized below.

Simulation game for $\tau^{\mathrm{bw}}(\sqsubseteq_0, \sqsubseteq_1)$

Current configuration $\langle q_i, s_i \rangle$ - - - - - - → Next configuration $\langle q_{i+1}, s_{i+1} \rangle$

Spoiler chooses $q_i \sqsubseteq_0 \xleftarrow{a_i} q_{i+1}$

Duplicator replies with $s_i \sqsubseteq_1 \xleftarrow{a_i} s_{i+1}$

Winning condition

1) $\forall (i \geq 0) \cdot q_i \sqsubseteq_0 \xleftarrow{a_i}_F q_{i+1} \implies s_i \sqsubseteq_1 \xleftarrow{a_i}_F s_{i+1}$
2) $\forall (i \geq 0) \cdot q_i \sqsubseteq_0 I \implies s_i \sqsubseteq_1 I$

As usual, $q\ \tau^x(\sqsubseteq_0, \sqsubseteq_1)\ s$ holds iff Duplicator wins the jumping x-simulation game $G^x_{q,s}(\sqsubseteq_0, \sqsubseteq_1)$. Moreover, when Spoiler and Duplicator have the same jumping capabilities w.r.t. a given preorder $\sqsubseteq$, we simply write $\tau^x(\sqsubseteq)$ instead of $\tau^x(\sqsubseteq, \sqsubseteq)$; this is very important, since in this case we show that $\tau^x(\sqsubseteq)$ is also a preorder (cf. Corollary 4.3.7 in Section 4.3.3).

4.3.2 Basic properties

We investigate some basic properties of τ^x, which are immediate consequences of jumping games.

Ordinary simulation When no jumps are allowed, we clearly recover ordinary simulation: That is, for $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$, we have $\tau^x(Id) = \sqsubseteq^x$.

Inclusions Forward simulations for $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$ are linearly ordered by inclusion, direct simulation being the finest and fair simulation the coarsest. This follows immediately from the fact that "easier" winning conditions favor Duplicator.

Lemma 4.3.1. *Direct, delayed, and fair jumping simulations are ordered by inclusion. That is, for $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$ and preorders $\sqsubseteq_0, \sqsubseteq_1$, let $\preceq^x = \tau^x(\sqsubseteq_0, \sqsubseteq_1)$. Then,*

$$\preceq^{\mathrm{di}} \subseteq \preceq^{\mathrm{de}} \subseteq \preceq^{\mathrm{f}}$$

Monotonicity and non-decreasingness When we give more jumping power to Duplicator, τ^x clearly grows. Symmetrically, when Spoiler can perform "longer" jumps, τ^x shrinks. Therefore, the two-arguments transformer $\tau^x(\cdot, \cdot)$ is *antitone* in the first argument and *isotone* in the second argument. This means that, for preorders $\sqsubseteq_0, \sqsubseteq_1, \sqsubseteq'_0, \sqsubseteq'_1$,

$$\sqsubseteq_0 \subseteq \sqsubseteq'_0 \text{ and } \sqsubseteq_1 \subseteq \sqsubseteq'_1 \quad \text{implies} \quad \tau^x(\sqsubseteq'_1, \sqsubseteq_0) \subseteq \tau^x(\sqsubseteq_1, \sqsubseteq'_0)$$

In particular, when we only allow Duplicator to jump, we get a reflexive relation (not necessarily transitive) coarser than ordinary simulation.

Lemma 4.3.2. *Jumping simulation is coarser than simulation when only Duplicator is allowed to jump. Formally, for $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}, \mathrm{bw}\}$, let $\sqsubseteq^x = \tau^x(Id)$ be the corresponding ordinary simulation preorder. Then, for any preorder $\sqsubseteq$ and states q and s,*

$$q \sqsubseteq^x s \quad \textit{implies} \quad q\ \tau^x(Id, \sqsubseteq)\ s$$

Otherwise, when we give the same jumping power to both players at the same time, the resulting one-argument transformer $\tau^x(\cdot)$ is no longer monotone. The lack of such a basic property might seem unfortunate. However, we can prove that $\tau^x(\cdot)$ enjoys another related useful property, that of being *non-decreasing* (up to transposition). That is, $\tau^x(\sqsubseteq)$ is at least as coarse as the transpose of $\sqsubseteq$.

Lemma 4.3.3 (Non-decreasingness)**.** *Let $\sqsubseteq$ be a preorder. Then, (the transpose of) $\sqsubseteq$-jumping simulation is coarser than $\sqsubseteq$. Formally, for $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}, \mathrm{bw}\}$ and states q and s,*

$$q \sqsubseteq s \quad \textit{implies} \quad s\ \tau^x(\sqsubseteq)\ q$$

Proof. Let $\preceq = \tau^x(\sqsubseteq)$ and assume $q \sqsubseteq s$. Intuitively, the statement holds since, if Spoiler can jump to any proxy $\sqsubseteq$-larger than s, then, by transitivity of $\sqsubseteq$, also Duplicator can jump from q to the same proxy.

For $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$, let's Spoiler select transition $s \sqsubseteq \hat{s} \xrightarrow{a} s'$, as required by the definition of $\preceq$. By transitivity, $q \sqsubseteq \hat{s}$. Thus, Duplicator replies with $q \sqsubseteq \hat{s} \xrightarrow{a} s'$.

The winning condition is immediately satisfied since $s \sqsubseteq^F \hat{s}$ implies $q \sqsubseteq^F \hat{s}$. From configuration $\langle s', s' \rangle$, Duplicator obviously wins.

The proof for $x = \mathrm{bw}$ is analogous. Let's Spoiler select transition $s \sqsubseteq \hat{s} \xleftarrow{a} s'$. By transitivity, $q \sqsubseteq \hat{s}$. Thus, Duplicator replies with $q \sqsubseteq \hat{s} \xleftarrow{a} s'$. The winning condition is satisfied since 1) $s \sqsubseteq^F \hat{s}$ implies $q \sqsubseteq^F \hat{s}$, and 2) $s \sqsubseteq I$ implies $q \sqsubseteq I$. From $\langle s', s' \rangle$, Duplicator then obviously wins. □

Preservation When Duplicator plays according to a winning strategy, the relation $\tau^x(\sqsubseteq_0, \sqsubseteq_1)$ is preserved during the simulation game: That is, if in any round the current configuration is $\langle q, s \rangle$ and $q\ \tau^x(\sqsubseteq_0, \sqsubseteq_1)\ s$, then she can ensure that, if the next configuration is $\langle q', s' \rangle$, then $q'\ \tau^x(\sqsubseteq_0, \sqsubseteq_1)\ s'$. This follows immediately from the definition of winning strategy.

Lemma 4.3.4. *Let $q\ \tau^x(\sqsubseteq_0, \sqsubseteq_1)\ s$.*

a) Let $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$. For every $\sqsubseteq_0$-jumping forward transition $q \sqsubseteq_0 \xrightarrow{a} q'$, there exists a $\sqsubseteq_1$-jumping forward transition $s \sqsubseteq_1 \xrightarrow{a} s'$ s.t. $q'\ \tau^x(\sqsubseteq_0, \sqsubseteq_1)\ s'$.

For $x = \mathrm{di}$, if $q \sqsubseteq_0 \xrightarrow{a}_F q'$, then s' can be chosen s.t. $s \sqsubseteq_1 \xrightarrow{a}_F s'$.

b) Let $x = \mathrm{bw}$. For every $\sqsubseteq_0$-jumping backward transition $q \sqsubseteq_0 \xleftarrow{a} q'$, there exists a $\sqsubseteq_1$-jumping backward transition $s \sqsubseteq_1 \xleftarrow{a} s'$ s.t. $q'\ \tau^x(\sqsubseteq_0, \sqsubseteq_1)\ s'$.

Moreover, if $q \sqsubseteq_0 \xleftarrow{a}_F q'$, then s' can be chosen s.t. $s \sqsubseteq_1 \xleftarrow{a}_F s'$.

4.3.3 Composing strategies and transitivity

In this section, fix preorders $\sqsubseteq_0$, $\sqsubseteq_1$, $\sqsubseteq_1'$ and $\sqsubseteq_2$. We show how Duplicator's strategies can be composed horizontally with a partial composition operator $\bowtie$: The idea is that two strategies for a $(\sqsubseteq_0, \sqsubseteq_1)$- and a $(\sqsubseteq_1', \sqsubseteq_2)$-game starting from positions $\langle q, r \rangle$ and $\langle r, s \rangle$, respectively, can be composed into a strategy for a $(\sqsubseteq_0, \sqsubseteq_2)$-game starting from position $\langle q, s \rangle$ (under the assumption $\sqsubseteq_1 \subseteq \sqsubseteq_1'$, see next). We show that composition preserves winning strategies (Lemma 4.3.5), and, in turn, this is used to establish that the one-argument transformer is transitive (Corollary 4.3.7).

Let $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$, and consider states q, r, s. Let σ_0 be a Duplicator strategy in the game $G_0 = G^x_{q,r}(\sqsubseteq_0, \sqsubseteq_1)$ between q and r, and let σ_1 be a Duplicator strategy in the game $G_1 = G^x_{r,s}(\sqsubseteq_1', \sqsubseteq_2)$ between r and s. If, additionally, Duplicator's $\sqsubseteq_1$-jumps in G_0 can be interpreted as Spoiler's $\sqsubseteq_1'$-jumps in G_1, then we can define a composite

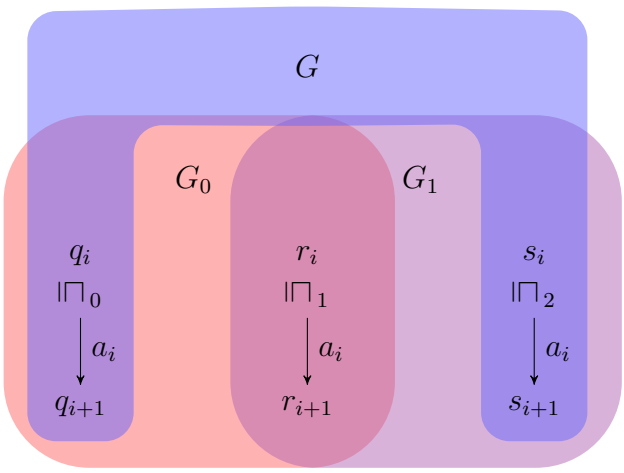


Figure 4.1: Composing games

strategy, denoted $\sigma_0 \bowtie \sigma_1$, for Duplicator in the game $G = G^x_{q,s}(\sqsubseteq_0, \sqsubseteq_2)$ between q and s. Therefore, we say that σ_0 and σ_1 are *composable* if, additionally, $\sqsubseteq_1 \subseteq \sqsubseteq'_1$.

We now define $\sigma_0 \bowtie \sigma_1$ for two composable strategies σ_0 and σ_1. Duplicator plays G and at the same time updates G_0, G_1 accordingly. In round i, if the G-configuration is $\langle q_i, s_i\rangle$, then there exists r_i s.t. the G_0-configuration is $\langle q_i, r_i\rangle$ and the G_1-configuration is $\langle r_i, s_i\rangle$. The i-th round of the game is shown in Figure 4.1. Let Spoiler in G_0 choose a $\sqsubseteq_0$-jumping transition $q_i \sqsubseteq_0 \xrightarrow{a_i} q_{i+1}$ in G. This is also a transition for Spoiler in G_0. By applying σ_0, we get a matching $\sqsubseteq_1$-jumping transition $r_i \sqsubseteq_1 \xrightarrow{a_i} r_{i+1}$ for Duplicator. Since $\sqsubseteq_1 \subseteq \sqsubseteq'_1$, the transition above is also $\sqsubseteq'_1$-jumping. Therefore, it can be interpreted as a transition by Spoiler in G_1. Thus, let Spoiler in G_1 take transition $r_i \sqsubseteq'_1 \xrightarrow{a_i} r_{i+1}$. By applying σ_1, we get a $\sqsubseteq_2$-jumping transition $s_i \sqsubseteq_2 \xrightarrow{a_i} s_{i+1}$ for Duplicator in G_1. Since the latter is also a transition for Duplicator in G, strategy $\sigma_0 \bowtie \sigma_1$ is defined as to play $s_i \sqsubseteq_2 \xrightarrow{a_i} s_{i+1}$ in G. The configurations are updated as follows: The game G_0 goes to $\langle q_{i+1}, r_{i+1}\rangle$, G_1 goes to $\langle r_{i+1}, s_{i+1}\rangle$ and G goes to $\langle q_{i+1}, s_{i+1}\rangle$.

The definition of composition for $x = \mathrm{bw}$ is analogous, and can be obtained by just inverting the direction of transitions.

The crucial property of the composition operator $\bowtie$ is that the composition of two winning strategies is winning.

Lemma 4.3.5. *For $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}, \mathrm{bw}\}$, if σ_0 and σ_1 are winning, then $\sigma_0 \bowtie \sigma_1$ is winning.*

Proof. First, notice that, since $\sqsubseteq_1 \subseteq \sqsubseteq'_1$, also $\sqsubseteq^F_1 \subseteq [\sqsubseteq'_1]^F$, which we often use below.

For $x = \mathrm{di}$, assume $q_i \sqsubseteq_0 \xrightarrow{a_i}_F q_{i+1}$. Since σ_0 is winning, $r_i \sqsubseteq_1 \xrightarrow{a_i}_F r_{i+1}$, thus $r_i \sqsubseteq'_1 \xrightarrow{a_i}_F r_{i+1}$. Since σ_1 is winning, $s_i \sqsubseteq_2 \xrightarrow{a_i}_F s_{i+1}$. This shows that $\sigma_0 \bowtie \sigma_1$ is

winning for $x = \mathrm{di}$.

For $x = \mathrm{bw}$, accepting states are propagated as above (by flipping transitions). For initial states, assume $q_i \sqsubseteq_0 I$. Since σ_0 is winning, $r_i \sqsubseteq_1 I$, therefore $r_i \sqsubseteq'_1 I$. Since σ_1 is winning, $s_i \sqsubseteq_2 I$. Therefore, $\sigma_0 \bowtie \sigma_1$ is winning for $x = \mathrm{bw}$.

For $x = \mathrm{de}$, assume $q_i \sqsubseteq_0 \xrightarrow{a_i}_F q_{i+1}$. Since σ_0 is winning in G_0, there exists $k \geq i$ s.t. $r_k \sqsubseteq_1 \xrightarrow{a_k}_F r_{k+1}$, thus $r_k \sqsubseteq'_1 \xrightarrow{a_k}_F r_{k+1}$. Since σ_1 is winning in G_1, there exists $j \geq k \geq i$ s.t. $s_j \sqsubseteq_2 \xrightarrow{a_j}_F s_{j+1}$. Thus, $\sigma_0 \bowtie \sigma_1$ is winning for $x = \mathrm{de}$.

Finally, for $x = \mathrm{f}$, assume $q_i \sqsubseteq_0 \xrightarrow{a_i}_F q_{i+1}$ for infinitely many i's. Since σ_0 is winning, $r_i \sqsubseteq_1 \xrightarrow{a_i}_F r_{i+1}$ for infinitely many i's, which implies $r_i \sqsubseteq'_1 \xrightarrow{a_i}_F r_{i+1}$ for infinitely many i's. Since σ_1 is winning, $s_i \sqsubseteq_2 \xrightarrow{a_i}_F s_{i+1}$ for infinitely many i's. Hence, $\sigma_0 \bowtie \sigma_1$ is winning for $x = \mathrm{f}$. □

As an immediate corollary, we have the following closure property of transformers.

Corollary 4.3.6. *Let $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}, \mathrm{bw}\}$. For preorders $\sqsubseteq_0, \sqsubseteq_1, \sqsubseteq'_1, \sqsubseteq_2$, assume $\sqsubseteq_1 \subseteq \sqsubseteq'_1$. Then,*

$$q\ \tau^x(\sqsubseteq_0, \sqsubseteq_1)\ r \ \textit{ and } \ r\ \tau^x(\sqsubseteq'_1, \sqsubseteq_2)\ s \quad \textit{ implies } \quad q\ \tau^x(\sqsubseteq_0, \sqsubseteq_2)\ s$$

By taking $\sqsubseteq = \sqsubseteq_0 = \sqsubseteq_1 = \sqsubseteq'_1 = \sqsubseteq_2$ in the corollary above, we have that $\tau^x(\sqsubseteq)$ is transitive. Since it is also clearly reflexive, it is a preorder.

Corollary 4.3.7. *Let $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}, \mathrm{bw}\}$. For any preorder $\sqsubseteq$, $\tau^x(\sqsubseteq)$ is a preorder.*

4.4 Language containment and inclusion

In this section, we relate jumping simulations to jumping containment and inclusion. In Section 4.4.1, we establish that jumping simulations are sound under-approximations of jumping containment (cf. Lemma 4.4.1). In Section 4.4.2, we show that jumping containment can be used to prove jumping language inclusion (cf. Lemma 4.4.2). Moreover, if the input preorder is jumping-safe, then also non-jumping language inclusion—that is, ordinary language inclusion—can be under-approximated (see Theorem 4.4.3 in Section 4.4.3). Thus, jumping simulations can be used as an under-approximation to language inclusion.

Going from backward containment to forward language inclusion is a non-trivial task, since it requires filling the gap between finite paths (as in backward containment) and infinite ones (as in ω-language inclusion). In Section 4.4.4, we convert a sequence

of longer and longer finite paths into a single infinite path; we discuss *coherence*, which is a sufficient condition for the infinite path to be fair.

4.4.1 Jumping simulation implies jumping containment

We start off by establishing that jumping simulations imply jumping containment. This is an analogue of the corresponding classic result about ordinary simulation preorders.

Lemma 4.4.1 (Simulation implies containment)**.** *Jumping simulations imply jumping containment. Formally, for* $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}, \mathrm{bw}\}$*, let* $\preceq^x = \tau^x(\sqsubseteq_0, \sqsubseteq_1)$*. Then, for two states* q *and* s*, and* $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$*,*

$$q \preceq^x s \quad \textit{implies} \quad q \subseteq^{\mathrm{fw}}(\sqsubseteq_0, \sqsubseteq_1)\ s$$
$$q \preceq^{\mathrm{bw}} s \quad \textit{implies} \quad q \subseteq^{\mathrm{bw}}(\sqsubseteq_0, \sqsubseteq_1)\ s$$

Proof. We first prove the lemma for forward transformers $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$. By the inclusions in Lemma 4.3.1, it suffices to consider $x = \mathrm{f}$. Let $q \preceq^{\mathrm{f}} s$, and assume $w \in \mathcal{L}^{\sqsubseteq_0}(q)$. That is, there exists a fair $\sqsubseteq_0$-jumping path

$$\pi = q_0 \sqsubseteq_0 \xrightarrow{a_0} q_1 \sqsubseteq_0 \xrightarrow{a_1} \cdots$$

starting at $q_0 = q$. In the simulation game, from the initial configuration $\langle q, s\rangle$, we let Spoiler play as to follow π. That is, in round i, Spoiler plays transition $q_i \sqsubseteq_0 \xrightarrow{a_i} q_{i+1}$. Since $q \preceq^{\mathrm{f}} s$, Duplicator has a winning strategy to reply with. Thus, Duplicator builds a $\sqsubseteq_1$-jumping path

$$\pi' = s_0 \sqsubseteq_1 \xrightarrow{a_0} s_1 \sqsubseteq_1 \xrightarrow{a_1} \cdots$$

starting at $s_0 = s$. Since π is fair, Spoiler plays accepting transitions infinitely often. But Duplicator is winning for $x = \mathrm{f}$, therefore also Duplicator plays accepting transitions infinitely often. Thus π' is fair as well, and $w \in \mathcal{L}^{\sqsubseteq_1}(s)$.

For $x = \mathrm{bw}$, let $q \preceq^{\mathrm{bw}} s$ and let

$$\pi = q_0 \sqsupseteq_0 \hat{q}_0 \xrightarrow{a_0} q_1 \sqsupseteq_0 \hat{q}_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{n-1}} q_n \sqsupseteq_0 \hat{q}_n, \text{ with } \hat{q}_n = q$$

be an initial $\sqsupseteq_0$-jumping path ending in q. The argument is the same as above. The simulation game starts from configuration $\langle q, s\rangle$, and Spoiler plays by choosing backward transitions according to π. Since $q \preceq^{\mathrm{bw}} s$, Duplicator has a winning strategy in

the simulation game. Therefore, Duplicator builds a matching initial $\sqsupseteq_1$-jumping path π' ending in s,

$$\pi' = \hat{s}_0 \xrightarrow{a_0} s_1 \sqsupseteq_1 \hat{s}_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{n-1}} s_n \sqsupseteq_1 \hat{s}_n, \text{ with } \hat{s}_n = s$$

By the definition of backward jumping simulation,

1) If $q_i \sqsupseteq_0^F \hat{q}_i$, then Spoiler is accepting: $\hat{q}_i \sqsubseteq_0^F q_i \xleftarrow{a_{i-1}} \hat{q}_{i-1}$. Since Duplicator is winning, she is accepting as well: $\hat{s}_i \sqsubseteq_1^F s_i \xleftarrow{a_{i-1}} \hat{s}_{i-1}$. Therefore, $s_i \sqsupseteq_1^F \hat{s}_i$.

2) Since q_0 is initial and $q_0 \sqsupseteq_0 \hat{q}_0 \preceq^{\text{bw}} \hat{s}_0$, there exists s_0 initial s.t. $s_0 \sqsubseteq_1 \hat{s}_0$.

Thus, take $\pi'' = s_0 \sqsupseteq_1 \hat{s}_0 \xrightarrow{a_0} s_1 \sqsupseteq_1 \hat{s}_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{n-1}} s_n \sqsupseteq_1 \hat{s}_n$ to be the initial $\sqsupseteq_1$-jumping path as required in the definition of backward containment. □

4.4.2 Jumping containment implies jumping inclusion

In the previous section, we have shown that forward and backward jumping simulations under-approximate forward and backward jumping containments, respectively. In this section, we give a condition under which jumping containment between states can show jumping inclusions between automata. Specifically, given two automata $\mathcal{Q}$ and $\mathcal{S}$, we show that,

- If every *initial* state of the former is in jumping *forward* containment with some initial state of the latter, then the jumping language of the former is included in the jumping language of the latter.
- Dually, if every *accepting* state of the former is in jumping *backward* containment with some accepting state of the latter, then the jumping language of the former is included in the jumping language of the latter.

Lemma 4.4.2. *Let $\mathcal{Q}$ and $\mathcal{S}$ be two nondeterministic automata. Then,*

$$\forall q \in I_{\mathcal{Q}} \cdot \exists s \in I_{\mathcal{S}} \cdot q \subseteq^{\text{fw}}(\sqsubseteq_0, \sqsubseteq_1)\, s \implies \mathcal{L}^{\sqsubseteq_0}(\mathcal{Q}) \subseteq \mathcal{L}^{\sqsubseteq_1}(\mathcal{S})$$
$$\forall q \in F_{\mathcal{Q}} \cdot \exists s \in F_{\mathcal{S}} \cdot q \subseteq^{\text{bw}}(\sqsubseteq_0, \sqsubseteq_1)\, s \implies \mathcal{L}^{\sqsupseteq_0}(\mathcal{Q}) \subseteq \mathcal{L}^{\sqsupseteq_1}(\mathcal{S})$$

The proof of the forward case is trivial. For the backward case, we need to link backward containment to forward language inclusion. This requires to move from the finite paths of backward containment to the infinite paths of language inclusion. We postpone this till Section 4.4.4, where we introduce *coherent sequences of finite paths*,

which are showed to induce infinite, fair paths. This allows us to prove the backward case of Lemma 4.4.2, whose proof is given at the end of Section 4.4.4.

In the next section, we draw some useful consequence of Lemma 4.4.2.

4.4.3 Deciding ordinary language inclusion

In the previous section, we have shown that jumping forward and backward containments can be used to establish jumping language inclusion between automata. Here, we specialize the approach to get good under-approximations of *ordinary* (i.e., non-jumping) language inclusion. This shows that jumping simulation, like ordinary simulation, can be used to prove language inclusion between automata.

Let $\mathcal{Q}$ and $\mathcal{S}$ be two automata. In Section 4.4.2, we have considered $\sqsubseteq_0$-jumping paths in $\mathcal{Q}$ and $\sqsubseteq_1$-jumping paths in $\mathcal{S}$, for any two preorders $\sqsubseteq_0$ and $\sqsubseteq_1$. Here, since we are interested in showing ordinary language inclusion between $\mathcal{Q}$ and $\mathcal{S}$, we consider ordinary paths in $\mathcal{Q}$ and jumping-safe paths in $\mathcal{S}$. That is, we take $\sqsubseteq_0 = Id$ and $\sqsubseteq_1 = \sqsubseteq$, where $\sqsubseteq$ is a jumping-safe preorder. Jumping-safety allows us to convert fair jumping paths in $\mathcal{S}$ into ordinary ones. It is important for $\sqsubseteq$ to be a subset of $S \times S$, for paths should not jump to the other automaton.

Therefore, we consider a simulation transformer $\preceq^x = \tau^x(Id, \sqsubseteq)$, where we allow only Duplicator to $\sqsubseteq$-jump during the simulation game. This induces a reflexive (but not necessarily transitive) relation $\preceq^x$, which is coarser than both $\sqsupseteq$, by Lemma 4.3.3, and the corresponding ordinary x-simulation $\sqsubseteq^x$, by Lemma 4.3.2. This asymmetry in the definition makes $\preceq^x$—and, in particular, $\preceq^{\mathrm{f}}$ and $\preceq^{\mathrm{bw}}$—potentially much coarser than any other known PTIME-computable under-approximations to language inclusion.

Theorem 4.4.3 (Jumping simulation implies language inclusion)**.** *Let $\mathcal{Q}$ and $\mathcal{S}$ be two nondeterministic automata.*

1. *Let $\sqsubseteq$ be a jumping-safe preorder on $\mathcal{S}$ (i.e., $\sqsubseteq \subseteq S \times S$). If every initial state of $\mathcal{Q}$ is forward $\sqsubseteq$-simulated by some initial state of $\mathcal{S}$, then the language of $\mathcal{Q}$ is included in that of $\mathcal{S}$.*

   Formally, for $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$, let $\preceq^x = \tau^x(Id, \sqsubseteq)$. Then,

$$\forall q \in I_{\mathcal{Q}} \cdot \exists s \in I_{\mathcal{S}} \cdot q \preceq^x s \implies \mathcal{L}(\mathcal{Q}) \subseteq \mathcal{L}(\mathcal{S})$$

2. *Let $\sqsupseteq$ be a jumping safe preorder on $\mathcal{S}$. If every accepting state of $\mathcal{Q}$ is backward $\sqsubseteq$-simulated by some accepting state of $\mathcal{S}$, then the language of $\mathcal{Q}$ is included in that of $\mathcal{S}$.*

Formally, let $\preceq^{\mathrm{bw}} = \tau^{\mathrm{bw}}(Id, \sqsubseteq)$. *Then,*

$$\forall q \in F_{\mathcal{Q}} \cdot \exists s \in F_{\mathcal{S}} \cdot q \preceq^{\mathrm{bw}} s \implies \mathcal{L}(\mathcal{Q}) \subseteq \mathcal{L}(\mathcal{S})$$

Proof. For the forward case, let $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$. By Lemma 4.4.1, $\tau^x(Id, \sqsubseteq)$ is included in forward containment $\subseteq^{\mathrm{fw}}(Id, \sqsubseteq)$. By Lemma 4.4.2, $\mathcal{L}^{Id}(\mathcal{Q}) \subseteq \mathcal{L}^{\sqsubseteq}(\mathcal{S})$. But $\mathcal{L}^{Id}(\mathcal{Q}) = \mathcal{L}(\mathcal{Q})$ and, since $\sqsubseteq$ is jumping-safe, $\mathcal{L}^{\sqsubseteq}(\mathcal{S}) = \mathcal{L}(\mathcal{S})$. Thus, $\mathcal{L}(\mathcal{Q}) \subseteq \mathcal{L}(\mathcal{S})$.

The backward case is analogous. □

Remark 4.4.4. Since $\sqsubseteq$ is assumed to be jumping-safe, and jumping-safe preorders are GFQ, one might wonder whether it is more convenient to first quotient $\mathcal{S}$ w.r.t. the equivalence induced by $\sqsubseteq$, and only then check inclusion with ordinary forward (or backward) simulation $\tau^x(Id, Id)$. The answer is negative: Even after quotienting, there might still be (inequivalent) $\sqsubseteq$-related states, and this can be exploited by Duplicator while playing the game for $\tau^x(Id, \sqsubseteq)$. We believe that this asymmetry has a great potential in efficiently under-approximating language inclusion.

4.4.4 Coherent sequences of jumping paths

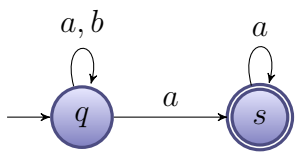
In this section, we present a general method to show the existence of fair paths based on the existence of certain sequences of *finite* paths. We apply this technique to prove Lemma 4.4.2 from Section 4.4.2.

Fix an infinite word $w = a_0 a_1 \cdots \in \Sigma^\omega$. The idea is to start with a sequence of longer and longer finite initial paths $\Pi := \pi_0, \pi_1, \ldots$ over suitable prefixes of w. We are interested in finding a sufficient condition for the existence of an initial and fair infinite path over w. Since fair paths have infinitely many accepting states, a necessary condition is that the number of accepting states in paths π_i's grows unboundedly.

In the case of deterministic automata, this condition is also sufficient: Indeed, in a deterministic automaton there exists a unique run over w, which is accepting precisely when the number of accepting stated visited by its prefixes goes to infinity. In this case, we say that the π_i's are *strongly coherent*, since longer paths conservatively extends shorter ones.

However, in the general case of nondeterministic automata it is quite possible to have paths that visit arbitrarily many accepting states, and, still, no accepting run exists. This occurs because accepting states can appear arbitrarily late in the path, as shown in the next example.

Example 4.4.1 - Visiting arbitrarily many accepting states is not sufficient



Consider the automaton $\mathcal{Q}$ above. Take the infinite word $w = aba^2ba^3b\cdots$. For every prefix of the form $w_i = aba^2b\cdots a^{i-1}ba^i$, there exists a w_i-path

$$\pi_i = q \xrightarrow{ab} q \xrightarrow{a^2b} \cdots \xrightarrow{a^{i-1}b} \underbrace{q \xrightarrow{a} s \xrightarrow{a} s \xrightarrow{a} \cdots \xrightarrow{a} s}_{i \text{ times}}$$

visiting the accepting state s i times. Still, no fair path exists over w. Therefore $w \notin \mathcal{L}(\mathcal{Q})$.

The issue is that accepting states appear just in the tail of the path, and they never "stabilize" in any prefix of bounded length. To prevent this, we require accepting states to spread uniformly across the path. We split the infinite time horizon into slices $0 < j_0 < j_1 < \cdots$, and we require that for each interval $[0, j_i]$ and index $k \geq i$, path π_k visits at least i accepting states within the first j_i steps. See Figure 4.2. When this condition is satisfied, we say that the sequence of paths Π is *coherent*.

We give the formal definitions for the more general case of jumping paths. For a (finite or infinite) $\sqsubseteq$-jumping path $\pi = q_0 \sqsubseteq \xrightarrow{a_0} q_1 \sqsubseteq \xrightarrow{a_1} \cdots$, let $\#F(\pi, j)$ be the number of accepting transitions within the first j transitions:

$$\#F(\pi, j) := \sum_{0 \leq i < j} [q_i \sqsubseteq \xrightarrow{a_i}_F q_{i+1}]$$

Definition 4.4.5. *Fix a word $w \in \Sigma^\omega$. Let $\Pi := \pi_0, \pi_1, \ldots$ be an infinite sequence of finite $\sqsubseteq$-jumping w-paths. Π is* coherent *if the following property holds:*

$$\forall i \cdot \exists (j > i) \cdot \forall (k \geq i) \cdot \#F(\pi_k, j) \geq i \,. \tag{4.1}$$

Let Π be defined as above. Sometimes we are interested in infinite subsequences of Π, obtained by removing some of its paths (possibly infinitely many). Formally, Π' is an *infinite subsequence of* Π iff $\Pi' := \pi_{f(0)}, \pi_{f(1)}, \ldots$ for some $f : \omega \mapsto \omega$ with $f(0) < f(1) < \cdots$. Coherent sequences are obviously preserved under the operation of taking infinite subsequences.

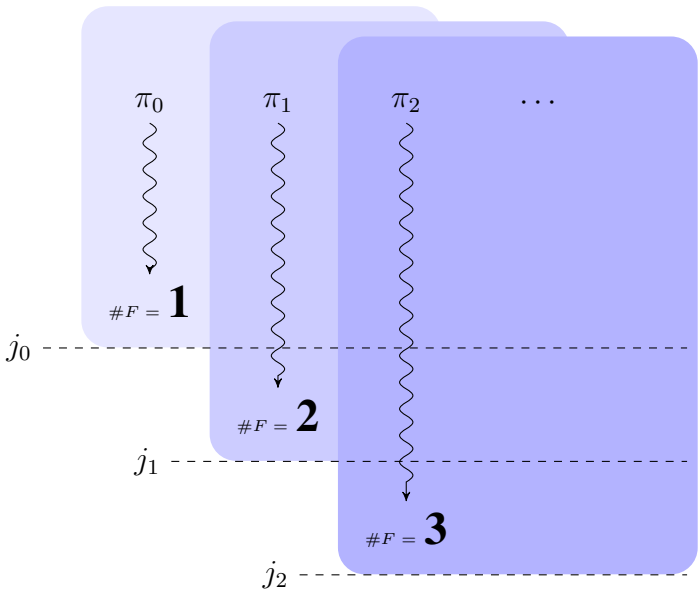


Figure 4.2: Coherent paths

Lemma 4.4.6. *If Π is coherent, then any infinite subsequence Π' thereof is coherent.*

The main result of this section states that coherence is strong enough for the existence of fair paths.

Lemma 4.4.7. *Fix a word $w \cdots \in \Sigma^\omega$ and a finitely-branching automaton $\mathcal{Q}$. If $\Pi := \pi_0, \pi_1, \ldots$ is a coherent sequence of $\sqsubseteq$-jumping w-paths, then there exists a fair, $\sqsubseteq$-jumping w-path ρ. Moreover, if all π_i's are initial, then ρ is initial.*

Proof. Let $\Pi = \pi_0, \pi_1, \ldots$ be a coherent $\sqsubseteq$-jumping sequence of w-paths. Consider index j_0. Since the π_k's are branches in a finitely branching tree, there are only finitely many different prefixes of length j_0. Therefore, there exists a prefix ρ_0 which is common to infinitely many paths. Let $\Pi' = \pi'_0, \pi'_1, \ldots$ be the infinite subsequence of Π containing only suffixes of ρ_0. By construction, ρ_0 contains at least 1 accepting transition, and each π' in Π' extends ρ_0. By Lemma 4.4.6, Π' is coherent. Now consider index j_1. We can apply the same reasoning again to Π', and we obtain a longer prefix ρ_1 extending ρ_0, which contains at least 2 accepting transitions. Let Π'' be the coherent subsequence of Π' containing only suffixes of ρ_1, and so on. In this fashion, we obtain an infinite sequence of *strongly* coherent (finite) $\sqsubseteq$-jumping paths $\rho_0, \rho_1, \cdots$ s.t. ρ_i extends ρ_{i-1} and contains at least $i + 1$ accepting transitions. The infinite path to which the sequence converges $\rho = \lim_i \rho_i$ is the fair $\sqsubseteq$-jumping path we are after. $\square$

The machinery of coherent sequences is used to finally prove Lemma 4.4.2 from Section 4.4.2.

Proof of Lemma 4.4.2. Forward case. For any initial state $q \in I_{\mathcal{Q}}$ of $\mathcal{Q}$, let $s_q \in I_{\mathcal{S}}$ be a corresponding initial state of $\mathcal{S}$ s.t. $q \subseteq^{\mathrm{fw}}(\sqsubseteq_0, \sqsubseteq_1)\ s_q$, which exists by assumption. By the definition of containment, $\mathcal{L}^{\sqsubseteq_0}(q) \subseteq \mathcal{L}^{\sqsubseteq_1}(s_q)$. Then,

$$\mathcal{L}^{\sqsubseteq_0}(\mathcal{Q}) = \bigcup_{q \in I_{\mathcal{Q}}} \mathcal{L}^{\sqsubseteq_0}(q) \ \subseteq \ \bigcup_{s_q \in I_{\mathcal{S}}} \mathcal{L}^{\sqsubseteq_1}(s_q) \ \subseteq \ \mathcal{L}^{\sqsubseteq_1}(\mathcal{S})$$

Backward case. Let $\preceq^{\mathrm{bw}} = \subseteq^{\mathrm{bw}}(\sqsubseteq_0, \sqsubseteq_1)$, and assume $w = a_0 a_1 \cdots \in \mathcal{L}^{\sqsupseteq_0}(\mathcal{Q})$. There exists an initial and fair $\sqsupseteq_0$-jumping path $\pi = \hat{q}_0 \sqsupseteq_0 q_0 \xrightarrow{a_0} \hat{q}_1 \sqsupseteq_0 q_1 \xrightarrow{a_1} \cdots$. Let $k_0 < k_1 < \cdots$ be any infinite sequence s.t., for any $i \geq 0$, $\hat{q}_{k_i} \sqsupseteq_0^F q_{k_i}$. This sequence exists since π is fair. For each $i \geq 0$, there exists an accepting proxy $p_i \in F_{\mathcal{Q}}$ s.t. $\hat{q}_{k_i} \sqsupseteq_0 p_i \sqsupseteq_0 q_{k_i}$. Let $w_i = a_0 a_1 \cdots a_{k_i - 1}$, and consider the following prefix of π over w_i

$$\pi_i = \hat{q}_0 \sqsupseteq_0 q_0 \xrightarrow{a_0} \hat{q}_1 \sqsupseteq_0 q_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{k_i - 1}} \hat{q}_{k_i} \sqsupseteq_0^F p_i$$

By assumption, there exists a corresponding accepting proxy $t_i \in F_{\mathcal{S}}$ s.t. $p_i \preceq^{\mathrm{bw}} t_i$. Since π_i is an initial, $\sqsupseteq_0$-jumping path ending at p_i, by the definition of backward containment there exists a corresponding initial, $\sqsupseteq_1$-jumping path π'_i ending at t_i,

$$\pi'_i = \hat{s}_{i,0} \sqsupseteq_1 s_{i,0} \xrightarrow{a_0} \hat{s}_{i,1} \sqsupseteq_1 s_{i,1} \xrightarrow{a_1} \cdots \xrightarrow{a_{k_i - 1}} \hat{s}_{i,k_i} \sqsupseteq_1^F t_i$$

s.t., for any $0 \leq k < k_i$, if $\hat{q}_k \sqsupseteq_0^F q_k$, then $\hat{s}_{i,k} \sqsupseteq_1^F s_{i,k}$.

Take $\Pi = \pi'_0, \pi'_1, \ldots$. Clearly, for any $i \geq 0$, π'_i contains at least $i + 1$ accepting transitions within k_i steps. Thus, Π is coherent. As we have already observed, each π'_i is initial. By Lemma 4.4.7, there exists an initial and fair $\sqsupseteq_1$-jumping w-path π'. Therefore, $w \in \mathcal{L}^{\sqsupseteq_1}(\mathcal{S})$. □

4.5 Reflexivity and jumping-safety

The central result of this section is that transformers preserve jumping-safe preorders. See Theorems 4.5.3 and 4.5.4 in Section 4.5.2. This is the main result of the chapter, which makes jumping simulations interesting and useful.

We derive the above theorems as an easy consequence of a crucial property of jumping simulation games, which we refer to as *reflexivity*. Reflexivity is a main technical tool in dealing with jumping simulations; it studied in Section 4.5.1.

In the rest of this section, fix a preorder $\sqsubseteq$, and let $\preceq^x = \tau^x(\sqsubseteq)$ for $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{bw}, \mathrm{f}\}$.

4.5.1 Reflexivity

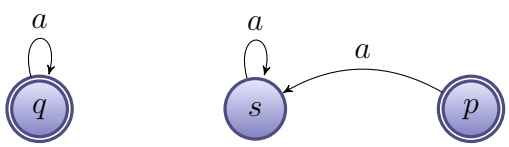
We study a modified simulation transformer where Spoiler can jump more than Duplicator. Nonetheless, we establish that it is reflexive, that is that Duplicator wins from diagonal configurations $\langle q, q\rangle$.

We have already observed that $\preceq^x$ is reflexive (for any $\sqsubseteq$); that is, for any state q, $q \preceq^x q$ holds. Recall that, by Lemma 4.3.3, $\succeq^x$ (the transpose of $\preceq^x$) is at least as coarse as $\sqsubseteq$. It turns out that even the modified transformer $\tau^x(\succeq^x, \sqsubseteq)$, where Spoiler has the capability of taking $\succeq^x$-jumps which are "longer" than Duplicator's $\sqsubseteq$-jumps, is reflexive.

Lemma 4.5.1 (Reflexivity)**.** *For a preorder $\sqsubseteq$ and $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{bw}\}$, let $\preceq^x = \tau^x(\sqsubseteq)$. Then, $\tau^x(\succeq^x, \sqsubseteq)$ is reflexive, i.e., for any state q, $q\ \tau^x(\succeq^x, \sqsubseteq)\ q$.*

This is a crucial property of simulation transformers, and it is used throughout the rest of the chapter. We prove it later in Section 4.5.3. Note that Lemma 4.5.1 fails for the fair transformer, as the next example shows.

Example 4.5.1 - The transformer $\tau^{\mathrm{f}}(\succeq^{\mathrm{f}}, \sqsubseteq)$ is not reflexive



Consider the unary automaton above. Let $\sqsubseteq = Id$ and let $\preceq^{\mathrm{f}}$ be ordinary fair simulation $\preceq^{\mathrm{f}} = \tau^{\mathrm{f}}(Id)$. Since $\Sigma = \{a\}$, the only possible infinite word is a^ω. Notice that there is no fair a^ω path, neither from s nor from p. Therefore, s fairly simulates p, $s \succeq^{\mathrm{f}} p$, for the simple reason that Spoiler cannot build a fair path from p. However, since $s \succeq^{\mathrm{f}} p$, if we consider the fair $(\succeq^{\mathrm{f}}, Id)$-jumping game starting from configuration $\langle s, s\rangle$, Spoiler can now play the following fair $\succeq^{\mathrm{f}}$-jumping path from s:

$$\pi = s \succeq^{\mathrm{f}} p \xrightarrow{a}_F s \succeq^{\mathrm{f}} p \xrightarrow{a}_F \cdots$$

Since no ordinary fair path exists from s, Duplicator loses. Therefore, $s\ \tau^{\mathrm{f}}(\succeq^{\mathrm{f}}, Id)\ s$ does not hold.

The corollary below is an immediate consequence of Lemma 4.5.1, and it is at the heart of the central preservation property of Theorems 4.5.3 and 4.5.4 of Section 4.5.2.

Corollary 4.5.2. *For a preorder $\sqsubseteq$ and $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{bw}\}$, let $\preceq^x = \tau^x(\sqsubseteq)$. Then, for $y \in \{\mathrm{di}, \mathrm{de}\}$, $\subseteq^{\mathrm{fw}}(\succeq^y, \sqsubseteq)$ and $\subseteq^{\mathrm{bw}}(\succeq^{\mathrm{bw}}, \sqsubseteq)$ are reflexive.*

Proof. For any state q, by Lemma 4.5.1, $q\ \tau^x(\succeq^x, \sqsubseteq)\ q$, and, by Lemma 4.4.1, jumping simulation implies jumping containment. □

4.5.2 Preserving jumping-safe preorders

In this section we state the central result of this chapter: *direct, delayed and backward simulation transformers preserve jumping-safe preorders*. See Theorems 4.5.3 and 4.5.4. The proofs are elementary; they crucially use the reflexivity property of Corollary 4.5.2.

This opens the possibility of repeatedly applying simulation transformers to the identity relation (which is trivially jumping-safe) to get coarser and coarser jumping-safe preorders; this is explored in Section 4.6.

Forward transformers

Theorem 4.5.3. *The (the transpose of) direct and delayed jumping transformer preserve jumping-safe preorders. Formally, for a preorder $\sqsubseteq$ and $x \in \{\mathrm{di}, \mathrm{de}\}$, let $\preceq^x = \tau^x(\sqsubseteq)$. If $\sqsubseteq$ is a jumping-safe preorder, then $\succeq^x$ is a jumping-safe preorder coarser than $\sqsubseteq$.*

In analogy with forward simulations, it is the transpose of $\preceq^x$ which is jumping-safe, not $\preceq^x$ itself.

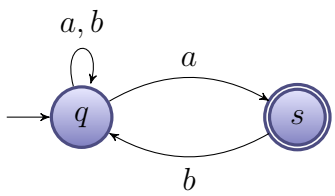
Proof. Let $\sqsubseteq$ be jumping-safe. By Lemma 4.3.3, $\succeq^x$ is coarser than $\sqsubseteq$. We show that $\succeq^x$ is jumping safe, i.e., $\mathcal{L}^{\succeq^x}(\mathcal{Q}) = \mathcal{L}(\mathcal{Q})$. The right-to-left direction is immediate. For the other direction,

$$\mathcal{L}^{\succeq^x}(\mathcal{Q}) = \bigcup_{q \in I} \mathcal{L}^{\succeq^x}(q) \overset{(0)}{\subseteq} \bigcup_{q \in I} \mathcal{L}^{\sqsubseteq}(q) = \mathcal{L}^{\sqsubseteq}(\mathcal{Q}) \overset{(1)}{=} \mathcal{L}(\mathcal{Q})$$

where inclusion (0) follows from $q \subseteq^x(\succeq^x, \sqsubseteq)\ q$ (by Corollary 4.5.2), and equality (1) holds since $\sqsubseteq$ is jumping-safe. □

Theorem 4.5.3 fails for the fair transformer. In the example below we show that this holds already by taking as input the identity relation $\sqsubseteq = Id$.

Example 4.5.2 - The fair transformer does not preserve jumping-safe preorders



It is well-known that ordinary fair simulation $\sqsubseteq^{\mathrm{f}} = \tau^{\mathrm{f}}(Id)$ is not a GFQ preorder [44]. It turns out that this happens precisely because $\sqsupseteq^f$ is not jumping-safe. Consider the automaton above. State q recognizes the language $(\Sigma^* ab)^\omega$, and $q \sqsupseteq^{\mathrm{f}} s$. Consider $w = b^\omega$; w admits the following fair $\sqsupseteq^f$-jumping path from q:

$$\pi = q \sqsupseteq^f s \xrightarrow{b} q \sqsupseteq^f s \xrightarrow{b} \cdots$$

Therefore, $w \in \mathcal{L}^{\sqsupseteq^f}(q)$. But $w \notin \mathcal{L}(q)$, since there is no ordinary fair path from q over w. Therefore, $\sqsupseteq^f$ is not jumping-safe.

Backward transformer

Theorem 4.5.4. *The backward jumping simulation transformer preserves (the transpose of) jumping-safe preorders. Formally, for a preorder $\sqsubseteq$, let $\preceq^{\mathrm{bw}} = \tau^{\mathrm{bw}}(\sqsubseteq)$. If $\sqsupseteq$ is a jumping-safe preorder, then $\preceq^{\mathrm{bw}}$ is a jumping-safe preorder coarser than $\sqsupseteq$.*

Proof. By Lemma 4.3.3, $\preceq^{\mathrm{bw}}$ is coarser than $\sqsupseteq$. Since $\sqsupseteq$ is jumping-safe, it suffices to show $\mathcal{L}^{\preceq^{\mathrm{bw}}}(\mathcal{Q}) = \mathcal{L}^{\sqsupseteq}(\mathcal{Q})$. The right-to-left direction follows directly from the inclusion $\sqsupseteq \subseteq \preceq^{\mathrm{bw}}$ (by Lemma 4.3.3). For the other direction, we use Lemma 4.4.2, where we take both automata to be $\mathcal{Q}$: Let q be any accepting state. By Corollary 4.5.2, $q \subseteq^{\mathrm{bw}}(\succeq^{\mathrm{bw}}, \sqsubseteq)\, q$. Therefore, taking $s = q$ satisfies the premise of Lemma 4.4.2. Thus, $\mathcal{L}^{\preceq^{\mathrm{bw}}}(\mathcal{Q}) \subseteq \mathcal{L}^{\sqsupseteq}(\mathcal{Q})$. □

4.5.3 Proof of reflexivity

In this section, we prove Lemma 4.5.1. We do so by showing a winning strategy for Duplicator in $G^x_{q,q}(\succeq^x, \sqsubseteq)$. Recall that $\preceq^x = \tau^x(\sqsubseteq)$, and in the game $G^x_{q,q}(\succeq^x, \sqsubseteq)$ Spoiler takes $\succeq^x$-jumps while Duplicator takes $\sqsubseteq$-jumps. Therefore, since $\succeq^x$ is at least

as coarse as $\sqsubseteq$, we need to show that Duplicator wins even if Spoiler is allowed to "jump more".

The following lemma states two basic properties of this special $(\succeq^x, \sqsubseteq)$-jumping games. It is the counterpart of Lemma 4.3.4 from Section 4.3.

Lemma 4.5.5. *Let $q \preceq^x s$.*

a) Let $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$. For every $\succeq^x$-jumping forward transition $q \succeq^x \xrightarrow{a} q'$, there exists a $\sqsubseteq$-jumping forward transition $s \sqsubseteq \xrightarrow{a} s'$ s.t. $q' \preceq^x s'$.

For $x = \mathrm{di}$, if $q \succeq^{\mathrm{di}} \xrightarrow{a}_F q'$, then s' can be chosen s.t. $s \sqsubseteq \xrightarrow{a}_F s'$.

b) Let $x = \mathrm{bw}$. For every $\succeq$-jumping backward transition $q \succeq^{\mathrm{bw}} \xleftarrow{a} q'$, there exists a $\sqsubseteq$-jumping backward transition $s \sqsubseteq \xleftarrow{a} s'$ s.t. $q' \preceq^{\mathrm{bw}} s'$.

Moreover, if $q \succeq^{\mathrm{bw}} \xleftarrow{a}_F q'$, then s' can be chosen s.t. $s \sqsubseteq \xleftarrow{a}_F s'$.

Proof. For readability, we write just $\preceq$ instead of $\preceq^x$.

Point a). Fix a $\succeq$-jumping forward transition from q, as in the top row on the right. Since $q \succeq \hat{q}$ and $\hat{q} \xrightarrow{a} q'$, by Lemma 4.3.4(a), there exists a $\sqsubseteq$-jumping forward transition from q, as in the middle row. In particular, notice that $q' \preceq r'$. Since $q \preceq s$, by Lemma 4.3.4(a) again, we obtain the $\sqsubseteq$-jumping transition from s in the bottom row. Notice that $q' \preceq r' \preceq s'$. By transitivity (cf. Corollary 4.3.7), $q' \preceq s'$.

$$\begin{array}{ccccc} q & \succeq & \hat{q} & \xrightarrow{a} & q' \\ \parallel & & & & \rotatebox{90}{$\preceq$} \\ q & \sqsubseteq & \hat{r} & \xrightarrow{a} & r' \\ \rotatebox{90}{$\preceq$} & & & & \rotatebox{90}{$\preceq$} \\ s & \sqsubseteq & \hat{s} & \xrightarrow{a} & s' \end{array}$$

For $x = \mathrm{di}$, assume $q \succeq \xrightarrow{a}_F q'$. There exists q^F accepting s.t. $q \succeq q^F \succeq \hat{q}$. By the previous part, there exists a transition $q^F \sqsubseteq \xrightarrow{a}_F t'$ with $q' \preceq t'$. The transition is accepting for $q^F \in F$. Since $q^F \preceq q \preceq s$, by transitivity (cf. Corollary 4.3.7) $q^F \preceq s$. By Lemma 4.3.4(a), there exists an accepting transition $s \sqsubseteq \xrightarrow{a}_F s'$ with $t' \preceq s'$, and, thus, $q' \preceq s'$. The proof for Point b) is identical, by invoking Lemma 4.3.4(b). □

As an immediate consequence, we have that Duplicator has $\succeq^x$-preserving strategies (cf. Section 2.3) in the game $G^x_{q,q}(\succeq^x, \sqsubseteq)$.[1]

Corollary 4.5.6. *For $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}, \mathrm{bw}\}$, let $\preceq^x = \tau^x(\sqsubseteq)$. For any state q, there exists a $\preceq^x$-respecting strategy for Duplicator in $G^x_{q,q}(\succeq^x, \sqsubseteq)$.*

[1] This is analogous to Corollary 6 of [55] in the context of alternating Büchi automata.

However, while winning strategies are $\succeq^x$-preserving, the converse does not hold in general. Nonetheless, for $x \in \{\mathrm{di}, \mathrm{bw}\}$, we can construct a winning strategy with an inductive use of Lemma 4.5.5.

Proof of Lemma 4.5.1 (for $x \in \{\mathrm{di}, \mathrm{bw}\}$). Let us stipulate that in round i the game is in configuration $\langle q_i, s_i \rangle$. The initial configuration is $\langle q_0, s_0 \rangle = \langle q, q \rangle$. In round i, Duplicator ensures the invariant $q_i \preceq s_i$. For $i = 0$, the invariant clearly holds. For $i > 0$, assume $q_i \preceq s_i$.

For $x = \mathrm{di}$, let Spoiler take a $\succeq$-jumping forward transition $q_i \succeq \xrightarrow{a_i} q_{i+1}$. By Lemma 4.5.5(a), Duplicator has a matching $\sqsubseteq$-jumping forward transition $s_i \sqsubseteq \xrightarrow{a_i} s_{i+1}$ s.t. $q_{i+1} \preceq s_{i+1}$. The invariant is preserved. For the winning condition, if Spoiler takes an accepting transition, so can do Duplicator.

For $x = \mathrm{bw}$, transitions are matched backwards in a similar way, by invoking Lemma 4.5.5(b). For the winning condition, 1) propagation of accepting states is clear. Moreover, 2) for initial states, if $q_i \succeq I$, then, from the invariant and transitivity (cf. Corollary 4.3.7), $s_i \succeq I$. Therefore, by the definition of $\preceq$, $s_i \sqsubseteq I$. □

The case $x = \mathrm{de}$ For the delayed transformer, winning strategies have a more complex structure. In the rest of this section, let $\preceq = \preceq^{\mathrm{de}}$. The idea is that, by appropriately composing a $\preceq$-respecting strategy and a winning strategy, we get a winning strategy in $G^{\mathrm{de}}_{q,q}(\succeq, \sqsubseteq)$ (cf. [55]).

We need two preliminary facts. First, since $\preceq$ is transitive, composition preserves the property of being $\preceq$-respecting.

Lemma 4.5.7. *If σ_0 and σ_1 are $\preceq$-respecting (and composable), then $\sigma_0 \bowtie \sigma_1$ is $\preceq$-respecting.*

Second, de-winning strategies propagate accepting states under composition on the left.

Lemma 4.5.8. *Let σ^0, σ^1 be two composable strategies for Duplicator in $G^{\mathrm{de}}_{q_0,r_0}(\sqsubseteq_0, \sqsubseteq_1)$ and $G^{\mathrm{de}}_{r_0,s_0}(\sqsubseteq'_1, \sqsubseteq_2)$, respectively. If $r_0 \in F$ and σ^1 is a winning strategy, then, for any $(\sigma^0 \bowtie \sigma^1)$-conform play $\pi = \pi_0 \times \pi_2$ from $\langle q_0, s_0 \rangle$,*

$$\pi_0 = q_0 \sqsubseteq_0 \xrightarrow{a_0} q_1 \sqsubseteq_0 \xrightarrow{a_1} \cdots \qquad \pi_2 = s_0 \sqsubseteq_2 \xrightarrow{a_0} s_1 \sqsubseteq_2 \xrightarrow{a_1} \cdots$$

Duplicator is eventually accepting, *i.e., there exists $i \geq 0$ s.t. $s_i \sqsubseteq_2 \xrightarrow{a_i}_F s_{i+1}$.*

Proof. Let $\pi = \pi_0 \times \pi_2$ be any $(\sigma^0 \bowtie \sigma^1)$-conform play, Moreover, let $\pi_1 = r_0 \sqsubseteq_1 \xrightarrow{a_0} r_1 \sqsubseteq_1 \xrightarrow{a_1} \cdots$ be the intermediate play. It follows that $\pi_1 \times \pi_2$ is σ^1-conform. Since $r \in F$ and σ^1 is winning, there exists $i \geq 0$ s.t. $s_i \sqsubseteq_2 \xrightarrow{a_i}_F s_{i+1}$. □

We can now finish the proof for the last case of Lemma 4.5.1.

Proof of Lemma 4.5.1 (for $x = \mathrm{de}$). The idea is to split the play in $G^{\mathrm{de}}_{q,q}(\succeq, \sqsubseteq)$ into stages $k_0 < k_1 < \cdots$ (starting at $k_0 = 0$), and define a sequence of strategies $\sigma_0, \sigma_1, \ldots,$ s.t., during the i-th stage, Duplicator plays according to the i-th strategy. Stage i starts when, in round k_i, Spoiler is accepting. Therefore, during stage i, there is a pending obligation for Duplicator to visit an accepting state. When this obligation is eventually fulfilled by σ_i, the next stage can start as soon as Spoiler is accepting again. When this does not happen, there are only finitely many stages. Otherwise, a new stage $i+1$ starts when Spoiler is accepting in round k_{i+1}, and Duplicator switches to the next strategy σ_{i+1}.

In round i, we stipulate that the current configuration of the game is $\langle q_i, s_i \rangle$, where, initially, $\langle q_0, s_0 \rangle = \langle q, q \rangle$. Moves of Spoiler take the form of $\succeq$-jumping transitions $q_i \succeq \xrightarrow{a_i} q_{i+1}$, whereas Duplicator's responses are $\sqsubseteq$-jumping transitions of the form $s_i \sqsubseteq \xrightarrow{a_i} s_{i+1}$.

Formally, we define two sequences of indices $\{k_i\}_{i \geq 0}$ and $\{h_i\}_{i \geq 0}$ by induction. Initially, let $k_0 = h_0 = -1$. For $i \geq 0$,

$$k_{i+1} = \min(\{j > h_i \mid q_j \succeq \xrightarrow{a_j}_F q_{j+1}\} \cup \{\omega\})$$
$$h_{i+1} = \min(\{j \geq k_{i+1} \mid s_j \sqsubseteq \xrightarrow{a_j}_F s_{j+1}\} \cup \{\omega\})$$

Intuitively, h_i is the time Duplicator matches the i-th pending obligation and k_{i+1} is the time Spoiler raises the next, $(i+1)$-th pending obligation. If the i-th pending obligation is not eventually fulfilled, then $k_i < h_i = k_{i+1} = \cdots = \omega$. If the i-th pending obligation is the last one to be raised and it is eventually fulfilled, then $h_i < k_{i+1} = h_{k+1} = \cdots = \omega$. Otherwise, when infinitely many obligations are raised and fulfilled, the sequence does not converge. In general, the two sequences are thus interleaving:

$$k_0 \leq h_0 < k_1 \leq h_1 < \cdots$$

At any stage $i \geq 0$,

- Let $q_0^F = q_0$ and, for $i > 0$ and $k_i < \omega$, let $q_i^F \in F$ be the accepting state s.t.

  $$q_{k_i} \succeq q_i^F \succeq \xrightarrow{a_{k_i}} q_{k_i+1}$$

  which exists by the definition of k_i.

- Let σ_i^0 be a $\preceq$-respecting strategy in the game $G^{\mathrm{de}}_{q_i^F, q_i^F}(\succeq, \sqsubseteq)$, which exists by Corollary 4.5.6.

- Assuming $q_i^F \preceq s_{k_i}$, let σ_i^1 be a *winning strategy* in the game $G^{\mathrm{de}}_{q_i^F, s_{k_i}}(\sqsubseteq, \sqsubseteq)$.

Let k be the current round, and let $\pi = \pi_0 \times \pi_1$ be the current partial play, where

$$\pi_0 = q_0 \succeq \xrightarrow{a_0} q_1 \succeq \xrightarrow{a_1} q_2 \succeq \xrightarrow{a_2} \cdots \succeq \xrightarrow{a_{k-1}} q_k$$
$$\pi_1 = s_0 \sqsubseteq \xrightarrow{a_0} s_1 \sqsubseteq \xrightarrow{a_1} s_2 \sqsubseteq \xrightarrow{a_2} \cdots \sqsubseteq \xrightarrow{a_{k-1}} s_k$$

Let Spoiler play the jumping transition $q_k \succeq \xrightarrow{a_k} q_{k+1}$. Assume the game is in stage i, that is, $k_i \leq k < k_{i+1}$. Let $\pi[k_i, k] = \pi'_0 \times \pi'_1$ be the modified suffix of π starting in round k_i, where

$$\pi'_0 = q_i^F \succeq \xrightarrow{a_{k_i}} q_{k_i+1} \succeq \xrightarrow{a_{k_i+1}} \cdots \succeq \xrightarrow{a_{k-1}} q_k$$
$$\pi'_1 = s_{k_i} \sqsubseteq \xrightarrow{a_{k_i}} s_{k_i+1} \sqsubseteq \xrightarrow{a_{k_i+1}} \cdots \sqsubseteq \xrightarrow{a_{k-1}} s_k$$

Then, Duplicator plays according to the following strategy σ_i:

$$\sigma_i(\pi) := [\sigma_i^0 \bowtie \sigma_i^1](\pi[k_i, k])$$

Notice that π'_0 starts at q_i^F (and not at q_{k_i}), since σ_i^0 is a strategy starting from configuration $\langle q_i^F, q_i^F \rangle$.

σ_i^0 and σ_i^1 are composable. To show that σ_i is well-defined, we need to ensure that σ_i^1 always exists, i.e., that $q_i^F \preceq s_{k_i}$ holds throughout the game. This holds initially, since both q_0^F and s_{k_0} equal q, and $\preceq$ is reflexive. Inductively, assume $q_i^F \preceq s_{k_i}$ holds. Therefore, σ_i is well-defined. Since during stage i the play is σ_i-conform, and both components σ_i^0, σ_i^1 are $\preceq$-respecting, by Lemma 4.5.7, σ_i is $\preceq$-respecting. Therefore, for any round k with $k_i \leq k \leq k_{i+1}$, $s_k \preceq t_k$. In particular, at the end of the stage $q_{k_{i+1}} \preceq s_{k_{i+1}}$. By the definition of q_{i+1}^F, $q_{i+1}^F \preceq q_{k_{i+1}}$, and, by transitivity (cf. Corollary 4.3.7), $q_{i+1}^F \preceq s_{k_{i+1}}$.

Finally, to show that σ_i is winning, it suffices to show that, for any $i > 0$, if $k_i < \omega$, then $h_i < \omega$. Therefore, assume k_i is finite, which implies s_i^F is defined and accepting. By Lemma 4.5.8, there exists $j \geq k_i$ s.t. $s_j \sqsubseteq^F \hat{s}_j$. Take h_i to be a minimal such j. $\square$

4.6 Proxy simulation hierarchies

We have seen in Section 4.5.2 that the direct, delayed and backward transformers preserve jumping-safe preorders. A natural idea is to compose transformers in order to get even coarser jumping-safety-preserving transformers: One can start with a given jumping-safe preorder $\sqsubseteq$, and repeatedly apply these transformers (in some order);

since transformers are non-decreasing (by Lemma 4.3.3), one obtains a non-decreasing chain of jumping-safe preorders.

In Section 4.6.1, we show that transformers are idempotent; thus, we rule out the possibility of applying the same transformer twice. In Section 4.6.2, we consider hierarchies obtained by composing alternately forward and backward transformers starting from the identity relation; this gives rise to *proxy simulations*. Finally, in Section 4.6.3, we compare proxy simulations against iteratively quotienting the automaton w.r.t. ordinary forward and backward simulations.

4.6.1 Idempotence

We show that transformers are idempotent (up to transposition); thus, there is no benefit in composing the same transformer with itself.

Lemma 4.6.1 (Idempotence)**.** *The direct, delayed, and backward jumping transformers are idempotent (up to transposition). Formally, for a preorder* $\sqsubseteq$ *and* $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{bw}\}$*, let* $\preceq = \tau^x(\sqsubseteq)$*. Then,* $\tau^x(\succeq) = \preceq$*.*

The lemma above can be interpreted by saying that the coarser $\succeq$-jumps do not give more power to Duplicator. Compare this with Lemma 4.5.1, where we have shown that coarser $\succeq$-jumps do not give more power to Spoiler either.

Proof. We have to show $\tau^x(\succeq) = \preceq$. The right-to-left inclusion follows directly from Lemma 4.3.3. For the other direction, let us expand the definitions and rewrite it as $\tau^x(\succeq, \succeq) \subseteq \tau^x(\sqsubseteq, \sqsubseteq)$. Assume $q \; \tau^x(\succeq, \succeq) \; s$. Then,

$$\underbrace{q \; \tau^x(\sqsubseteq, \succeq) \; q}_{(0)} \; \tau^x(\succeq, \succeq) \; \underbrace{s \; \tau^x(\succeq, \sqsubseteq) \; s}_{(1)}$$

where (0) holds trivially since Spoiler can "jump less" than Duplicator ($\sqsubseteq \subseteq \succeq$ by Lemma 4.3.3), and (1) holds by Lemma 4.5.1. By Corollary 4.3.6 twice, $q \; \tau^x(\sqsubseteq, \sqsubseteq) \; s$. That is, $q \preceq s$. □

For $x = \mathrm{f}$, the coarser $\succeq$-jumps *do* give more power to Duplicator, as the next example shows.

Example 4.6.1 - The fair transformer τ^{f} is not idempotent

Consider the automaton from Example 4.5.1 again. Since q has a fair path over a^ω but s doesn't, the latter cannot fairly simulate the former, i.e., $q \not\preceq^{\mathrm{f}} s$. However, since $s \succeq^{\mathrm{f}} p$, Duplicator can play the following fair $\succeq^{\mathrm{f}}$-jumping path over a^ω from s:

$$\pi = s \succeq^{\mathrm{f}} p \xrightarrow{a}_F s \succeq^{\mathrm{f}} p \xrightarrow{a}_F \cdots$$

Therefore, $q\ \tau^{\mathrm{f}}(\succeq^{\mathrm{f}})\ s$ holds and τ^{f} is not idempotent, $\tau^{\mathrm{f}}(\succeq^{\mathrm{f}}) \neq \preceq^{\mathrm{f}}$.

Thus, we only compose different transformers. However, also composing direct with the delayed transformer is not better than the delayed transformer alone.

Lemma 4.6.2. *For $x \in \{\mathrm{di}, \mathrm{de}\}$, let $\preceq^x = \tau^x(\sqsubseteq)$. Then,*

$$\tau^{\mathrm{di}}(\succeq^{\mathrm{de}}) \subseteq \preceq^{\mathrm{de}}$$
$$\tau^{\mathrm{de}}(\succeq^{\mathrm{di}}) \subseteq \preceq^{\mathrm{de}}$$

Proof. For the first inclusion, assume $q\ \tau^{\mathrm{di}}(\succeq^{\mathrm{de}})\ s$. Then, by Lemma 4.3.1, $q\ \tau^{\mathrm{de}}(\succeq^{\mathrm{de}})\ s$, and, by Lemma 4.6.1, $q \preceq^{\mathrm{de}} s$.

For the second inclusion, assume $q\ \tau^{\mathrm{de}}(\succeq^{\mathrm{di}})\ s$. That is, $q\ \tau^{\mathrm{de}}(\succeq^{\mathrm{di}}, \succeq^{\mathrm{di}})\ s$. Obviously, $q\ \tau^{\mathrm{de}}(\sqsubseteq, \succeq^{\mathrm{di}})\ q$ (since $\sqsubseteq \subseteq \succeq^{\mathrm{di}}$ by Lemma 4.3.3). Also, $s\ \tau^{\mathrm{di}}(\succeq^{\mathrm{di}}, \sqsubseteq)\ s$ by Lemma 4.5.1, and, by Lemma 4.3.1, $s\ \tau^{\mathrm{de}}(\succeq^{\mathrm{di}}, \sqsubseteq)\ s$. Putting the pieces together,

$$q\ \tau^{\mathrm{de}}(\sqsubseteq, \succeq^{\mathrm{di}})\ q\ \tau^{\mathrm{de}}(\succeq^{\mathrm{di}}, \succeq^{\mathrm{di}})\ s\ \tau^{\mathrm{de}}(\succeq^{\mathrm{di}}, \sqsubseteq)\ s$$

By Corollary 4.3.6, $q\ \tau^{\mathrm{de}}(\sqsubseteq, \sqsubseteq)\ s$; that is, $q \preceq^{\mathrm{de}} s$. □

Therefore, we restrict ourselves to composing forward and backward transformers in a strictly alternating fashion, as we explore in the next section.

4.6.2 Proxy simulations

For two transformers τ and τ', let their composition be $\tau; \tau'$, where τ is applied first:

$$\tau; \tau'(\sqsubseteq) = \tau'(\tau(\sqsubseteq))$$

As basic building blocks we define the four composite transformers below, obtained by composing a forward transformer with the backward one:

$$\tau^{\mathrm{di+bw}} := \tau^{\mathrm{di}}; \tau^{\mathrm{bw}} \qquad \tau^{\mathrm{bw+di}} := \tau^{\mathrm{bw}}; \tau^{\mathrm{di}}$$
$$\tau^{\mathrm{de+bw}} := \tau^{\mathrm{de}}; \tau^{\mathrm{bw}} \qquad \tau^{\mathrm{bw+de}} := \tau^{\mathrm{bw}}; \tau^{\mathrm{de}}$$

Forward proxy simulations are obtained by applying a cascade of $\tau^{\mathrm{di+bw}}/\tau^{\mathrm{de+bw}}$ operations starting from the identity relation. See Figure 4.3(a), where each node in the tree is a forward proxy simulation. Backward proxy simulations are defined similarly, but w.r.t. transformers $\tau^{\mathrm{bw+di}}/\tau^{\mathrm{bw+de}}$. See Figure 4.3(b).

Definition 4.6.3. *$\preceq$ is a* forward proxy simulation *iff there exists a sequence of transformers $\tau_0\tau_1\ldots\tau_n \in \{\tau^{\mathrm{di+bw}}, \tau^{\mathrm{de+bw}}\}^*$ s.t. $\preceq\, = \tau_0;\tau_1;\cdots;\tau_n(Id)$. Similarly, $\preceq$ is a* backward proxy simulation *iff there exists a sequence of transformers $\tau_0\tau_1\ldots\tau_n \in \{\tau^{\mathrm{bw+di}}, \tau^{\mathrm{bw+de}}\}^*$ s.t. $\preceq\, = \tau_0;\tau_1;\cdots;\tau_n(Id)$.*

Proxy simulations are jumping-safe by an immediate inductive argument from Theorems 4.5.3 and 4.5.4.

Lemma 4.6.4. *If $\preceq$ is a forward or backward proxy simulation, then $\succeq$ (resp., $\preceq$) is jumping-safe. In particular, $\preceq$ is GFQ.*

Remark 4.6.5. In previous work [28], we introduced two preorders, which we called direct and delayed proxy simulations. They coincide with the first level $\tau^{\mathrm{bw+di}}(Id)$ and $\tau^{\mathrm{bw+de}}(Id)$ of the backward hierarchy, respectively. Here, we obtain an entire hierarchy of coarser jumping-safe PTIME preorders, thus improving upon [28].

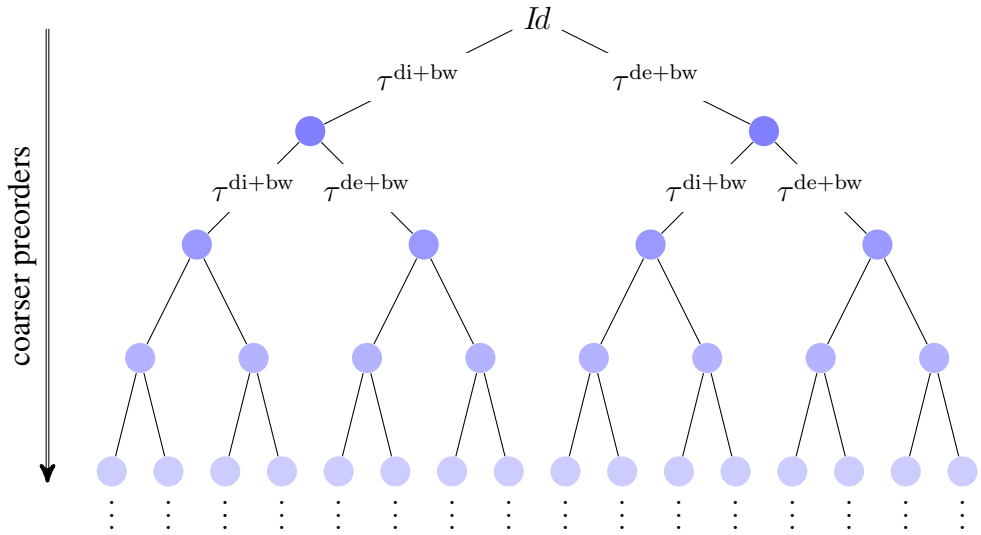
Proxy simulations along a branch of the tree form a non-decreasing chain of jumping-safe preorders. That is, if $\tau_0\tau_1\cdots\tau_n$ is a prefix of $\tau_0\tau_1\cdots\tau_{n'}$, with $n' \geq n$, then $\tau_0;\tau_1;\cdots;\tau_n(Id) \subseteq \tau_0;\tau_1;\cdots;\tau_{n'}(Id)$. On finite automata, these chains eventually stabilize after a finite number of steps. For any automaton, there exists a uniform n s.t., for any sequence of transformers $\tau_0\tau_1\cdots$,

$$\tau_0;\tau_1;\cdots;\tau_n(Id) = \tau_0;\tau_1;\cdots;\tau_{n+1}(Id)$$

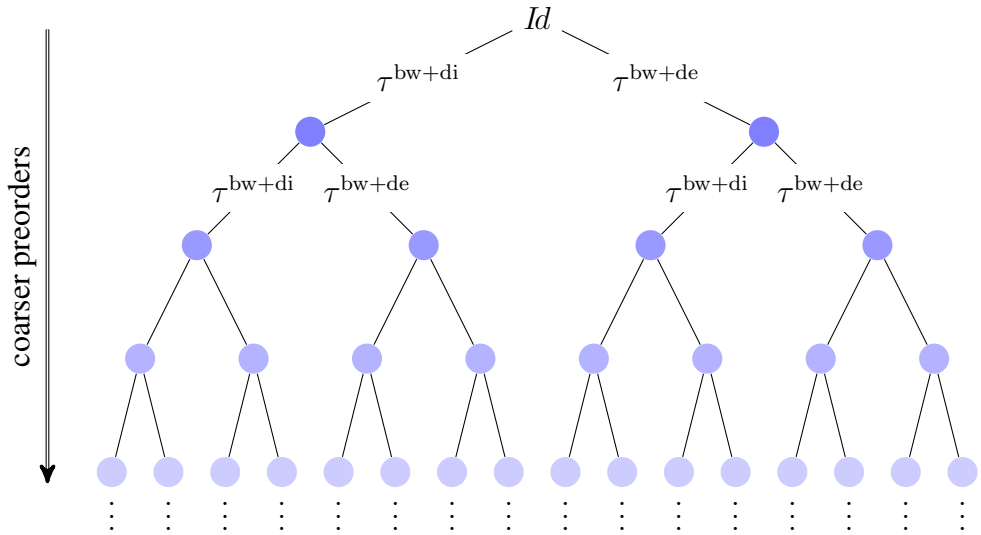
We say that a proxy simulation $\preceq$ is *maximal* iff $\preceq\, = \tau_0;\tau_1;\cdots;\tau_n(Id)$, with n as above.

Maximal simulations have the property of being a fixpoint of both a forward and a backward transformer. We exemplify this in the case of forward proxy simulations. Let $\preceq$ be a maximal proxy simulation in the forward hierarchy. Then, $\preceq\, = \tau_0;\tau_1;\cdots;\tau_n(Id) = \tau_0;\tau_1;\cdots;\tau_{n+1}(Id)$, with $\tau_{n+1} = \tau^{x+\mathrm{bw}}$, for $x \in \{\mathrm{di}, \mathrm{de}\}$. Consequently,

$$\begin{aligned}
\preceq\, \subseteq\, [\tau^x(\preceq)]^{-1} &\subseteq \tau^{\mathrm{bw}}(\tau^x(\preceq)) && \text{(by Lemma 4.3.3 twice)}\\
&= \tau^x;\tau^{\mathrm{bw}}(\preceq) && \text{(by definition of composition)}\\
&= \preceq && \text{(since } \preceq \text{ is maximal)}
\end{aligned}$$



(a) Forward hierarchy



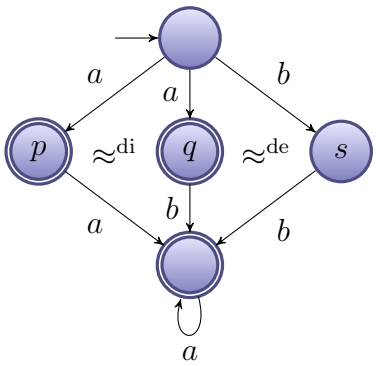
(b) Backward hierarchy

Figure 4.3: Proxy simulations hierarchies

Thus, $\preceq = [\tau^x(\preceq)]^{-1} = \tau^{\mathrm{bw}}(\tau^x(\preceq))$. That is, $\succeq = \tau^x(\preceq)$ and $\preceq = \tau^{\mathrm{bw}}(\succeq)$. Therefore, $\preceq$ is a fixpoint of both τ^{bw} and τ^x (up to transposition). For maximal backward proxy simulations $\preceq$'s, an analogous argument shows that $\preceq = \tau^x(\succeq)$ and $\succeq = \tau^{\mathrm{bw}}(\preceq)$.

Proxy simulations along different branches are generally incomparable preorders. Thus, we get a great number of maximal, incomparable proxy simulations, and it is not clear if there is any reason to prefer any one of them over the others. Moreover, incomparable simulations cannot be simultaneously used for quotienting; this happens already in the first level of the hierarchy, as shown for the forward hierarchy in the next example. A similar example can be given for the backward hierarchy.

Example 4.6.2 - Simultaneous quotienting w.r.t. $\tau^{\mathrm{di+bw}}(Id)$ and $\tau^{\mathrm{de+bw}}(Id)$ is incorrect



Consider the automaton above. Let $\preceq^{\mathrm{di}} = \tau^{\mathrm{di+bw}}(Id)$ and $\preceq^{\mathrm{de}} = \tau^{\mathrm{de+bw}}(Id)$, and let $\approx^{\mathrm{di}}$ and $\approx^{\mathrm{de}}$ be the corresponding induced equivalences. Direct and delayed simulation are trivially the identity, except for states q and s which are delayed simulation equivalent. Therefore, they are also $\preceq^{\mathrm{de}}$-equivalent, $q \approx^{\mathrm{de}} s$.

Since direct simulation is the identity, $\preceq^{\mathrm{di}}$ is just backward simulation. States p and q are $\preceq^{\mathrm{di}}$-equivalent, and the only ones to be related by $\preceq^{\mathrm{di}}$. However, they are not $\preceq^{\mathrm{de}}$-equivalent: In particular, $q \not\preceq^{\mathrm{de}} p$, since Spoiler can jump from q to s (since s delayed-simulates q) and take the backward b transition therefrom, which is not available from p.

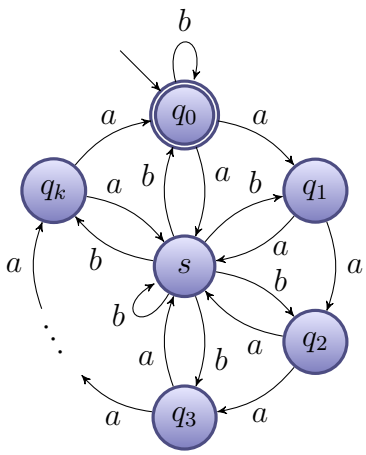
Hence, we have only the following non-trivial equivalences:

$$p \approx^{\mathrm{di}} q \approx^{\mathrm{de}} s$$

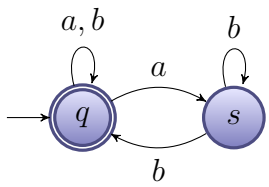
However, like in Example 4.1.1, quotienting w.r.t. the equivalence induced by the transitive closure of $\approx^{\mathrm{di}} \cup \approx^{\mathrm{de}}$ is incorrect: Indeed, it would put p, q and s in the same equivalence class, and the new word ba^ω, not in the language of the original automaton, would be incorrectly accepted by the quotient automaton.

In the next example, we show that proxy simulation quotients can outperform other simulation-like quotients.

Example 4.6.3 - Proxy simulation quotients achieve arbitrarily high compression ratios



a) Family of automata $\mathcal{Q}_k$



b) The quotients $\mathcal{Q}_k/\approx$

We show that backward proxy simulation quotients can outperform many other quotient techniques based on simulation-like preorders. Consider the family of automata $\mathcal{Q}_k$ above. Each automaton has $k+2$ states. We consider quotienting $\mathcal{Q}_k$ w.r.t. various simulation-like preorders.

Direct and delayed simulation No two different states are comparable w.r.t. delayed simulation (and, thus, w.r.t. direct simulation). Indeed, no state can simulate q_0: From q_0, Spoiler can play both actions a and b, whereas from states $s, q_1, q_2, \ldots, q_k$ only one of these actions is available. This implies that no two different states in the ring are comparable: From configuration $\langle q_i, q_j \rangle$ with $i \neq j$, Spoiler repeatedly plays a until forcing configuration $\langle q_0, q_h \rangle$, for some $h \neq 0$; from the latter configuration, Spoiler wins as above. (We assume that Duplicator does not go to s, which would make her lose early.) Finally, s cannot simulate any state q_i (since action a is unavailable from s),

and no state q_i can simulate s: If $i \neq 0$, q_i cannot do b, and if $i = 0$, Spoiler goes from s to q_1 with action b and then wins as above.

Multipebble and fixed-word simulations Duplicator does not benefit from either having multiple pebbles, or from knowing the input word in advance. Indeed, Spoiler's choices are actually independent of Duplicator's moves (if Duplicator does not lose early). In other words, Duplicator loses even if she knows in advance what transition will be played next. Consequently, no two states are in multipebble or fixed-word simulation. This also shows that proxy simulations are incomparable with the latter.

Backward simulation Also, no two different states are backward simulation equivalent. For example, states in the ring are backward incomparable, since different states can only reach the initial state via a different number of a's. Also, no state in the ring can simulate s: From configuration $\langle s, q_i \rangle$, it suffices for Spoiler to take transition $s \xleftarrow{a} q_i$, and then we are in the previous case. So, backward simulation quotienting does not help either.

While no previous quotienting method managed to reduce the size of $\mathcal{Q}_k$, we finally show that backward proxy simulations can.

Backward proxy simulations We have observed above that there are no two backward simulation-equivalent states. However, backward simulation is not the identity itself. Indeed, s backward simulates all the states in the ring except q_0:

$$q_1, q_2, \ldots, q_k \sqsubseteq^{\mathrm{bw}} s$$

Indeed, if Spoiler takes any transition $q_{i+1} \xleftarrow{a} q_i$, then Duplicator can reply with $s \xleftarrow{a} q_i$, and similarly if Spoiler goes to s via action b. This gives more power to Duplicator in the delayed proxy simulation game, with the consequence that any two states in the ring are $\tau^{\mathrm{bw+de}}(Id)$-equivalent. To see why, notice that the ability of $\sqsubseteq^{\mathrm{bw}}$-jumping to s before taking a transition effectively adds an edge b between the following states:

$$q_1, q_2, \ldots, q_k \xrightarrow{b} q_0, q_1, \ldots, q_k$$

Let $\approx$ be the equivalence induced by $\tau^{\mathrm{bw+de}}(Id)$. To show $q_i \approx q_j$, we describe how Duplicator can force infinitely many visits to the accepting state q_0. We distinguish two cases.

- In the first case, consider configurations of the form $\langle s, q_j \rangle$. Spoiler has to play action b, and Duplicator takes a jumping b-transition $q_j \sqsubseteq^{\mathrm{bw}} s \xrightarrow{b} q_0$ to q_0, which is accepting. Duplicator stays in q_0 as long as Spoiler plays transition $s \xrightarrow{b} s$. If at any point Spoiler plays transition $s \xrightarrow{b} q_0$, then we are in configuration $\langle q_0, q_0 \rangle$, from which Duplicator clearly wins.

- In the second case, consider configurations of the form $\langle q_i, q_j \rangle$, with $i \neq j$. As long as Spoiler plays action a, Duplicator does the same and stays in the ring. In this way, she will eventually visit q_0. Otherwise, if at any point Spoiler plays action b (from some configuration of the form $\langle q_0, q_k \rangle$ or $\langle s, q_k \rangle$), then Duplicator jumps to q_0 immediately, and we are either in configuration $\langle q_0, q_0 \rangle$, which is immediately winning for Duplicator, or in $\langle s, q_0 \rangle$, as in the previous case.

  Notice that Duplicator never actually "stops" in state s, but only transits through it as to take jumping b-transitions.

Therefore, by letting $\approx$ be the equivalence induced by $\tau^{\mathrm{de+bw}}(Id)$, we have

$$q_0 \approx q_1 \approx q_2 \approx \cdots \approx q_k$$

The quotient automaton $\mathcal{Q}_k/{\approx}$ on the right has only 2 states, and $\tau^{\mathrm{de+bw}}(Id)$-quotienting achieves arbitrarily high compression ratios.

Notice that the direct counterpart $\tau^{\mathrm{bw+di}}(Id)$ does not help in this example, since states in the ring are $\tau^{\mathrm{bw+di}}(Id)$-incomparable: Indeed, from configuration $\langle q_i, q_j \rangle$, with $i \neq j$, Spoiler can play action a until she reaches the accepting state q_0, and Duplicator will not be accepting at that time. Thus, this example also shows that $\tau^{\mathrm{bw+de}}(Id)$ quotients can be better than $\tau^{\mathrm{bw+di}}(Id)$ quotients.

Forward proxy simulations Since delayed and direct simulation are just the identity, $\tau^{\mathrm{di+bw}}(Id)$ and $\tau^{\mathrm{de+bw}}(Id)$ both coincide with backward simulation; the forward hierarchy thus coincides with the backward one, and the same considerations as above can be made.

4.6.3 Iterated quotienting

We compare quotienting with proxy simulation against what we call iterated quotienting. In *iterated quotienting*, forward and backward simulation equivalences are repeatedly used to quotient the automaton, until no two states are either forward or backward

equivalent.

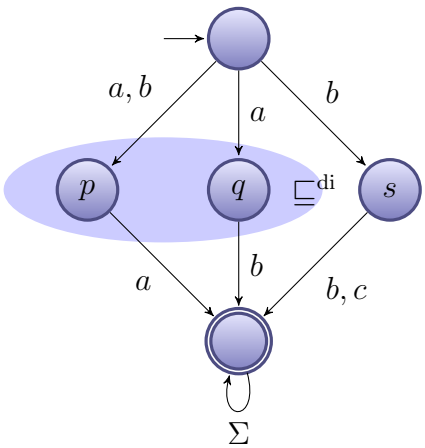
Fix a starting automaton $\mathcal{Q}$; for $x \in \{\mathrm{di}, \mathrm{de}\}$, let $\approx^x$ be x-simulation equivalence and let $\approx^{\mathrm{bw}}$ be backward simulation equivalence. Iterated quotienting gives rise to two hierarchies of automata, depending on whether we start by quotienting w.r.t. forward or backward simulation: Let $\mathcal{Q}_0 = \mathcal{S}_0 = \mathcal{Q}$, and, for $i \geq 0$,

$$\mathcal{Q}_{i+1} = \mathcal{S}_i / \approx^x \text{ , where } \approx^x \text{ is computed on } \mathcal{S}_i$$
$$\mathcal{S}_{i+1} = \mathcal{Q}_i / \approx^{\mathrm{bw}} \text{ , where } \approx^{\mathrm{bw}} \text{ is computed on } \mathcal{Q}_i$$

Remark 4.6.6. Note that quotienting w.r.t. forward simulation disrupts backward simulation, and vice versa; thus, one needs to recompute the simulation at every step.

In general, iterated quotienting and quotienting w.r.t. proxy simulations are incomparable. Below, we show an example where proxy simulations perform better.

Example 4.6.4 - Proxy quotients can outperform iterated quotienting



Consider the automaton above. No two states are either forward or backward simulation equivalent. Thus, iterated quotienting does not produce any reduction.

There is not even any so-called "little brother" [19], w.r.t. neither forward nor backward simulation. (A state s' is a forward little brother if there exist states q and q' s.t. $q \xrightarrow{a} q'$, $q \xrightarrow{a} s'$ and $s' \sqsubseteq^{\mathrm{di}} q'$. Similarly, a state s' is a backward little brother if there exist states q and q' s.t. $q \xleftarrow{a} q'$, $q \xleftarrow{a} s'$ and $s' \sqsubseteq^{\mathrm{bw}} q'$.)

However, quotienting w.r.t. proxy simulation merges states p and q. This is shown in detail later in Example 4.7.2.

However, it turns out that iterated quotienting is closely related to another hierarchy of GFQ relations. Let $\approx_0^{\text{fw-}x} = \approx_0^{\text{bw-}x} = Id$, and, for $i \geq 0$,

$$\approx_{i+1}^{\text{fw-}x} = \preceq \cap \succeq, \text{ where } \preceq = \tau^x(\approx_i^{\text{bw-}x}) \tag{4.2}$$

$$\approx_{i+1}^{\text{bw-}x} = \preceq \cap \succeq, \text{ where } \preceq = \tau^{\text{bw}}(\approx_i^{\text{fw-}x}) \tag{4.3}$$

That is, we build a hierarchy similar to proxy simulation, but before going to the next level, we take the induced equivalence of the current preorder.

The following lemma holds for any equivalence $\approx$. It says that $\tau^x(\approx)$ computed on $\mathcal{Q}$ is the same as $\tau^x(Id)$ computed on $\mathcal{Q}/\approx$. Notice that the latter is just ordinary x-simulation.

Lemma 4.6.7. *Let $\approx$ be any equivalence on $\mathcal{Q}$, and consider the quotient automaton $\mathcal{Q}/\approx$. For $x \in \{\text{di}, \text{de}, \text{bw}\}$,*

$$\underbrace{q\ \tau^x(\approx)\ s}_{\text{in } \mathcal{Q}} \iff \underbrace{[q]\ \tau^x(Id)\ [s]}_{\text{in } \mathcal{Q}/\approx}$$

The following lemma says that the i-th automaton after iterated quotienting corresponds to $\mathcal{Q}$ quotiented by the i-th equivalence in the hierarchy of Equations (4.2) and (4.3).

Lemma 4.6.8. *For any $i \geq 0$,*

$$\mathcal{Q}_i = \mathcal{Q}/\approx_i^{\text{fw-}x} \qquad \textit{and} \qquad \mathcal{S}_i = \mathcal{Q}/\approx_i^{\text{bw-}x}$$

Proof. By induction. For $i = 0$ it holds trivially. For $i \geq 0$, we have

$$\begin{aligned} \mathcal{Q}_{i+1} &= \mathcal{S}_i/\approx^{\text{fw-}x} \qquad \text{(by induction hypothesis)} \\ &= (\mathcal{Q}/\approx_i^{\text{bw-}x})/\approx^{\text{fw-}x} \\ &= \mathcal{Q}/\approx_{i+1}^{\text{fw-}x} \end{aligned}$$

where the last step follows from Lemma 4.6.7, since computing forward simulation on $\mathcal{Q}/\approx_i^{\text{bw-}x}$ is the same as computing $\tau^x(\approx_i^{\text{bw-}x})$ on $\mathcal{Q}$, whose induced equivalence is $\approx_{i+1}^{\text{fw-}x}$. The calculation for $\mathcal{S}_{i+1}$ is analogous. □

4.7 Proxy simulations vs Mediated preorder

In this section we compare proxy simulations against another GFQ preorder which has been studied in literature called *mediated preorder* [4]. In Section 4.7.1 we recall the definition of mediated preorder for nondeterministic automata. In Section 4.7.2 we compare in detail quotienting w.r.t. mediated preorder and proxy simulations; in general, the two approaches are incomparable. Finally, in Section 4.7.3 we show how mediated preorder can be interpreted as a variant of proxy simulation.

4.7.1 Mediated preorder

Mediated preorder has been originally introduced and studied in the context of alternating Büchi automata [4]. It arises as a combination of direct and backward simulation, in a spirit not far from jumping simulations. In the context of nondeterministic automata, it can be defined as follows. (With "$\circ$" we denote relational composition: Given two binary relations R_0 and R_1, $(x, z) \in (R_0 \circ R_1)$ iff $\exists y \cdot (x, y) \in R_0 \wedge (y, z) \in R_1$.)

Definition 4.7.1. *Let $\sqsubseteq^{\mathrm{di}}$ be direct simulation and let $\sqsubseteq^{\mathrm{bw}}$ be backward simulation. A binary relation R is a* mediated simulation *iff*

1) $R \subseteq \sqsubseteq^{\mathrm{di}} \circ \sqsupseteq^{\mathrm{bw}}$ (where $\sqsupseteq^{\mathrm{bw}}$ is the transpose of $\sqsubseteq^{\mathrm{bw}}$), and

2) $R \circ \sqsubseteq^{\mathrm{di}} \subseteq R$.

In other words, if R is a mediated simulation and $q\ R\ s$ for two states q and s, then

1) There exists a state $\hat{q}$ s.t. $q \sqsubseteq^{\mathrm{di}} \hat{q}$ and $s \sqsubseteq^{\mathrm{bw}} \hat{q}$. State $\hat{q}$ is called a *mediator*, and depends in general on q and on s.

2) For any state $\hat{s}$ s.t. $s \sqsubseteq^{\mathrm{di}} \hat{s}$, we have $q\ R\ \hat{s}$.

Mediated simulations are closed under union, and *mediated preorder* $\sqsubseteq^{\mathrm{M}}$ is defined as the union of all mediated simulations, and, therefore, the largest such simulation.

$\sqsubseteq^{\mathrm{M}}$ is correctly called a preorder. First, it is clearly reflexive, since the identity relation is a mediated simulation. Second, is is also transitive: Indeed, the composite relation $\preceq := \sqsubseteq^{\mathrm{M}} \circ \sqsubseteq^{\mathrm{M}}$ is included in $\sqsubseteq^{\mathrm{M}}$. This is established by showing that $\preceq$ is itself a mediated simulation, which can be done with the following calculations: Condition 2)

$\preceq \circ \sqsubseteq^{\mathrm{di}} \subseteq \preceq$ follows from

$$\begin{aligned}
\preceq \circ \sqsubseteq^{\mathrm{di}} &= (\sqsubseteq^{\mathrm{M}} \circ \sqsubseteq^{\mathrm{M}}) \circ \sqsubseteq^{\mathrm{di}} && \text{by associativity} \\
&= \sqsubseteq^{\mathrm{M}} \circ (\sqsubseteq^{\mathrm{M}} \circ \sqsubseteq^{\mathrm{di}}) && \text{by 2) on } \sqsubseteq^{\mathrm{M}} \\
&\subseteq \sqsubseteq^{\mathrm{M}} \circ \sqsubseteq^{\mathrm{M}} = \preceq
\end{aligned}$$

and condition 1) $\preceq \subseteq \sqsubseteq^{\mathrm{di}} \circ \sqsupseteq^{\mathrm{bw}}$ follows from

$$\begin{aligned}
\preceq &= \sqsubseteq^{\mathrm{M}} \circ \sqsubseteq^{\mathrm{M}} && \text{by 1) on } \sqsubseteq^{\mathrm{M}} \\
&\subseteq \sqsubseteq^{\mathrm{M}} \circ (\sqsubseteq^{\mathrm{di}} \circ \sqsupseteq^{\mathrm{bw}}) && \text{by associativity} \\
&= (\sqsubseteq^{\mathrm{M}} \circ \sqsubseteq^{\mathrm{di}}) \circ \sqsupseteq^{\mathrm{bw}} && \text{by 2) on } \sqsubseteq^{\mathrm{M}} \\
&\subseteq \sqsubseteq^{\mathrm{M}} \circ \sqsupseteq^{\mathrm{bw}} && \text{by 1) on } \sqsubseteq^{\mathrm{M}} \\
&\subseteq (\sqsubseteq^{\mathrm{di}} \circ \sqsupseteq^{\mathrm{bw}}) \circ \sqsupseteq^{\mathrm{bw}} && \text{by associativity} \\
&= \sqsubseteq^{\mathrm{di}} \circ (\sqsupseteq^{\mathrm{bw}} \circ \sqsupseteq^{\mathrm{bw}}) && \text{since } \sqsupseteq^{\mathrm{bw}} \text{ is transitive} \\
&\subseteq \sqsubseteq^{\mathrm{di}} \circ \sqsupseteq^{\mathrm{bw}}
\end{aligned}$$

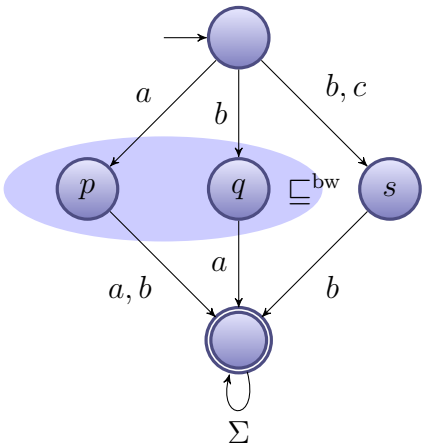
Finally, mediated preorder is at least as coarse as forward direct simulation. This follows directly from the fact that $\sqsubseteq^{\mathrm{di}}$ is itself a mediated simulation: 1) $\sqsubseteq^{\mathrm{di}} \subseteq \sqsubseteq^{\mathrm{di}} \circ \sqsupseteq^{\mathrm{bw}}$, and 2) $\sqsubseteq^{\mathrm{di}} \circ \sqsubseteq^{\mathrm{di}} \subseteq \sqsubseteq^{\mathrm{di}}$ (by transitivity of $\sqsubseteq^{\mathrm{di}}$). [4] establishes that mediated preorder can be used for quotienting.

Theorem 4.7.2. *$\sqsubseteq^{\mathrm{M}}$ is a GFQ preorder coarser than $\sqsubseteq^{\mathrm{di}}$.*

4.7.2 Comparison

We compare proxy simulation w.r.t. mediated preorder quotients. In general, the two approaches are incomparable. For example, mediated preorder and the backward simulation $\tau^{\mathrm{bw+di}}(Id)$ are incomparable; this is not surprising, as the former is at least as coarse as direct simulation, while the latter is at least as coarse as backward simulation. Below, we show an example where proxy simulation is better than mediated preorder.

Example 4.7.1 - Backward proxy simulation quotients can outperform mediated quotients



Consider the automaton above. We show that $\tau^{\mathrm{bw+di}}(Id)$ achieves greater reduction than $\sqsubseteq^{\mathrm{M}}$. The only direct or backward related states (except identical ones) are as follows:

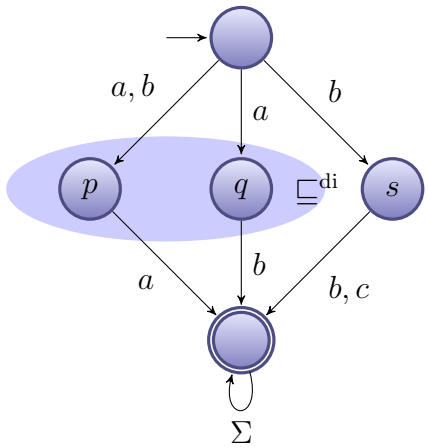
$$q, s \sqsubseteq^{\mathrm{di}} p \qquad \text{and} \qquad q \sqsubseteq^{\mathrm{bw}} s$$

Mediated preorder $\sqsubseteq^{\mathrm{M}}$ is just forward simulation in this example: The only new potential pair of states in mediated preorder not already in $\sqsubseteq^{\mathrm{di}}$ is (s, q) (since $s \sqsubseteq^{\mathrm{di}} s \sqsupseteq^{\mathrm{bw}} q$). But if $s \sqsubseteq^{\mathrm{M}} q$, then, since $q \sqsubseteq^{\mathrm{di}} p$, by the second condition defining $\sqsubseteq^{\mathrm{M}}$, one should have $s \sqsubseteq^{\mathrm{M}} p$ as well. But $(s, p) \notin \sqsubseteq^{\mathrm{di}} \circ \sqsupseteq^{\mathrm{bw}}$, therefore $s \not\sqsubseteq^{\mathrm{M}} q$. Thus, $\sqsubseteq^{\mathrm{M}} = \sqsubseteq^{\mathrm{di}}$, and mediated preorder does not help quotienting the automaton.

However, p and q are $\tau^{\mathrm{bw+di}}(Id)$-equivalent: $q \; \tau^{\mathrm{bw+di}}(Id) \; p$ since from p Duplicator can play actions a and b; $p \; \tau^{\mathrm{bw+di}}(Id) \; q$ since Duplicator can play action a from q, and she can jump from q to the $\sqsubseteq^{\mathrm{bw}}$-larger s if she needs to take action b.

Also mediated preorder and the forward proxy simulation $\tau^{\mathrm{di+bw}}(Id)$ are incomparable. This is more interesting, since they are both coarser than direct simulation. This is shown in the next example, which is dual to Example 4.7.1.

Example 4.7.2 - Forward proxy simulation quotients can outperform mediated quotients



Consider the automaton above. (It is the same automaton as in Example 4.6.4.) We show that $\tau^{\mathrm{di+bw}}(Id)$ can achieve a better reduction than $\sqsubseteq^{\mathrm{M}}$. Direct and backward simulation-related states are as follows (as usual, we ignore the identity relation):

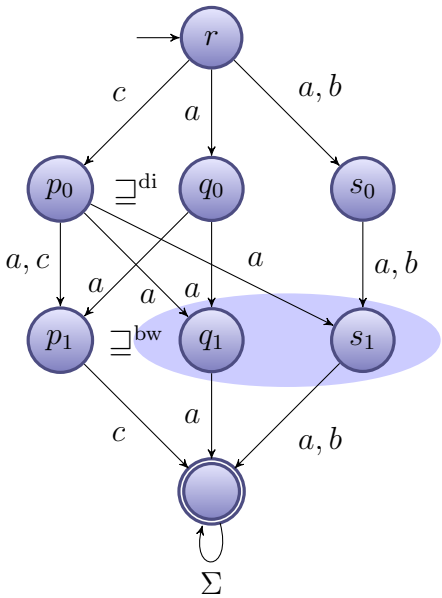
$$q \sqsubseteq^{\mathrm{di}} s \qquad \text{and} \qquad q, s \sqsubseteq^{\mathrm{bw}} p$$

Like in Example 4.7.1, also here mediated preorder coincides with forward simulation: Indeed, mediated preorder is a subset of $\sqsubseteq^{\mathrm{di}} \circ \sqsupseteq^{\mathrm{bw}}$, which is just $\sqsubseteq^{\mathrm{di}}$ in this example.

Yet, p and q are $\tau^{\mathrm{di+bw}}(Id)$-equivalent. p clearly simulates q, since Duplicator can take both backward actions a and b from p. Also, q simulates p: If Spoiler plays action a, then Duplicator can directly play the backward a-transition from q; otherwise, if Spoiler plays action b, then Duplicator can jump from state q to the $\sqsubseteq^{\mathrm{di}}$-larger state s, and play the backward b-transition from the latter state.

Finally, we show an example where mediated preorder gives a greater reduction than proxy simulations. This is slightly more involved; the reason is explained in Section 4.7.3.

Example 4.7.3 - Mediated quotients can outperform proxy simulation quotients



The only states related by ordinary forward and backward simulation are as follows (as usual, we omit the identity):

$$q_0 \sqsubseteq^{\mathrm{di}} p_0 \qquad \text{and} \qquad q_0 \sqsubseteq^{\mathrm{bw}} s_0$$
$$q_1 \sqsubseteq^{\mathrm{di}} s_1 \qquad\qquad q_1 \sqsubseteq^{\mathrm{bw}} p_1, s_1$$

First, we show that there are mediated preorder-equivalent states.

Mediated preorder q_1 and s_1 are mediated preorder equivalent:

- $q_1 \sqsubseteq^{\mathrm{M}} s_1$: just because of $q_1 \sqsubseteq^{\mathrm{di}} s_1$.
- $s_1 \sqsubseteq^{\mathrm{M}} q_1$: via mediator s_1, since $s_1 \sqsubseteq^{\mathrm{di}} s_1 \sqsupseteq^{\mathrm{bw}} q_1$. This is compatible with condition 2) for mediated preorder, since the only state $\sqsubseteq^{\mathrm{di}}$-larger than q_1 is s_1, and $s_1 \sqsubseteq^{\mathrm{M}} s_1$ trivially holds.

Then, we show that there are no equivalent states w.r.t. neither forward nor backward proxy simulations. This implies that mediated preorder can achieve better reduction than proxy simulations.

Forward proxy simulation Let $\preceq = \tau^{\mathrm{di+bw}}(Id)$. Clearly, no two states from different "rows" can be $\preceq$-comparable. For states in the same row, we proceed by excluding all possibilities.

- $q_0 \not\preceq p_0$: Duplicator cannot jump, since there is no state $\sqsubseteq^{\mathrm{di}}$-larger than p_0. Spoiler wins immediately by playing the backward a-transition from q_0, which is unavailable to Duplicator from p_0.

- $q_0 \not\preceq s_0$: Duplicator cannot jump, since there is no state $\sqsubseteq^{\mathrm{di}}$-larger than s_0. Spoiler can play the jumping transition $q_0 \sqsubseteq^{\mathrm{di}} p_0 \xleftarrow{c} r$, while action c is unavailable to Duplicator, who loses.

- $q_1 \not\preceq s_1$: Spoiler takes transition $q_1 \xleftarrow{a} q_0$. Duplicator cannot jump, since there is no state $\sqsubseteq^{\mathrm{di}}$-larger than s_1. If Duplicator takes transition $s_1 \xleftarrow{a} p_0$, then the game goes to configuration $\langle q_0, p_0 \rangle$, which is winning for Spoiler (first point above).

  Otherwise, if Duplicator takes transition $s_1 \xleftarrow{a} s_0$, then the game goes to configuration $\langle q_0, s_0 \rangle$, from which Spoiler wins as in the point above.

  In Section 4.7.3 we interpreted mediated preorder as a special kind of jumping simulation game. From the discussion there, it will be clear that, since $s_1 \sqsubseteq^{\mathrm{M}} q_1$, even if $q_1 \not\preceq s_1$, Duplicator cannot immediately lose in the first round of the latter game: We really need to allow Spoiler (and only Spoiler) to actively jump in the second round of the simulation game. That's why Example 4.7.3 has a higher depth w.r.t. the previous examples. Indeed, one can show that if players are allowed to jump only in the first round, then $\sqsubseteq^{\mathrm{M}}$ and (the transpose of) $\preceq$ would coincide.

- $p_0 \not\preceq s_0$ and $p_1 \not\preceq s_1$: Again, Duplicator cannot jump from neither s_0 nor s_1. From both configurations $\langle p_0, s_0 \rangle$ and $\langle p_1, s_1 \rangle$, Spoiler can play the backward c-transitions, while Duplicator can't.

- $q_1 \not\preceq p_1$: Spoiler can take the jumping b-transition $q_1 \sqsubseteq^{\mathrm{di}} s_1 \xleftarrow{b} s_0$, and Duplicator cannot do any b from p_1.

Backward proxy simulation Let $\preceq = \tau^{\mathrm{bw+di}}(Id)$. Again, no two states from different "rows" are $\preceq$-comparable. For states in the same row, we reason as follows.

- $p_1 \not\preceq s_1$: Spoiler can take a forward c-transition, and Duplicator can't.

- $q_1 \not\preceq s_1$: Spoiler can jump from q_1 to the $\sqsubseteq^{\mathrm{bw}}$-larger state p_1, and then wins as in the previous point.
- $q_1 \not\preceq p_1$: Spoiler takes a forward a-transition and Duplicator can't.
- $p_0 \not\preceq q_0$ and $p_0 \not\preceq s_0$: In both cases, Spoiler takes a forward c-transition from p_0, and Duplicator, even if $\sqsubseteq^{\mathrm{bw}}$-jumping, cannot simulate it.
- $q_0 \not\preceq s_0$: Spoiler takes transition $q_0 \xrightarrow{a} q_1$, and Duplicator is forced to take transition $s_0 \xrightarrow{a} s_1$. From configuration $\langle q_1, s_1 \rangle$, Spoiler wins as above.

4.7.3 Mediated preorder as a jumping simulation game

A deeper analysis shows that mediated preorder shares a structure which is very similar to forward proxy simulations. Let $\sqsupseteq^{\mathrm{M}}$ be the transpose of mediated preorder, and assume $q \sqsupseteq^{\mathrm{M}} s$. We interpret mediated preorder as a special kind of $\sqsubseteq^{\mathrm{di}}$-jumping backward simulation game. Let Spoiler jump to state $\hat{q}$ s.t. $q \sqsubseteq^{\mathrm{di}} \hat{q}$. By condition 2) of mediated preorder, $\hat{q} \sqsupseteq^{\mathrm{M}} s$. By condition 1), there exists a mediator $\hat{s}$ s.t. $s \sqsubseteq^{\mathrm{di}} \hat{s}$ and $\hat{q} \sqsubseteq^{\mathrm{bw}} \hat{s}$. So, let Duplicator reply by jumping to mediator $\hat{s}$. Therefore, if Spoiler selects a backward transition $\hat{q} \xleftarrow{a} q'$, then, by the definition of backward simulation, Duplicator can select matching transition $\hat{s} \xleftarrow{a} s'$ s.t. $q' \sqsubseteq^{\mathrm{bw}} s'$. This interaction is very similar to the one of $\tau^{\mathrm{di+bw}}(Id)$, which now is more convenient to write as $\tau^{\mathrm{bw}}(\sqsubseteq^{\mathrm{di}})$. We remark two important differences.

1. We summarize the first difference as "proxies are *dynamic*, while mediators are *static*": With this we mean that the mediator state $\hat{s}$ above depends only on $\hat{q}$, and not on the completed transition $\hat{q} \xleftarrow{a} q'$. Conversely, proxies in jumping games (like the one for $\tau^{\mathrm{bw}}(\sqsubseteq^{\mathrm{di}})$) generally depend on the full move by Spoiler.

   This gives more power to Duplicator, and it is the key reason as to why $\tau^{\mathrm{bw}}(\sqsubseteq^{\mathrm{di}})$ outperforms $\sqsubseteq^{\mathrm{M}}$ in Example 4.7.2: Indeed, when Spoiler moves from p, Duplicator jumps to a state depending on whether Spoiler plays an a or a b action; in the former case, Duplicator just stays in q, while in the latter case she jumps to s.

2. The second difference concerns the situation when jumps are allowed during the simulation game. In jumping simulation games, we allow both players to jump at any round. For example, in the game for $\tau^{\mathrm{bw}}(\sqsubseteq^{\mathrm{di}})$, both players can always jump to $\sqsubseteq^{\mathrm{di}}$-larger states before taking a backward transition.

In mediated preorder games, *players can jump only in the first round*, and then take a backward transition. Indeed, condition 2) of mediated preorder allows Spoiler to jump during the first round, while condition 1) allows the same for Duplicator. But, after the jump, in mediated preorder one requires $q' \sqsubseteq^{\mathrm{bw}} s'$, which is to say $q' \; \tau^{\mathrm{bw}}(Id) \; s'$. That is, from the second round on, we fall back to an ordinary backward simulation game with no jumps.

This discrepancy between mediated simulation games and proxy simulation games is exploited in Example 4.7.3 to show that mediated preorder can outperform proxy simulations. In particular, to show a difference between mediated preorder and forward proxy simulations, it is necessary to have a jumping game with at least two non-trivial rounds—i.e., such that diagonal configurations $\langle x, x\rangle$ are not reached before two rounds. This explains why Example 4.7.3 is more complex than the previous ones.

4.8 Computing jumping simulations

We present simple and efficient PTIME algorithms for computing jumping simulation transformers. In previous work [44], computing simulations has been reduced to solving 2 player games with a parity winning condition [58]. We take an equivalent approach based on solving fixpoint equations over suitable transition systems.

4.8.1 Jumping predecessor operators

Let $\sqsubseteq_0$ and $\sqsubseteq_1$ be two preorders. Configurations are of the form $\langle q, s\rangle$, with $q, s \in Q$, and $C = Q \times Q$ is the set of all configurations. We take the point of view of Spoiler, and we define a controlled predecessor operator $\mathsf{CPre}(\cdot)$ s.t., for sets of configurations X, Y and Z, $\mathsf{CPre}(X, Y, Z)$ represents all configurations from which Spoiler has a move s.t., for any Duplicator's reply, either

- Spoiler has taken an accepting transition, Duplicator hasn't, and the game is forced into X; or
- Duplicator cannot take an accepting transition, and the game is forced into Y; or
- The game is forced into Z.

Formally, we have the following definition.

$$\begin{aligned}\mathsf{CPre}(X,Y,Z) = \bigcup_{a\in\Sigma}\{\langle q,s\rangle \mid \exists(q\sqsubseteq_0 \xrightarrow{a} q') \cdot \forall(s\sqsubseteq_1 \xrightarrow{a} s') \cdot \\ \textit{either} \quad (q\sqsubseteq_0 \xrightarrow{a}_F q') \wedge \neg(s\sqsubseteq_1 \xrightarrow{a}_F s') \wedge \langle q',s'\rangle \in X \\ \textit{or} \quad \neg(s\sqsubseteq_1 \xrightarrow{a}_F s') \wedge \langle q',s'\rangle \in Y \\ \textit{or} \quad \langle q',s'\rangle \in Z\}\end{aligned}$$

Based on the above definition for $\mathsf{CPre}(\cdot)$, we obtain two more predecessor operators by specialization. For sets of configurations X and Y,

- $\mathsf{CPre}^1(X,Y)$ is the set of configurations from which Spoiler can ensure to either take an accepting transition, while preventing Duplicator from doing so, and to go in X; or, to go in Y.
- $\mathsf{CPre}^0(X,Y)$ is the set of configurations from which Spoiler can ensure that either Duplicator does not take an accepting transition and go in X, or to go in Y.

Formally,

$$\mathsf{CPre}^1(X,Y) = \mathsf{CPre}(X,Y,Y)$$
$$\mathsf{CPre}^0(X,Y) = \mathsf{CPre}(X,X,Y)$$

Having these predecessor operators in hand, we are ready to write down suitable fixpoint equations representing the winning region W^x of Spoiler, for $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}, \mathrm{bw}\}$. The jumping simulation preorder can be obtained by complementation,

$$\tau^x(\sqsubseteq_0, \sqsubseteq_1) = C \setminus W^x$$

4.8.2 Direct and backward simulations

Spoiler wins the direct simulation game iff she can drive the game to a configuration where she can take an accepting transition and Duplicator can't:

$$W^{\mathrm{di}} = \mu X \cdot \mathsf{CPre}^1(C,X)$$

For backward simulation, Spoiler wins also if the game reaches a configuration from which she can jump to an initial state, but Duplicator can't.

$$W^{\mathrm{bw}} = \mu X \cdot T \cup \mathsf{CPre}^{1'}(C,X)$$
$$\text{where } T = \{\langle q,s\rangle \mid q \sqsubseteq_0 I \wedge s \not\sqsubseteq_1 I\}$$

The predecessor operator $\mathsf{CPre}^{1'}(\cdot)$ is the same as $\mathsf{CPre}^1(\cdot)$, except that transitions are taken backwards, in accord with the definition of backward simulation.

4.8.3 Fair simulation

Spoiler wins the fair simulation game if she can eventually force the game into a position from which Duplicator cannot take accepting transitions anymore, while Spoiler can still take accepting transitions infinitely often.

$$W^{\mathrm{f}} = \mu X \cdot \nu Y \cdot \mu Z \cdot \mathsf{CPre}(Y, Z, X)$$

4.8.4 Delayed simulation

The fixpoint formula for jumping delayed simulation is an adaption to the jumping case of an analogous formula for ordinary delayed simulation [73]. Spoiler wins the delayed game iff she can eventually take an accepting transition (and Duplicator can't), and go to a configuration from which Duplicator cannot take accepting transitions anymore.

$$W^{\mathrm{de}} = \mu X \cdot \mathsf{CPre}^1(\nu Y \cdot \mathsf{CPre}^0(Y, X), X)$$

Chapter 5

Multipebble simulations for ABAs

Contents

5.1 Applications of alternating automata **124**

5.1.1 Automata complementation 124

5.1.2 Linear temporal logic 125

5.1.3 Alternating automata with existential and universal states . 126

5.2 Simulations for alternating Büchi automata **127**

5.3 Multipebble simulations for alternating Büchi automata **130**

5.4 Basic properties of multipebble simulations **135**

5.4.1 Multipaths 135

5.4.2 Hierarchies of multipebble simulations 136

5.4.3 Containment and language inclusion 138

5.5 Uniform simulation games **145**

5.5.1 Translation to uniform automata 146

5.5.2 Correctness of the translation 152

5.6 Transitivity **158**

5.6.1 Proof overview 159

5.6.2 Direct simulation 162

5.6.3 Delayed and fair simulation 168

5.6.4 Comparison with previous work [29] 174

5.7 Multipebble simulations and subset-like constructions **177**

5.7.1 Characterizing $(n, 1)$-fair simulation 177

5.7.2 Characterizing $(1, n)$-fair simulation 184
5.7.3 Characterizing (n, n)-fair simulation 191
5.8 Quotienting **193**
5.8.1 Quotienting difficulties 195
5.8.2 Minimax quotients 196
5.8.3 Requirements for delayed simulation 204
5.8.4 Semielective quotients 208
5.9 Computing multipebble simulations **219**
5.9.1 Direct simulation 220
5.9.2 Fair simulation 220
5.9.3 Delayed simulation 222
5.10 Complexity of multipebble simulations **225**
5.10.1 PSPACE-membership 227
5.10.2 Domino-tiling games 232
5.10.3 PSPACE-hardness of $(1, n)$-simulations 233
5.10.4 EXPTIME-hardness of $(1, n)$-simulations on ABAs 236
5.10.5 EXPTIME-hardness of $(1, \sqrt{n})$-simulations on NBAs . . . 240

Overview

We generalize multipebble simulation relations from nondeterministic to alternating Büchi automata (ABA). As in the previous chapters, our aim is to obtain relations which can be used for quotienting and for showing language inclusion. While in the fixed-word and proxy simulations of Chapters 3 and 4 we have addressed quotienting and language inclusion for nondeterministic automata, in this chapter we develop techniques which can be applied to alternating automata.

Alternating automata are a model of computation where existential and universal choice coexist. The standard model of alternation considers transitions as positive boolean combinations of successors. In this chapter, instead, we adopt the model of [55], where the states are partitioned into existential and universal states. For a comparison with the standard model and applications, see Section 5.1.

Multipebble simulations are designed to given more power to Duplicator in the simulation game than in ordinary 1-pebble simulation. Having multiple pebbles allows Duplicator to "hedge her bets" to delay decisions, thus yielding a simulation relation which is coarser than ordinary 1-pebble simulation. We consider multiple pebbles on both sides, i.e., (k_0, k_1)-simulations, with k_0 pebbles on the left and k_1 pebbles on the right. Intuitively, Duplicator controls pebbles on universal states on the left and existential states on the right (and dually for Spoiler). This generalizes previous work on ordinary $(1, 1)$-simulation for nondeterministic Büchi automata (NBA) [44] and ABA [55], and $(1, k)$-simulation for NBA [42].

We consider direct, delayed and fair simulations. In each case, (k_0, k_1)-simulations form a complete lattice, where $(1, 1)$- and (n, n)-simulations are the bottom and top element (if the automaton has n states), respectively, and the order is strict in general. We show that multipebble simulations can be used to prove language inclusion between ABA, and, for the direct and delayed variant, they can also be used for quotienting ABA. More generally, multipebble simulations fulfill the simulations desiderata of Section 2.5:

(Da) Multipebble simulation implies language inclusion.

(Db) (k_0, k_1)-direct and $(1, n)$-delayed simulations are GFQ.

(Dc) For any fixed $k_0, k_1 > 0$, (k_0, k_1)-simulation is computable in polynomial time.

(Dd) $(1, 1)$, $(1, n)$, $(n, 1)$ and (n, n)-simulations are transitive, where n is the number of states in the automaton.

The GFQ property generally fails for naïve quotients, and other notions of quotients have to be introduced. Additionally, multipebble simulations yield new insights into the Miyano-Hayashi construction for alternation removal [92].

Summary of results and structure of the chapter

We explain in greater detail the results of this chapter, together with a summary of its structure. See also Table 5.1. The material presented is a major extension of [29]. Moreover, many results are new and have not been published before; they are marked with a star in the list below.

- In Section 5.2, we review previous work on simulation preorders for alternating Büchi automata.

Language incl. (Section 5.4.3)	Pebbles $(1,1)$	fixed (k_0,k_1)	$(1,n)$	$(n,1)$	(n,n)
all sims.	✓	✓	✓	✓	✓
Transitivity (Sec. 5.6)					
all sims.	✓	×	✓	✓	✓
Quotienting (Sec. 5.8)					
Direct sim.	✓	✓	✓	✓	✓
Delayed sim.	✓	×	✓	×	×
Fair sim.	×	×	×	×	×
Complexity (Sec. 5.10)					
on NBAs	PTIME	PTIME	PSPACE	—	—
on ABAs	PTIME	PTIME	EXPTIME	EXPTIME	PSPACE

Table 5.1: Multipebble simulations summary

- In Section 5.3, we introduce our generalization to multiple pebbles.
- In Section 5.4, we discuss fundamental properties of multipebble simulations. In particular, in Section 5.4.2 we show that (k_0, k_1)-simulations can be naturally arranged into a bidimensional hierarchy as follows (where the inclusions, in general, are strict):

$$\begin{array}{ccccc}
\vdots & & \vdots & & \iddots \\
\sqsubseteq_{(2,1)} & \subseteq & \sqsubseteq_{(2,2)} & \subseteq & \cdots \\
\cup\!\mid & & \cup\!\mid & & \\
\sqsubseteq_{(1,1)} & \subseteq & \sqsubseteq_{(1,2)} & \subseteq & \cdots
\end{array}$$

- *Language inclusion (Da):* In Section 5.4.3, we show that all simulations we consider imply language inclusion. That is, for any $k_0, k_1 > 0$, (k_0, k_1)-direct, delayed and fair simulation imply language inclusion.
- *Uniform simulation games*:* In Section 5.5, we introduce a restricted class of multipebble simulation games, called *uniform games*, where we make further assumptions on the structure of plays and on the behavior of players. We show that uniform games are equivalent to ordinary, unrestricted games, in the sense

that, for any ordinary game, there exists a uniform game where the winner is preserved. Many subsequent results are proved, without loss of generality, in the restricted case of uniform games.

- *Transitivity (Dd):* In general, (k_0, k_1)-simulations are not transitive. However, limit simulation *are* transitive, i.e., when Duplicator uses either just one or n pebbles on either side (where n is the number of the states in the automaton). Transitivity for $(1, 1)$-simulations has been established in [55]. In Section 5.6, we extend this result to $(1, n)$, $(n, 1)$ and (n, n)-direct, delayed and fair simulations. The result for (n, n)-transitivity has not been published before.

- *Subset constructions:* Maximal fair multipebble simulations give new insights into classical subset constructions for alternating automata. In Section 5.7, we show that $(n, 1)$-fair simulation characterizes the Miyano-Hayashi alternation-removal construction, while $(1, n)$-simulation characterizes a kind of dual construction. Finally, (n, n)-fair simulation is shown to characterize both. The characterization for $(1, n)$ and (n, n)-simulation has not been published before.

- *Quotienting (Db):* Quotienting in alternating automata is non-trivial. In Section 5.8, we generalize minimax and semielective quotients from [55], and we show that they are sound for direct and delayed multipebble simulation, respectively.

  However, while unrestricted (k_0, k_1)-direct simulation is GFQ (w.r.t. minimax quotients), we show that only $(1, n)$-delayed simulation is GFQ (w.r.t. semielective quotients). Indeed, for the (k_0, k_1), $(n, 1)$ and (n, n) cases of delayed simulation, we show that there is no reasonable sound definition of quotient.

- *Algorithms and complexity (Dc)*:* (k_0, k_1)-direct, delayed and fair simulation is computable in polynomial time for fixed $k_0, k_1 > 0$. However, for large number of pebbles, the problem becomes PSPACE-hard. In certain cases, it is even EXPTIME-hard. See Section 5.9 for practical algorithms, and Section 5.10 for complexity results. No computational complexity result was previously known, even for $(1, k)$-simulations on NBAs (i.e., in the setting of [42]).

5.1 Applications of alternating automata

Alternating automata are a model of computation with both universal and existential choice. The standard model of alternation considers as transitions positive boolean combinations of successors. I.e., for a state p and input symbol a, a valid transition is, e.g., $\delta(p, a) = (q \wedge r) \vee s$; this means that the rest of the input has to be accepted by either both q and r, or by s alone. Alternating automata find their applications in efficient translations of modal and temporal logics (such as LTL [115]), and as intermediate representations in automata complementation algorithms.

Note that alternating automata are easily seen to be closed under union and intersection; for example, given two states p and q, their intersection is given by a new state r s.t., on input a, $\delta(r, a) = \delta(p, a) \wedge \delta(q, a)$. Complementation is slightly more involved, since we also need to dualize the acceptance condition, as shown next.

5.1.1 Automata complementation

Complementation of Büchi automata is a difficult task, both theoretically [117, 106], and practically [113]. Its applications in formal verification include checking the correctness of LTL translation algorithms [63], and the *Safraless* approach to synthesis [79].

For a given automaton $\mathcal{B}$ with n states, its complement $\mathcal{B}^c$ can have as many as $\Omega(2^{n \log n})$ states. Therefore, even a small reduction in size in $\mathcal{B}$ can result in an exponentially smaller complement automaton $\mathcal{B}^c$. Thus, reducing an automaton *before* complementation is crucial to any complementation procedure. Moreover, complementation algorithms that manipulate intermediate automata can benefit from quotienting these intermediate representations as well. One such example is given by the so-called *Rank-based complementation* [81], which utilizes a small intermediate alternating automaton. Schematically, it works as follows: Given a nondeterministic Büchi automaton $\mathcal{B}$ with n states,

- One builds an alternating co-Büchi automaton $\mathcal{C}$ with n states (obtained by just dualizing the transition relation and the acceptance condition) accepting the complement language: $\mathcal{L}(\mathcal{C}) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{B})$.
- From $\mathcal{C}$, one builds an equivalent *weak alternating automaton* $\mathcal{W}$ with $O(n^2)$ states (following the construction in [81]) s.t. $\mathcal{L}(\mathcal{W}) = \mathcal{L}(\mathcal{C})$.
- From $\mathcal{W}$, one applies the standard Miyano-Hayashi de-universalization construction in order to remove universal non-determinism [92], and obtains an equivalent

nondeterministic Büchi automaton $\mathcal{B}^c$ with $O(2^{n \log n})$ states s.t. $\mathcal{L}(\mathcal{B}^c) = \mathcal{L}(\mathcal{W})$.

Overall, one has $\mathcal{L}(\mathcal{B}^c) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{B})$, that is, $\mathcal{B}^c$ is the complement automaton one is looking for.

Simulation-based optimizations can be applied at every stage of the algorithm. In particular, the intermediate weak alternating automaton $\mathcal{W}$ can be quotiented with simulation-based methods before it is fed into the exponential Miyano-Hayashi construction. Moreover, the Miyano-Hayashi construction itself can be optimized with on-the-fly simulation-based pruning of transitions, in the spirit of [54].

5.1.2 Linear temporal logic

The capability of having arbitrary positive boolean formulas in the transitions makes alternating automata a natural model when translating from modal and temporal logics. Here, we recall the translation for the temporal logic LTL [115]. Formulas in linear temporal logic (LTL) are build from a set of (positive and negative) atomic symbols Σ by combining boolean and temporal connectives, as per the following abstract syntax:

$$\phi ::= a \mid \neg a \mid \phi \wedge \phi \mid \phi \vee \phi \mid \mathbf{X}\phi \mid \phi \mathbf{U} \phi \mid \phi \mathbf{R} \phi, \qquad \text{where } a \in \Sigma$$

LTL formulas are interpreted over ω-words: For a LTL formula ϕ, a word $w \in \Sigma^\omega$, and a time point $i < \omega$, we write $w, i \models \phi$ if the suffix of w starting at position i satisfies ϕ. The satisfaction relation is defined by structural induction on formulas:

$$\begin{array}{rcl}
w, i \models a & \text{iff} & w(i) = a \\
w, i \models \neg a & \text{iff} & w(i) \neq a \\
w, i \models \phi_0 \wedge \phi_1 & \text{iff} & w, i \models \phi_0 \text{ and } w, i \models \phi_1 \\
w, i \models \phi_0 \vee \phi_1 & \text{iff} & w, i \models \phi_0 \text{ or } w, i \models \phi_1 \\
w, i \models \mathbf{X}\phi & \text{iff} & w, i+1 \models \phi \\
w, i \models \phi_0 \mathbf{U} \phi_1 & \text{iff} & \text{there exists } j \geq i \text{ s.t. } w, j \models \phi_1 \text{ and,} \\
 & & \text{for any } i \leq k < j,\ w, k \models \phi_0 \\
w, i \models \phi_0 \mathbf{R} \phi_1 & \text{iff} & \text{for every } j \geq i, \text{ either } w, j \models \phi_1, \\
 & & \text{or there exists } i \leq k < j \text{ s.t. } w, k \models \phi_0
\end{array}$$

Intuitively, atomic symbols $a \in \Sigma$ are used to check the current input symbol in the word; boolean symbols are as usual; the formula $\mathbf{X}\phi$ holds now if in the next instant ϕ

holds; the formula $\phi_0 \mathbf{U} \phi_1$ holds if eventually in the future ϕ_1 holds, and, at any time strictly before then, ϕ_0 holds; finally, $\phi_0 \mathbf{R} \phi_1$ is the dual of $\phi_0 \mathbf{U} \phi_1$.

Given an LTL formula ϕ_0, the set of words w s.t. $w, 0 \models \phi$ holds is ω-regular, and it can be succinctly recognized by an alternating automaton of size linear in the number of subformulas of ϕ_0. States of the automaton are subformulas of ϕ_0. The initial state is ϕ_0 itself. Transitions are defined by structural induction on formulas [115]:

- if $\phi = a$, then $\delta(a, a) = \texttt{true}$ and $\delta(a, b) = \texttt{false}$, for $b \neq a$;
- if $\phi = \neg a$, then $\delta(\neg a, a) = \texttt{false}$ and $\delta(\neg a, b) = \texttt{true}$, for $b \neq a$;
- if $\phi = \psi_0 \wedge \psi_1$, then $\delta(\phi, a) = \delta(\psi_0, a) \wedge \delta(\psi_1, a)$;
- if $\phi = \psi_0 \vee \psi_1$, then $\delta(\phi, a) = \delta(\psi_0, a) \vee \delta(\psi_1, a)$;
- if $\phi = \mathbf{X}\psi$, then $\delta(\phi, a) = \psi$;
- if $\phi = \psi_0 \mathbf{U} \psi_1$, then $\delta(\phi, a) = \delta(\psi_1, a) \vee (\delta(\psi_0, a) \wedge \phi)$;
- if $\phi = \psi_0 \mathbf{R} \psi_1$, then $\delta(\phi, a) = \delta(\psi_1, a) \wedge (\delta(\psi_0, a) \vee \phi)$.

Accepting states are those of the form $\psi_0 \mathbf{R} \psi_1$; the additional states `true` and `false` recognize Σ^ω and $\emptyset$, respectively. The transition for $\mathbf{U}$ is obtained by applying the usual fixpoint characterization

$$\psi_0 \mathbf{U} \psi_1 = \psi_1 \vee (\psi_0 \wedge \mathbf{X}(\psi_0 \mathbf{U} \psi_1)) \ ,$$

and similarly for $\mathbf{R}$.

If $\mathcal{A}_{\phi_0}$ is the automaton obtained by the procedure above from ϕ_0, then one can prove that $\mathcal{A}_{\phi_0}$ recognizes exactly the words w which are models for ϕ_0; i.e., $\mathcal{L}(\mathcal{A}_{\phi_0}) = \{w \in \Sigma^\omega \mid w, 0 \models \phi_0\}$ [115]. Therefore, ϕ_0 is satisfiable iff $\mathcal{A}_{\phi_0}$ has non-empty language, which shows a linear-time reduction from LTL satisfiability to non-emptiness of alternating automata. Again, simulation-based optimizations can be applied to $\mathcal{A}_{\phi_0}$ before it is fed into more expensive constructions.

5.1.3 Alternating automata with existential and universal states

Both the complementation and translation procedures from LTL make use of alternating automata with transitions containing arbitrary boolean combinations. In the simplified model with existential and universal states of [55] (which is the one that we consider

in this chapter), one only considers a normal form for transitions, where there is only a purely disjunctive or a purely conjunctive combination at every state. This more restrictive model of alternation is no longer suited for translating from temporal logics, or for complementation.

One workaround is to introduce epsilon transitions, and to generalize simulations to this new setting. This is done, e.g., in [54], where it is proposed a translation from LTL to alternating automata with epsilon transitions, along with a notion of simulation for the new model. The intuition is that, by using epsilon transitions and additional intermediate states, one can simulate arbitrary boolean combinations with just existential and universal states. The same technique can also be applied in the complementation algorithm.

In this chapter, we do not consider models or simulations with epsilon transitions. This this makes our quotienting techniques not directly applicable to alternating automata obtained from LTL or from complementation. However, this chapter can serve as a roadmap for the design of more complex GFQ multipebble simulations for quotienting automata with a richer model of alternation, in the same way as [55] provides the theoretical foundations for the extensions of [54] in the 1-pebble case. Moreover, considering a simplified alternation model allows us to isolate the key technical difficulties in quotienting alternating automata, which would would be otherwise obscured by a more complex model. Thus, the choice of a restrictive model of alternation allows us to expose these issues in their simplest form. Any generalization of multipebble simulations to alternating automata with either boolean combinations or epsilon transitions will have to consider and address the technicalities that we solve here.

5.2 Simulations for alternating Büchi automata

In this section, we review previous work on simulations for ABAs. All the definitions and results presented here are from [55].

Extending simulations to alternating models In ordinary single-pebble simulation games for nondeterministic models, Spoiler always moves the left pebble and Duplicator the right one. When generalizing simulations to alternating models, in order for simulation to be compatible with language inclusion and quotienting (i.e., desiderata (Da) and (Db)), the two players need to "swap side" when pebbles are on universal states; more precisely, when a pebble is on a universal state, it is controlled by Spoiler

if it is on the right and by Duplicator if it is on the left.

Formally, given a simulation game played on an alternating automaton $\mathcal{Q} = (Q, \Sigma, q_I, \Delta, E, U, F)$, if the current configuration of the simulation game is $\langle q, s \rangle$, then, in next round,

1. Spoiler selects an input symbol $a \in \Sigma$, and
   - If $q \in E$, then Spoiler selects a transition $q \xrightarrow{a} q'$.
   - If $s \in U$, then Spoiler selects a transition $s \xrightarrow{a} s'$.
2. Duplicator replies:
   - If $q \in U$, then Duplicator selects a transition $q \xrightarrow{a} q'$.
   - If $s \in E$, then Duplicator selects a transition $s \xrightarrow{a} s'$.

This is well-defined since a state is either existential or universal, but not both; the next configuration is $\langle q', s' \rangle$. Notice that,

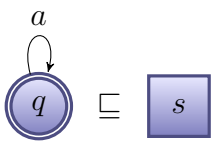
- If both pebbles are existential, then we have the same behavior as in usual simulation games on nondeterministic models.
- If both pebbles are universal, the usual behavior is reversed, with Spoiler moving on the right and Duplicator replying on the left.
- If the left pebble is existential and the right pebble universal, then only Spoiler plays in this round.
- Symmetrically, if the left pebble is universal and the right pebble existential, then only Duplicator plays—except for the choice of the next input symbol which is always up to Spoiler.

Premature end condition One needs to address what happens if a pebble cannot be moved because there exists no a-successor. We say that a pebble on state q is *stuck* if q has no a-successor (where a is clear from the context). In nondeterministic models, if the left pebble is stuck, then Duplicator wins, and, symmetrically, if the right pebble is stuck, then Spoiler wins. Since the left and right pebbles are also those controlled by Spoiler and Duplicator, respectively, one might be tempted to generalize this principle by saying:

If Spoiler's pebble is stuck, then Duplicator wins; and vice versa.

Indeed, this is how the winning condition for alternating simulation is defined in [55] when the game ends early.

However, this turns out to be incorrect. For example, consider the following situation:



With the rule above, Duplicator wins since the pebble on s is stuck (there is no a-successor), however $a^\omega = \mathcal{L}(q) \not\subseteq \mathcal{L}(s) = \emptyset$, and simulation preorder does not imply language inclusion (Da). Therefore, if we let Duplicator win when Spoiler is stuck on a universal right pebble, then the simulation is too coarse; on the other side, letting Spoiler win when Duplicator is stuck on a universal left pebble is too restrictive.

The corrected rule only regards the side in which pebbles are stuck, and not the player controlling them:

If the left pebble is stuck, then Duplicator wins; and vice versa.

Remark 5.2.1. Of course, one can assume w.l.o.g. that automata are total, so that simulation games never end prematurely. Nonetheless, we prefer to give a definition which works in the general case, and then assume totality only when proving its properties (as [55] does).

Simulation desiderata If the game does not end prematurely, two infinite paths $\pi_0 = q_0 \xrightarrow{a_0} \cdots$ and $\pi_1 = s_0 \xrightarrow{a_0} \cdots$ are built. The winner depends on the type of simulation one is considering. The notions of direct, delayed and fair simulation for nondeterministic models [44] immediately generalize to alternating models. We recall them once again:

$$
\begin{aligned}
x = \mathrm{di}: &\quad \forall (i \geq 0) \cdot q_i \in F \implies s_i \in F \\
x = \mathrm{de}: &\quad \forall (i \geq 0) \cdot q_i \in F \implies \exists (j \geq i) \cdot s_j \in F \\
x = \mathrm{f}: &\quad \pi_0 \text{ fair} \implies \pi_1 \text{ fair}
\end{aligned}
$$

As usual, one writes $q \sqsubseteq^x s$ iff Duplicator wins the x-simulation game starting from configuration $\langle q, s \rangle$.

Theorem 5.2.2 ([55]). *Simulations for ABA satisfy the simulation desiderata (Da)-(Dd).*

Among the simulation desiderata, quotienting (Db) is the most delicate, and represents the main issue when dealing with alternating models. The strength of $\sqsubseteq^x$ is that it allows to relate together existential and universal states. While being as general as possible allows for more reduction under quotienting, it also raises considerable issues when *mixed* quotient classes with both existential and universal states are formed. We present quotienting for alternating simulations in Section 5.8.

We are now ready to introduce our generalization of simulation to multiple pebbles.

5.3 Multipebble simulations for alternating Büchi automata

In this section, we define multipebble simulations for alternating models, thus generalizing both the alternating simulations of [55] (cf. Section 5.2) to multiple pebbles, and the multipebble simulations of [42] (cf. Section 3.2) from nondeterministic to alternating Büchi automata.

Intuition We have seen that, in ordinary 1-pebble simulations for alternating models, players can sometimes control both left and right pebbles, depending on the existential/universal type of the pebble. (Recall that Duplicator controls right existential and left universal pebbles; symmetrically, Spoiler controls left existential and right universal pebbles.) In a multipebble simulation game, we give more power to Duplicator in order to get a coarser simulation relation. "More power" takes the form of allowing Duplicator to have multiple pebbles, both on the left and on the right. When up to $k_0 > 0$ pebbles are used on the left and $k_1 > 0$ on the right, we call it a (k_0, k_1)-simulation game; ordinary simulation corresponds to $(1, 1)$-simulation, and the multipebble simulations for NBAs of [42] correspond to $(1, k)$-simulation.

Having more pebbles on both sides allows Duplicator to "hedge her bets" by splitting pebbles to several successors, in order to delay committing to particular decisions. To ensure that more pebbles are never harmful, we allow pebbles to be "taken away" from their current position; in particular, not all pebbles need to be used at any time. Therefore, $k + 1$ pebbles are always at least as good as k pebbles (on either side).

We call a pebble existential if it is on an existential state, and universal otherwise. Pebbles are moved by conservatively extending the rule for 1-pebble simulations: Duplicator controls right existential and left universal pebbles, while Spoiler controls

left existential and right universal pebbles. However, the two players move pebbles in a rather different way. While Duplicator is allowed to split pebbles to several successors and to take them away, Spoiler is forced to move each pebble independently to exactly one successor; in other words, Spoiler cannot take pebbles away, or split them.

We have not yet specified in which order pebbles should be moved. Since our aim is to get a simulation relation which is as large as possible (but satisfying the simulation desiderata), we make Spoiler move all pebbles under her control first, and then Duplicator will follow by moving the remaining pebbles—perhaps splitting them and taking away unwanted ones (possibly including Spoiler's). Thus, a generic round of the (k_0, k_1)-simulation game is informally played as follows:

- Spoiler selects an input symbol $a \in \Sigma$, and, for every existential-left and universal-right pebble, she selects an a-successor.
- Duplicator selects a (possibly empty) set of successors for every universal-left and existential-right pebble. Then, amongst all successor pebbles (including Spoiler's), she selects at most k_0 right and at most k_1 left pebbles.

The winning condition for games that stop prematurely is as follows:

If any left pebble gets stuck, then Duplicator wins;
otherwise, if all right pebbles are stuck, then Spoiler wins.

In any other case, two multipaths are built, i.e., paths of sets of states, and the winner depends on a predicate on such multipaths.

The design of the winning condition for multipebble simulations is delicate; it usually takes the form of an implication (for two properties of multipaths P_0 and P_1):

If the left multipath satisfies P_0, then the right multipath satisfies P_1.

For example, in ordinary 1-pebble fair simulation, both predicates P_0 and P_1 state that the path is fair. A necessary requirement is that simulation implies containment (Da). Therefore, the winning condition should be s.t., if an automaton $\mathcal{B}$ simulates an automaton $\mathcal{A}$, and a word $w \in \Sigma^\omega$ is accepted by $\mathcal{A}$, then Spoiler should be able to play s.t. a winning Duplicator is a witness for w to be accepted by $\mathcal{B}$. In general, the left condition P_0 should be a *necessary* condition for w to be accepted by $\mathcal{A}$, while the right condition P_1 should be a *sufficient* condition for w to be accepted by $\mathcal{B}$. For multipebble simulations, we take P_0 and P_1 to be the notions of being "universally" and "existentially fair", respectively (see later, Definitions 5.3.3 and 5.3.2).

Definition We now formally define the multipebble simulation game. Let $\mathcal{Q}$ be an alternating automaton, $\mathbf{q}_0 \in 2^{Q,k_0}$ a k_0-set and $\mathbf{s}_0 \in 2^{Q,k_1}$ a k_1-set. For any set $\mathbf{q} \subseteq Q$, let $\mathbf{q}^E := \mathbf{q} \cap E$ be the set of existential states in $\mathbf{q}$, and similarly $\mathbf{q}^U := \mathbf{q} \cap U$. We define the basic (k_0, k_1)-simulation game $\mathbb{G}^{(\mathrm{k}_0,\mathrm{k}_1)}(\mathbf{q}_0, \mathbf{s}_0)$ as follows. (Formally, Spoiler is Player 0 and Duplicator is Player 1.) Spoiler's positions are in $P_{\text{Sp}} := 2^{Q,k_0} \times 2^{Q,k_1}$, and Duplicator's in $P_{\text{Dup}} := 2^{Q,k_0} \times 2^{Q,k_1} \times \Sigma \times 2^{Q,k_0} \times 2^{Q,k_1}$. In the initial configuration $\langle \mathbf{q}_0, \mathbf{s}_0 \rangle$, left pebbles are on $\mathbf{q}_0$ and right pebbles on $\mathbf{s}_0$. If the current configuration at round i is $\langle \mathbf{q}_i, \mathbf{s}_i \rangle$, then the next configuration $\langle \mathbf{q}_{i+1}, \mathbf{s}_{i+1} \rangle$ is determined in two steps as follows:

- First, Spoiler chooses a transition $\langle (\mathbf{q}_i, \mathbf{s}_i), (\mathbf{q}_i, \mathbf{s}_i, a_i, \mathbf{q}', \mathbf{s}') \rangle \in \Gamma^{\text{Sp}} \subseteq P_{\text{Sp}} \times P_{\text{Dup}}$ iff a_i is chosen as the next input symbol, and

  - $\mathbf{q}'$ is obtained from $\mathbf{q}_i^E$ by choosing exactly one successor for *each pebble* in $\mathbf{q}_i^E$. Formally[1], $\mathbf{q}' \in \bigotimes\{\Delta(q, a) \mid q \in \mathbf{q}_i^E\}$.

    If two states have exactly the same set of successors, we can assume that Spoiler chooses the same successor for both.

  - Symmetrically, $\mathbf{s}'$ is obtained from $\mathbf{s}_i^U$ by choosing one successor for each pebble in $\mathbf{s}_i^U$, i.e., $\mathbf{s}' \in \bigotimes\{\Delta(s, a) \mid s \in \mathbf{s}_i^U\}$.

  For notational convenience, we henceforth write Spoiler's transitions just as $(\mathbf{q}_i, \mathbf{s}_i, a_i, \mathbf{q}', \mathbf{s}') \in \Gamma^{\text{Sp}}$.

- Then, Duplicator chooses a transition $\langle (\mathbf{q}_i, \mathbf{s}_i, a_i, \mathbf{q}', \mathbf{s}'), (\mathbf{q}_{i+1}, \mathbf{s}_{i+1}) \rangle \in \Gamma^{\text{Dup}} \subseteq P_{\text{Dup}} \times P_{\text{Sp}}$ iff

  - $\mathbf{q}_{i+1}$ is a non-empty k_0-subset of $\mathbf{q}' \cup \Delta(\mathbf{q}_i^U, a)$, and
  - $\mathbf{s}_{i+1}$ is a non-empty k_1-subset of $\mathbf{s}' \cup \Delta(\mathbf{s}_i^E, a)$.

Thus, $\mathbf{q}_i \xRightarrow{a_i} \mathbf{q}_{i+1}$ and $\mathbf{s}_i \xRightarrow{a_i} \mathbf{s}_{i+1}$, which we simply write as $\langle \mathbf{q}_i, \mathbf{s}_i \rangle \xRightarrow{a_i} \langle \mathbf{q}_{i+1}, \mathbf{s}_{i+1} \rangle$.

Remark 5.3.1. Notice that Duplicator is always allowed to "take pebbles away", and to "hedge her bets" by splitting pebbles into different successors.

The game ends prematurely when either 1) some left pebble is stuck, and Duplicator wins in this case, or 2) no left pebble is stuck and all right pebbles are stuck, and Spoiler

[1] The unordered cartesian product $\bigotimes$ is defined in Section 2.2.

wins in this case. In all other cases, all left pebbles can be moved and at least one right pebble can be moved, and the players build two infinite, nonempty multipaths

$$\pi_0 = \mathbf{q}_0 \overset{a_0}{\Longrightarrow} \mathbf{q}_1 \overset{a_1}{\Longrightarrow} \cdots$$
$$\pi_1 = \mathbf{s}_0 \overset{a_0}{\Longrightarrow} \mathbf{s}_1 \overset{a_1}{\Longrightarrow} \cdots$$

with induced play $\pi = \pi_0 \times \pi_1 = \langle \mathbf{q}_0, \mathbf{s}_0 \rangle \overset{a_0}{\Longrightarrow} \langle \mathbf{q}_1, \mathbf{s}_1 \rangle \overset{a_1}{\Longrightarrow} \cdots$. We define different winning conditions $W^x \subseteq P^\omega_{\mathrm{Sp}}$ for Duplicator, for $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$, corresponding to the generalization to multiple pebbles of direct, delayed and fair simulation. The corresponding x-game is $\mathbb{G}^x_{(k_0,k_1)}(\mathbf{q}_0, \mathbf{s}_0)$.

The winning condition for delayed and fair simulation needs some technical preparation, which consists in the notion of "being existentially/universally good". Recall the definition of "being eventually good" from Section 3.2: $\mathbf{s}_i \Longrightarrow^*_F \mathbf{s}_j$ iff, for any $s_j \in \mathbf{s}_j$, there exists $s_i \in \mathbf{s}_i$ s.t. $s_i \longrightarrow^*_F s_j$. In this chapter, we write $\Longrightarrow^\exists_F$ instead of $\Longrightarrow^*_F$.

Definition 5.3.2 (Existentially good pebbles)**.** $\mathbf{s}_i$ *is* existentially good at round j *iff* $\mathbf{s}_i \Longrightarrow^\exists_F \mathbf{s}_j$*, and* $\mathbf{s}_i$ *is* existentially good*, written* $\mathrm{good}^\exists(\mathbf{s}_i)$*, iff there exists* $j \geq i$ *s.t.* $\mathbf{s}_i$ *is existentially good at round* j*. A multipath* π_1 *is* existentially fair *iff* $\mathbf{s}_i$ *is existentially good for infinitely many* i*'s.*

We introduce a dual notion: Given a source state $q_i \in \mathbf{q}_i$ and a destination state $q_j \in \mathbf{q}_j$ with $i \leq j$, we write $q_i \longrightarrow^\forall_F q_j$ iff every path $\pi = q_i \xrightarrow{a_i} q_{i+1} \xrightarrow{a_{i+1}} \cdots \xrightarrow{a_{j-i}} q_j$ going from q_i to q_j necessarily visits an intermediate accepting state $q_k \in F$, for some $i \leq k \leq j$.

Definition 5.3.3 (Universally good pebbles)**.** $\mathbf{q}_i$ *is* universally good at round j*, written* $\mathbf{q}_i \Longrightarrow^\forall_F \mathbf{q}_j$*, iff, for any* $q_j \in \mathbf{q}_j$ *and* $q_i \in \mathbf{q}_i$*,* $q_i \longrightarrow^\forall_F q_j$*. Similarly,* $\mathbf{q}_i$ *is* universally good*, written* $\mathrm{good}^\forall(\mathbf{q}_i)$*, iff there exists* $j \geq i$ *s.t.* $\mathbf{q}_i$ *is universally good at round* j*. A multipath* π_0 *is* universally fair *iff* $\mathbf{s}_i$ *is universally good for infinitely many* i*'s.*

While $\mathbf{q}_i \Longrightarrow^\forall_F \mathbf{q}_j$ does not require *per se* the existence of any path at all, $\mathbf{q}_i \Longrightarrow \mathbf{q}_j$ is guaranteed by construction.

We are now ready to define the winning condition for direct, delayed and fair multipebble simulation.

1. *Direct* (k_0, k_1)*-simulation*. Duplicator wins if, whenever *some* left pebble $q \in \mathbf{q}_i$ is accepting, then *every* right pebble $s \in \mathbf{s}_i$ is accepting:

$$\pi \in W^{\mathrm{di}} \iff (\forall i \cdot \mathbf{q}_i \cap F \neq \emptyset \implies \mathbf{s}_i \subseteq F)$$

2. *Delayed* (k_0, k_1)*-simulation*. Duplicator wins if, whenever $\mathbf{q}_i$ is universally good, then there exists $j \geq i$ s.t. $\mathbf{s}_j$ is existentially good:

$$\pi \in W^{\mathrm{de}} \iff \big(\forall i \cdot \mathrm{good}^{\forall}(\mathbf{q}_i) \implies \exists (j \geq i) \cdot \mathrm{good}^{\exists}(\mathbf{s}_j)\big)$$

3. *Fair* (k_0, k_1)*-simulation*. Duplicator wins if, whenever π_0 is universally fair, then π_1 is existentially fair:

$$\pi \in W^{\mathrm{f}} \iff (\pi_0 \text{ universally fair} \implies \pi_1 \text{ existentially fair })$$

A k_1-set $\mathbf{s}$ x-simulates a k_0-set $\mathbf{q}$, written $\mathbf{q} \sqsubseteq^x_{(k_0,k_1)} \mathbf{s}$, if Duplicator has a winning strategy in $\mathbb{G}^x_{(k_0,k_1)}(\mathbf{q}, \mathbf{s})$. We overload the simulation relation $\sqsubseteq^x_{(k_0,k_1)}$ on singletons, and write $q \sqsubseteq^x_{(k_0,k_1)} s$ iff $\{q\} \sqsubseteq^x_{(k_0,k_1)} \{s\}$. For two automata $\mathcal{A}$ and $\mathcal{B}$, let $\mathcal{A} \sqsubseteq^x_{(k_0,k_1)} \mathcal{B}$ iff $q_I^{\mathcal{A}} \sqsubseteq^x_{(k_0,k_1)} q_I^{\mathcal{B}}$, where the simulation is computed on the disjoint union of $\mathcal{A}$ and $\mathcal{B}$.

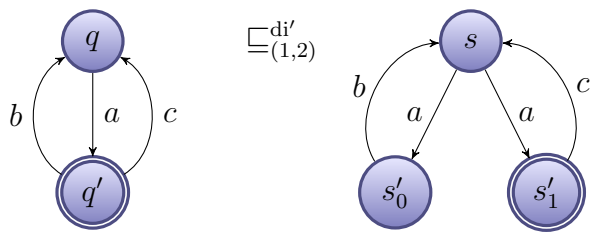
Remark 5.3.4 (Simulations for finite words). In the context of automata over finite words, one can use the following variant of multipebble direct simulation:

1'. *Direct'* (k_0, k_1)*-simulation*. Duplicator wins if, whenever *every* left pebble $q \in \mathbf{q}_i$ is accepting, then *some* right pebble $s \in \mathbf{s}_i$ is accepting:

$$\pi \in W^{\mathrm{di}'} \iff (\forall i \cdot \mathbf{q}_i \subseteq F \implies \mathbf{s}_i \cap F \neq \emptyset)$$

In [29], this notion is called *existential multipebble simulation*, while the notion we currently use in this thesis is called *universal multipebble simulation*. Multipebble existential-direct simulation is generally unsuitable for working with automata over infinite words, and we do not discuss it further in this thesis. For example, it does not imply ω-language inclusion (already for non-deterministic automata),

Example 5.3.1 - Multipebble existential-direct simulation does not imply ω-language inclusion



Note that $q \sqsubseteq^{\mathrm{di}'}_{(1,2)} s$ holds since Duplicator wins by splitting two right pebbles to $\{s'_0, s'_1\}$. However, $\mathcal{L}(q) \not\subseteq \mathcal{L}(s)$, since $(ab)^\omega \in \mathcal{L}(q)$, but $(ab)^\omega \notin \mathcal{L}(s)$.

In the next section, we explore some basic properties of multipebble simulations.

5.4 Basic properties of multipebble simulations

5.4.1 Multipaths

Fix a multipath $\pi = \mathrm{s}_0 \overset{a_0}{\Longrightarrow} \mathrm{s}_1 \overset{a_1}{\Longrightarrow} \cdots$.

Lemma 5.4.1 (Translation property)**.** *For any $h \leq i \leq j \leq l$,*

$$\mathrm{s}_i \Longrightarrow^{\forall}_F \mathrm{s}_j \quad \textit{implies} \quad \mathrm{s}_h \Longrightarrow^{\forall}_F \mathrm{s}_l$$
$$\mathrm{s}_i \Longrightarrow^{\exists}_F \mathrm{s}_j \quad \textit{implies} \quad \mathrm{s}_h \Longrightarrow^{\exists}_F \mathrm{s}_l$$

Proof. If $\mathrm{s}_i \Longrightarrow^{\forall}_F \mathrm{s}_j$, then, by definition, any path from s_i to s_j has to visit an accepting state. By definition, any path from s_h to s_l induces a path from s_i to s_j. Therefore, also any path from s_h to s_l has to visit and accepting state. That is, $\mathrm{s}_h \Longrightarrow^{\forall}_F \mathrm{s}_l$.

Assume $\mathrm{s}_i \Longrightarrow^{\exists}_F \mathrm{s}_j$. By definition of multipath, for each pebble $s'' \in \mathrm{s}_l$ there exists $s' \in \mathrm{s}_j$ s.t. $s' \longrightarrow^* s''$. By $\mathrm{s}_i \Longrightarrow^{\exists}_F \mathrm{s}_j$, there exists $s \in \mathrm{s}_i$ s.t. $s \longrightarrow^*_F s'$. Again by the definition of multipath, there exists $q \in \mathrm{s}_h$ s.t. $q \longrightarrow^* s$. Therefore,

$$q \longrightarrow^* s \longrightarrow^*_F s' \longrightarrow^* s''$$

which implies $q \longrightarrow^*_F s''$. Therefore, $\mathrm{s}_h \Longrightarrow^{\exists}_F \mathrm{s}_j$. □

The following is an immediate corollary of the previous lemma.

Corollary 5.4.2. *In a existential/universal fair multipath, pebbles are always existentially/universally good, respectively.*

$$\pi \textit{ is universally fair} \quad \textit{iff} \quad \forall j \geq 0 \cdot \mathrm{good}^{\forall}(\mathrm{s}_j)$$
$$\pi \textit{ is existentially fair} \quad \textit{iff} \quad \forall j \geq 0 \cdot \mathrm{good}^{\exists}(\mathrm{s}_j)$$

Let $\pi = \mathrm{s}_0 \overset{a_0}{\Longrightarrow} \cdots$ and $\pi' = \mathrm{s}'_0 \overset{a_0}{\Longrightarrow} \cdots$ be two multipaths.

Lemma 5.4.3 (Monotonicity)**.** *If, for any $i \leq j$, $\mathrm{s}_i \subseteq \mathrm{s}'_i$ and $\mathrm{s}_j \subseteq \mathrm{s}'_j$, then*

$$\mathrm{s}'_i \Longrightarrow^{\forall}_F \mathrm{s}'_j \quad \textit{implies} \quad \mathrm{s}_i \Longrightarrow^{\forall}_F \mathrm{s}_j$$
$$\mathrm{s}_i \Longrightarrow^{\exists}_F \mathrm{s}'_j \quad \textit{implies} \quad \mathrm{s}'_i \Longrightarrow^{\exists}_F \mathrm{s}_j$$

Proof. Immediate from the definition of being existentially/universally good. □

As a consequence of the first part, being universally good infinitely often is monotone w.r.t. taking submultipaths.

Lemma 5.4.4. *If π' is universally fair and $\pi \subseteq \pi'$, then π is universally fair.*

For two multipaths π and π' as above, define the *union multipath*

$$\pi \cup \pi' = (\mathbf{s}_0 \cup \mathbf{s}'_0) \xRightarrow{a_0} (\mathbf{s}_1 \cup \mathbf{s}'_1) \xRightarrow{a_1} \cdots$$

Existentially fair multipaths are closed under finite union.

Lemma 5.4.5. *Let π and π' be two multipaths as above, and consider the union multipath $\pi \cup \pi'$. If $\mathbf{s}_i \Longrightarrow_F^{\exists} \mathbf{s}_j$ in π and $\mathbf{s}'_i \Longrightarrow_F^{\exists} \mathbf{s}'_j$ in π', then $\mathbf{s}_i \cup \mathbf{s}'_i \Longrightarrow_F^{\exists} \mathbf{s}_j \cup \mathbf{s}'_j$ in $\pi \cup \pi'$. In particular, if π and π' are existentially fair, then $\pi \cup \pi'$ is existentially fair.*

Proof. Immediate from the definition of being existentially good. □

Inclusion of strategies We lift the inclusion of multipaths to strategies. For two Duplicator strategies σ_0 and σ_1, let $\sigma_0 \subseteq \sigma_1$ iff, for every Spoiler strategy δ and partial plays π_0 and π_1 of the same length, if π_0 is (δ, σ_0)-conform and π_1 is (δ, σ_1)-conform, then $\pi_0 \subseteq \pi_1$.

5.4.2 Hierarchies of multipebble simulations

Direct, delayed and fair simulation are naturally ordered by inclusion, like in the ordinary simulation case (cf. Lemma 2.5.4).

Lemma 5.4.6. *For every alternating Büchi automaton $\mathcal{B}$, and $k_0, k_1 > 0$,*

$$\sqsubseteq^{\mathrm{di}}_{(k_0,k_1)} \subseteq \sqsubseteq^{\mathrm{de}}_{(k_0,k_1)} \subseteq \sqsubseteq^{\mathrm{f}}_{(k_0,k_1)}$$

Moreover, for each inclusion, there exists an automaton $\mathcal{Q}$ s.t. the inclusion is strict.

Proof. The inclusion is immediate from the definition of direct, delayed and fair simulation, since the winning condition for Duplicator gets progressively weaker from left to right. The strictness follows from the ordinary 1-pebble case. □

Multipebble simulation relations are clearly monotone in the number of pebbles, since having more pebbles is never harmful for Duplicator (unneeded pebbles can always be thrown away). Thus, $\sqsubseteq^{x}_{(k_0,k_1)}$ is in general non-decreasing in k_0, k_1.

Furthermore, in certain cases more pebbles do actually give more power to Duplicator, which makes $\sqsubseteq^x_{(k_0,k_1)}$ strictly increasing in some examples. This is similar to the k-simulations for NBAs studied in [42] (cf. Section 3.2), but in our context there are two independent directions of "growing power", since Duplicator uses pebbles on both sides of the simulation game.

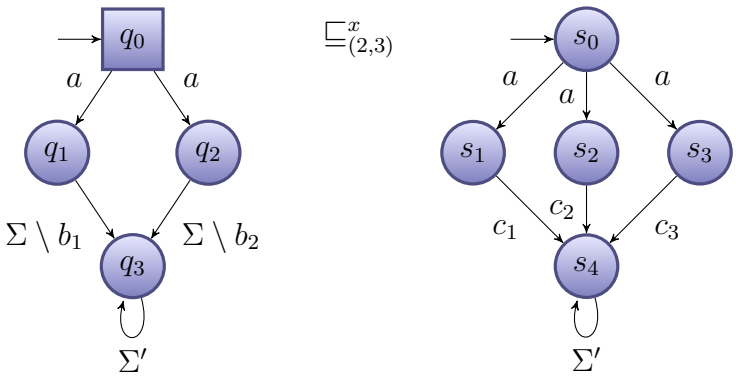
Theorem 5.4.7. *For $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$ and $0 < k_0 \leq k_0'$, $0 < k_1 \leq k_1'$,*

a) Inclusion: For every ABA $\mathcal{Q}$, $\sqsubseteq^x_{(k_0,k_1)} \subseteq \sqsubseteq^x_{(k_1',k_2')}$.

b) Strictness: If either $k_0 > k_0'$ or $k_1 > k_1'$, then there exists an ABA $\mathcal{Q}'$ such that $\sqsubseteq^x_{(k_0,k_1)} \neq \sqsubseteq^x_{(k_1',k_2')}$.

Proof. Point 1) follows directly from the definitions, and Point 2) is illustrated in the next example. □

Example 5.4.1 - The multipebble hierarchy is strict



Consider the two automata above, where $\Sigma = \{b_1, b_2, c_1, c_2, c_3\}$ and the alphabet is $\Sigma' = \{a\} \cup \Sigma$. Duplicator wins by "hedging her bets" on both sides, using two pebbles on the left and three pebbles on the right: Indeed, Spoiler plays action a, and Duplicator splits pebbles to $\langle\{q_1, q_2\}, \{s_1, s_2, s_3\}\rangle$. From the latter configuration, Spoiler cannot play neither b_1 nor b_2: Otherwise, at least one left pebble would be stuck and Duplicator would win. Therefore, Spoiler has to play an action $c_i \in \{c_1, c_2, c_3\}$, and Duplicator can move the corresponding right pebble s_i to s_4, and drop the others. From configuration $\langle q_3, s_4\rangle$, Duplicator obviously wins. Hence, $q \sqsubseteq^x_{(2,3)} s$ holds.

To see that the two left pebbles are actually necessary for Duplicator to win, it suffices to note that if there were only one left pebble, then Spoiler could play either

$$
\begin{array}{ccccccc}
\sqsubseteq^x_{(n,1)} & \subseteq & \sqsubseteq^x_{(n,2)} & \subseteq & \cdots & \subseteq & \sqsubseteq^x_{(n,n)} \\
\cup\!\mid & & \cup\!\mid & & & & \cup\!\mid \\
\vdots & & \vdots & & & & \vdots \\
\sqsubseteq^x_{(2,1)} & \subseteq & \sqsubseteq^x_{(2,2)} & \subseteq & \cdots & \subseteq & \sqsubseteq^x_{(2,n)} \\
\cup\!\mid & & \cup\!\mid & & & & \cup\!\mid \\
\sqsubseteq^x_{(1,1)} & \subseteq & \sqsubseteq^x_{(1,2)} & \subseteq & \cdots & \subseteq & \sqsubseteq^x_{(1,n)}
\end{array}
$$

Figure 5.1: Multipebble simulation hierarchy

letter b_1 or letter b_2, and every right pebble would get stuck, which is winning for Spoiler. Similarly, if there were only two right pebbles, a state $s_i \in \{s_1, s_2, s_3\}$ would necessarily remain uncovered, and Spoiler would win by playing action c_i.

This example can easily be generalized to arbitrary pebbles (k_0, k_1); moreover, similar examples can be designed using the acceptance condition instead of the stuckness condition.

The bidimensional hierarchy of multiple pebbles arising from Point 2) of the theorem above is exemplified in Figure 5.1.

5.4.3 Containment and language inclusion

A basic feature of simulation relations is to constitute an under-approximation for fair containment between states, and, more generally, for language inclusion between automata (Da). Also multipebble simulations fulfil this desideratum.

Theorem 5.4.8 (Multipebble simulations imply containment)**.** *For $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$, any ABA $\mathcal{Q}$, states $q, s \in Q$, and $k_0, k_1 > 0$,*

$$q \sqsubseteq^x_{(k_0,k_1)} s \quad \textit{implies} \quad \mathcal{L}(q) \subseteq \mathcal{L}(s)$$

By the containment between direct, delayed and fair simulation established in Lemma 5.4.6, it is sufficient to consider fair simulation, which is the coarsest one. Basically, the theorem is proved by showing that, if $w \in \mathcal{L}(q)$, then Spoiler can play the simulation game in order to force Duplicator witness $w \in \mathcal{L}(s)$. This requires three main ingredients:

- First, universally fair multipaths should be *necessary* for $w \in \mathcal{L}(q)$. That is, $w \in \mathcal{L}(q)$ implies that Spoiler can force a universally fair multipath on the left.
- Second, if the left multipath is universally fair, then the right multipath is existentially fair. This follows immediately from the definition of fair simulation.
- Third, existentially fair multipaths should be *sufficient* for the existence of fair ordinary paths.

From the third condition, we would like to conclude $w \in \mathcal{L}(s)$. Recall that acceptance for alternating automata is defined in terms of a game between Automaton (which wants to show acceptance) and Pathfinder (which wants to disprove it). Thus, to conclude $w \in \mathcal{L}(s)$, one should show a winning strategy for Automaton in the acceptance game. What the third point ensures, instead, is that, for any *fixed* Pathfinder's strategy, Automaton has a winning counter-strategy. That is, Pathfinder does not have a winning strategy. Since the acceptance game is determined, Automaton has a winning strategy.

Remark 5.4.9. This issue does not arise in 1-pebble games, since, in those games, Duplicator's strategy in the simulation game *step-wise* induces a strategy for Automaton in the acceptance game. Consequently, a winning strategy for Automaton can be built on the fly from Duplicator's strategy in 1-pebble games.

In the following, let $w = a_0 a_1 \cdots$ and assume that $\pi = \mathbf{q}_0 \overset{a_0}{\Longrightarrow} \mathbf{q}_1 \overset{a_1}{\Longrightarrow} \cdots$ and $\pi' = \mathbf{s}_0 \overset{a_0}{\Longrightarrow} \mathbf{s}_1 \overset{a_1}{\Longrightarrow} \cdots$ are the left and right multipaths at the end of the simulation game, respectively.

The second ingredient holds by the definition of fair simulation. The third ingredient is the content of the following lemma.

Lemma 5.4.10. *If π' is an existentially fair multipath, then there exists a fair path $\pi_{acc} \in \pi'$.*

Proof. We define the sequence of indices $\{i_j\}_{j \geq 0}$ as follows: $i_0 = 0$ and, inductively, i_j is the least index $i > i_{j-1}$ s.t. $\mathbf{s}_{j-1} \Longrightarrow_F^* s_i$. The sequence $i_0, i_1, \cdots$ is well-defined by Corollary 5.4.2, since π is fair. For any $j \geq 0$, consider the finite prefix $\pi_j = \mathbf{s}_0 \overset{a_0}{\Longrightarrow} \mathbf{s}_1 \overset{a_1}{\Longrightarrow} \cdots \overset{a_{i_{j-1}}}{\Longrightarrow} \mathbf{s}_{i_j}$ of π'. Fix any state $s \in \mathbf{s}_{i_j}$. By the definition of being existentially good, there exists a path $\pi'_j \in \pi_j$ ending in s which visits at least j accepting states.

Let $\Pi = \pi'_0, \pi'_1, \ldots$. Clearly, Π is a coherent sequence of paths. By Lemma 4.4.7, there exists a fair path $\pi_{\text{acc}} \in \pi'$. $\square$

It remains to settle the first ingredient. To this end, we have to argue that, if there exists an accepting run, then there exists a universally fair multipath. By Lemma 5.4.4, one can discard non-maximal multipaths, and concentrate on multipaths which are "large" by construction. Such "large" multipaths can be obtained by requiring that universal states induce all their successors in the multipath; we call multipaths with this property *U-saturating multipaths*.

Definition 5.4.11 (U-saturating multipaths)**.** *A multipath* $\pi_0 = \mathbf{q}_0 \overset{a_0}{\Longrightarrow} \mathbf{q}_1 \overset{a_1}{\Longrightarrow} \cdots$ *is* U-saturating *iff, for any* $k \geq 0$, $\bigcup \Delta(\mathbf{q}_k^U, a_k) \subseteq \mathbf{q}_{k+1}$.

U-saturating multipaths are necessary for the existence of an accepting run.[2]

Lemma 5.4.12. *If* $w = a_0 a_1 \cdots \in \mathcal{L}(q)$*, then there exists a universally fair* U*-saturating multipath* $\pi = \mathbf{q}_0 \overset{a_0}{\Longrightarrow} \mathbf{q}_1 \overset{a_1}{\Longrightarrow} \cdots$ *starting at* $\mathbf{q}_0 = \{q\}$.

Remark 5.4.13. U-saturating multipaths are just a different representation for strategies of Pathfinder in the acceptance game $\mathbb{G}^\omega(q, w)$ (with $\mathbf{q}_0 = \{q\}$), and universally fair such multipaths represent winning strategies.

In general, Duplicator tries to prevent Spoiler from building a universally fair multipath on the left. By Lemma 5.4.4, Duplicator is always better off by splitting pebbles maximally on the left and, when she has sufficiently many left pebbles, she can force a U-saturating multipath π on the left. Such strategies are called *left-blind*.

Definition 5.4.14 (Left blind strategy)**.** *A Duplicator's strategy* σ *is* left-blind *iff, for every round* k*, if* $\sigma(\pi_k)(a_k, \mathbf{q}'_k, \mathbf{s}'_k) = (\mathbf{q}_{k+1}, \mathbf{s}_{k+1})$*, then* $\mathbf{q}_{k+1} = \mathbf{q}'_k \cup \bigcup \Delta(\mathbf{q}_k^U, a_k)$.

By definition, if $\pi \times \pi'$ is conform to a left-blind strategy, then π is U-saturating. Therefore, left-blind strategies induce U-saturating left multipaths, and, consequently, left-blind strategies suffice for Duplicator in (n, k)-fair (and delayed) simulation games.

Lemma 5.4.15. *Let* $x \in \{\mathrm{de}, \mathrm{f}\}$*. If* $q \sqsubseteq^x_{(n,k)} s$*, then there exists a left-blind winning strategy for Duplicator.*

Before we turn to the proof of the lemma, let us introduce the useful notion of *restricted move*. Sometimes we consider Spoiler's move when restricted to a certain subset of states. Recall that, when Spoiler moves from configuration $\langle \mathbf{q}, \mathbf{s} \rangle$, she chooses a successor for each state in $\mathbf{q}^E$ and $\mathbf{s}^U$. For subsets $\widehat{\mathbf{q}} \subseteq \mathbf{q}$ and $\widehat{\mathbf{s}} \subseteq \mathbf{s}$, the restriction of Spoiler's move to $\widehat{\mathbf{q}}, \widehat{\mathbf{s}}$ is just this choice restricted to $\widehat{\mathbf{q}}^E, \widehat{\mathbf{s}}^E$.

[2] In fact, they are also sufficient, but we do not need this property.

Definition 5.4.16 (Restricted move)**.** *Let* $m = (\mathbf{q}, \mathbf{s}, a, \mathbf{q}', \mathbf{s}') \in \Gamma^{\mathrm{Sp}}$ *be a valid move by Spoiler. By definition,* $\mathbf{q}' \in \bigotimes \Delta(\mathbf{q}^E, a)$ *and* $\mathbf{s}' \in \bigotimes \Delta(\mathbf{s}^U, a)$*. For* $\widehat{\mathbf{q}} \subseteq \mathbf{q}$ *and* $\widehat{\mathbf{s}} \subseteq \mathbf{s}$*, the move* m restricted to $\widehat{\mathbf{q}}, \widehat{\mathbf{s}}$*, written* $m{\downarrow_{\widehat{\mathbf{q}},\widehat{\mathbf{s}}}}$*, is defined as follows (see Section 2.2 for the notation* $\downarrow$*):*

$$m{\downarrow_{\widehat{\mathbf{q}},\widehat{\mathbf{s}}}} = (\widehat{\mathbf{q}}, \widehat{\mathbf{s}}, a, \widehat{\mathbf{q}}', \widehat{\mathbf{s}}') \in \Gamma^{\mathrm{Sp}}$$

where

$$\widehat{\mathbf{q}}' = \mathbf{q}'{\downarrow_{\Delta(\widehat{\mathbf{q}}^E,a)}} \in \bigotimes \Delta(\widehat{\mathbf{q}}^E, a)$$
$$\widehat{\mathbf{s}}' = \mathbf{s}'{\downarrow_{\Delta(\widehat{\mathbf{s}}^U,a)}} \in \bigotimes \Delta(\widehat{\mathbf{s}}^U, a)$$

Restricted moves are used below in the proof of Lemma 5.4.15. They are also used in Sections 5.6 and 5.7.

Proof (of Lemma 5.4.15). Let σ be a Duplicator's winning strategy in $\mathbb{G} = \mathbb{G}^x_{(n,k)}(\mathbf{q}, \mathbf{s})$. We define a left-blind strategy σ^{lb} which acts like σ on right pebbles, and show that it is winning as well. As the main game $\mathbb{G}$ evolves according to σ^{lb}, we bookkeep an auxiliary game $\mathbb{G}'$ (which is of the same type as $\mathbb{G}$) which evolves according to σ. The two games progress in locksteps, while maintaining the following invariant: If

$$\pi^{\mathrm{lb}}_k = \langle \mathbf{q}^{\mathrm{lb}}_0, \mathbf{s}_0 \rangle \overset{a_0}{\Longrightarrow} \langle \mathbf{q}^{\mathrm{lb}}_1, \mathbf{s}_1 \rangle \overset{a_1}{\Longrightarrow} \cdots \langle \mathbf{q}^{\mathrm{lb}}_k, \mathbf{s}_k \rangle$$

is the current σ^{lb}-conform partial play in $\mathbb{G}$, then the σ-conform partial play π_k in $\mathbb{G}'$ is

$$\pi_k = \langle \mathbf{q}_0, \mathbf{s}_0 \rangle \overset{a_0}{\Longrightarrow} \langle \mathbf{q}_1, \mathbf{s}_1 \rangle \overset{a_1}{\Longrightarrow} \cdots \langle \mathbf{q}_k, \mathbf{s}_k \rangle$$

where $\mathbf{q}_i \subseteq \mathbf{q}^{\mathrm{lb}}_i$ for any $0 \leq i \leq k$; therefore, $\pi_k \subseteq \pi^{\mathrm{lb}}_k$.

At round k, let Spoiler move in $\mathbb{G}$ as $m = (\mathbf{q}^{\mathrm{lb}}_k, \mathbf{s}_k, a_k, \mathbf{q}', \mathbf{s}') \in \Gamma^{\mathrm{Sp}}$. By the invariant, $\mathbf{q}_k \subseteq \mathbf{q}^{\mathrm{lb}}_k$. Thus, move m induces, in $\mathbb{G}'$, the restricted move $m{\downarrow_{\mathbf{q}_k,\mathbf{s}_k}} = (\mathbf{q}_k, \mathbf{s}_k, a_k, \widehat{\mathbf{q}}', \mathbf{s}')$, for some $\widehat{\mathbf{q}}' \subseteq \mathbf{q}'$. We apply σ in $\mathbb{G}'$, to obtain $\sigma(\pi_k, (\mathbf{q}_k, \mathbf{s}_k, a_k, \widehat{\mathbf{q}}', \mathbf{s}')) = (\mathbf{q}'', \mathbf{s}'')$. We define $\sigma^{\mathrm{lb}}(\pi^{\mathrm{lb}}_k, (\mathbf{q}^{\mathrm{lb}}_k, \mathbf{s}_k, a_k, \mathbf{q}', \mathbf{s}')) = (\mathbf{q}_{k+1}, \mathbf{s}_{k+1})$ in $\mathbb{G}$, where $\mathbf{q}_{k+1} = \widehat{\mathbf{q}}' \cup \bigcup \Delta(\mathbf{q}^{\mathrm{lb}}_k \cap U, a_k)$ and $\mathbf{s}_{k+1} = \mathbf{s}''$. Clearly, σ^{lb} is left-blind.

We now argue that σ^{lb} is winning for delayed simulation. Let $\pi^{\mathrm{lb}} = \mathbf{q}^{\mathrm{lb}}_0 \overset{a_0}{\Longrightarrow} \mathbf{q}^{\mathrm{lb}}_1 \overset{a_1}{\Longrightarrow} \cdots$, $\pi = \mathbf{q}_0 \overset{a_0}{\Longrightarrow} \mathbf{q}_1 \overset{a_1}{\Longrightarrow} \cdots$, and $\pi' = \mathbf{s}_0 \overset{a_0}{\Longrightarrow} \mathbf{s}_1 \overset{a_1}{\Longrightarrow} \cdots$. Assume $\mathbf{q}^{\mathrm{lb}}_i \Longrightarrow^{\forall}_F \mathbf{q}^{\mathrm{lb}}_j$. Since $\pi \subseteq \pi^{\mathrm{lb}}$, by Lemma 5.4.4, $\mathbf{q}_i \Longrightarrow^{\forall}_F \mathbf{q}_j$. But σ is winning in $\mathbb{G}'$, therefore there exists $k \geq i$ s.t. $\mathbf{s}_i \Longrightarrow^{\exists}_F \mathbf{s}_k$. The argument for fair simulation is similar. □

We are now finally ready to prove that simulation implies language inclusion.

Proof (of Theorem 5.4.8). It suffices to prove the claim for $k_0 = k_1 = n$, where n is the cardinality of the automaton, since, by Theorem 5.4.7, (n, n)-simulation contains (k_0, k_1)-simulation.

Assume $q \sqsubseteq^{\mathsf{f}}_{(n,n)} s$ and let $w = a_0 a_1 \cdots \in \mathcal{L}(q)$. Let $\mathbb{G}^{\mathsf{sim}} = \mathbb{G}^{\mathsf{f}}_{(n,n)}(q, s)$ be the fair simulation game between q and s, let $\mathbb{G}^{\mathsf{acc}}_0 = \mathbb{G}^{\omega}(q, w)$ be the acceptance game for w from q, and let $\mathbb{G}^{\mathsf{acc}}_1 = \mathbb{G}^{\omega}(s, w)$ be the acceptance game for w from s. We show that, for any Pathfinder's strategy δ in $\mathbb{G}^{\mathsf{acc}}_1$, there exists a counter-strategy for Automaton in $\mathbb{G}^{\mathsf{acc}}_1$ s.t. Automaton wins the resulting play. Since the acceptance game is determined, it follows that Automaton has a winning strategy in $\mathbb{G}^{\mathsf{acc}}_1$. Thence, $w \in \mathcal{L}(s)$.

Fix a strategy δ for Pathfinder in $\mathbb{G}^{\mathsf{acc}}_1$. Let $\mathbf{q}_0 = \{q\}$ and $\mathbf{s}_0 = \{s\}$. The structure of the proof is as follows.

1. Since $w \in \mathcal{L}(q)$, by Lemma 5.4.12, there exists a universally fair U-saturating multipath

$$\pi = \mathbf{q}_0 \overset{a_0}{\Longrightarrow} \mathbf{q}_1 \overset{a_1}{\Longrightarrow} \cdots$$

2. By Lemma 5.4.15, we can assume that Duplicator in $\mathbb{G}^{\mathsf{sim}}$ uses a left-blind winning strategy σ^{lb}. Therefore, Duplicator plays in a uniform way on the left, by splitting pebbles maximally to all successors of universal states.

3. On the left of the simulation game, Spoiler plays as to force π (which is possible, since Duplicator plays according to the left-blind strategy σ^{lb}). On the right, Spoiler's moves are determined by Pathfinder's strategy δ (as we detail later).

4. Let $\pi_\omega = \pi \times \pi'$ be the resulting σ^{lb}-conform play, where

$$\pi' = \mathbf{s}_0 \overset{a_0}{\Longrightarrow} \mathbf{s}_1 \overset{a_1}{\Longrightarrow} \cdots$$

5. Since σ^{lb} is winning and π is universally fair, π' is existentially fair.

6. By Lemma 5.4.10, there exists a fair path $\pi_{\mathrm{acc}} = s \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots \in \pi'$.

7. The winning strategy δ' of Automaton in $\mathbb{G}^{\mathsf{acc}}_1$ is defined as to force the play π_{acc}. Since π_{acc} is fair, Automaton wins this play.

It remains to explain Point 3. above, i.e., how Pathfinder's strategy δ in $\mathbb{G}^{\mathsf{acc}}_1$ determines how Spoiler plays on the right of the simulation game. This is done by running several $\mathbb{G}^{\mathsf{acc}}_1$-acceptance games in parallel; for each one of them, we bookkeep the current,

δ-conform partial play by using a *logbook* [55]. Assume that, at round k, the current partial play in $\mathbb{G}^{\mathsf{sim}}$ is

$$\pi_k = \langle \mathbf{q}_0, \mathbf{s}_0 \rangle \xRightarrow{a_0} \langle \mathbf{q}_1, \mathbf{s}_1 \rangle \xRightarrow{a_1} \cdots \xRightarrow{a_{k-1}} \langle \mathbf{q}_k, \mathbf{s}_k \rangle$$

and that the remaining input word to be read is $w_k = a_k a_{k+1} \ldots$. Then, a logbook L_k for round k is a finite set of partial plays from $\mathbb{G}_1^{\mathsf{acc}}$:

$$L_k = \{\pi_{k,j} := \langle s_{0,j}, w_0 \rangle \xrightarrow{a_0} \langle s_{1,j}, w_1 \rangle \xrightarrow{a_1} \cdots \xrightarrow{a_{k-1}} \langle s_{k,j}, w_k \rangle \mid 1 \leq j \leq j_k\}$$

There are $|L_k| = j_k$ partial plays in L_k. We say that L_k is *valid* if it further satisfies the following *logbook properties*:

$$\mathbf{s}_k = \{s_{k,1}, \ldots, s_{k,j_k}\} \tag{LP1}$$

$$\forall (\pi_{k,j} \in L_k) \ \cdot \ \pi_{k,j} \text{ is a } \delta\text{-conform partial play} \tag{LP2}$$

(LP1) is a "covering" condition; it says that, for each right pebble, there is a corresponding entry in the logbook. Condition (LP2) ensures that the logbook records only δ-conform partial plays.

In the first round $k = 0$, the initial $\mathbb{G}^{\mathsf{sim}}$-configuration is $\langle \mathbf{q}_0, \mathbf{s}_0 \rangle$, and the acceptance game $\mathbb{G}_1^{\mathsf{acc}}$ is in configuration $\langle s, w \rangle$. Therefore, $L_0 = \{\langle s, w \rangle\}$ is a valid logbook.

Inductively, assume that L_k is a valid logbook for round k, and that the current configuration of the simulation game is $\langle \mathbf{q}_k, \mathbf{s}_k \rangle$. Then, Spoiler plays input symbol a_k. The next left configuration $\mathbf{q}_{k+1}$ is uniquely determined by π, as discussed in Points 1.–3. above. The next right configuration $\mathbf{s}_{k+1}$ is determined as follows.

- For any right pebble $s \in \mathbf{s}_k$, let π'_s be the corresponding δ-conform partial play in L_k ending in s, which exists by (LP1) and (LP2).

- For each universal right pebble $s \in \mathbf{s}_k^U$, a successor state s' is obtained by applying Pathfinder's strategy δ:

$$s' := \delta(\pi'_s)$$

- This induces a Spoiler's move on the right in the simulation game: Let $\mathbf{s}' = \{\delta(\pi'_s) \mid s \in \mathbf{s}_k^U\}$, then Spoiler moves as

$$(\mathbf{q}_k, \mathbf{s}_k, a_k, \mathbf{q}', \mathbf{s}') \in \Gamma^{\mathrm{Sp}}$$

  for some $\mathbf{q}' \subseteq \mathbf{q}_{k+1}$ uniquely determined by π.

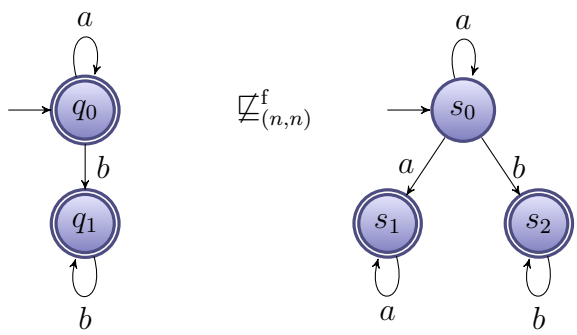
- Duplicator applies her strategy σ^{lb}, obtaining

$$\sigma^{\text{lb}}(\pi_k, (\mathbf{q}_k, \mathbf{s}_k, a_k, \mathbf{q}', \mathbf{s}')) = \langle \mathbf{q}_{k+1}, \mathbf{s}_{k+1} \rangle.$$

The logbook L_{k+1} is updated as follows. For any $s' \in \mathbf{s}_{k+1}$ and $s \in \mathbf{s}_k$ s.t. $s \xrightarrow{a_k} s'$, add $\pi_{s'} := \pi'_s \xrightarrow{a_k} s'$ to L_{k+1}. Clearly, $\pi_{s'}$ is δ-conform. Therefore, only conform plays are added to L_{k+1}, and condition (LP2) is satisfied. Since the process is repeated for any $s' \in \mathbf{s}_{k+1}$, also condition (LP1) is satisfied, and L_{k+1} is a valid logbook. □

Language inclusion is not "reached" by either direct, delayed or fair simulation; in fact, even (n, n)-fair simulation is finer than language inclusion.

Example 5.4.2 - Multipebble fair simulation is not necessary for ω-language inclusion



Consider the two automata above, for which inclusion holds, but simulation doesn't. Indeed, $\mathcal{L}(q_0) = \mathcal{L}(s_0) = a^\omega + a^*b^\omega$, therefore inclusion holds. However, $q_0 \not\sqsubseteq^x_{(n,n)} s_0$, for any $x \in \{\text{di}, \text{de}, \text{f}\}$. To see this, it suffices to consider fair simulation (which is the coarsest by Lemma 5.4.6), for which we show a winning strategy for Spoiler. From the initial configuration $\langle \{q_0\}, \{s_0\} \rangle$, Spoiler starts by repeatedly choosing transition $q_0 \xrightarrow{a} q_0$. Duplicator can either remain in s_0, or "hedge her bets" by going to $\{s_0, s_1\}$. In either case, Spoiler keeps looping on q_0. Since q_0 is accepting, Duplicator has to eventually take the pebble on s_0 away, because the latter is not accepting. Therefore, at some point the game is forced to configuration $\langle \{q_0\}, \{s_1\} \rangle$, from which Spoiler plays transition $q_0 \xrightarrow{b} q_1$ and Duplicator loses, his remaining pebble on s_1 being stuck.

The use of the stuckness condition is not essential, and one can similarly argue with the acceptance condition (by adding a rejecting sink state).

5.5 Uniform simulation games

In this section, we show that, without loss of generality, multipebble simulation games can be simplified in order to satisfy some useful restrictions. We call these restricted games *uniform simulation games*. There is no loss of generality in the sense that we can translate arbitrary games to uniform games, while preserving the winner of the game.[3] The properties of uniform games will be generally useful for the presentation of more advanced topics; in certain cases, like in transitivity (cf. Section 5.6), it is even not clear how to conduct the proofs without these restrictions in place.

Let $\pi = \pi_0 \times \pi_1$ be a play of the game over $w = a_0 a_1 \cdots$, with $\pi_0 = \mathbf{q}_0 \overset{a_0}{\Longrightarrow} \mathbf{q}_1 \overset{a_1}{\Longrightarrow} \cdots$ and $\pi_1 = \mathbf{s}_0 \overset{a_0}{\Longrightarrow} \mathbf{s}_1 \overset{a_1}{\Longrightarrow} \cdots$. Uniform simulation games satisfy the following properties:

(U1) At any round k, pebbles are either all existential or all universal: $\mathbf{q}_k \cup \mathbf{s}_k \in 2^E \cup 2^U$.

(U2) Pebbles are never "taken away" by Duplicator. I.e., for any k and for any $s \in \mathbf{s}_k$, there exists $s' \in \mathbf{s}_{k+1}$ s.t. $s \xrightarrow{a_k} s'$, and similarly for $\mathbf{q}_k$.

(U3) Pebbles merge only in distinguished sink states. I.e., for $s_0, s_1 \in \mathbf{s}_k$ and $s' \in \mathbf{s}_{k+1}$, if $s_0 \xrightarrow{a_k} s'$ and $s_1 \xrightarrow{a_k} s'$, then s' is a special sink state.

(U4) Existentially fair multipaths are also universally fair. (This follows immediately from the previous point.)

Condition (U1) ensures that, at any given round, Spoiler and Duplicator control either all or none of the pebbles of a given side. Condition (U2) facilitates reasoning since pebbles never "disappear" from the automaton; instead, they always go somewhere, perhaps in a special sink state. Condition (U3) guarantees that pebbles can only merge in a restricted fashion. Finally, Condition (U4) simplifies reasoning about delayed and fair simulation when games are composed together (like for transitivity); it follows from (U3) when the sink state is accepting: Indeed, if a pebble which has not visited an accepting state is about to merge with a pebble which has visited an accepting state (a good pebble), then the former pebble is moved directly to an accepting sink state.

Definition 5.5.1 (Uniform simulation games)**.** *A simulation game satisfying (U1)-(U4) is called a* uniform simulation game.

[3] The translation that we present does not preserve the language, or other features of the automaton in general.

We translate a non-uniform simulation game into a uniform game in a natural way: Given a simulation game $\mathcal{G}$ between two automata $\mathcal{Q}$ and $\mathcal{S}$, we construct two *uniform automata* $\mathcal{U}(\mathcal{Q})$ and $\mathcal{U}(\mathcal{S})$ s.t. the simulation game $\mathcal{G}'$ between $\mathcal{U}(\mathcal{Q})$ and $\mathcal{U}(\mathcal{S})$ is a uniform simulation game. We present this translation in Section 5.5.1.

Of course, after proving the relevant properties in the simpler setting of uniform games, we need a way to transfer these results back in the original non-uniform setting. For this purpose, we use the following *bridge lemma*, which shows how winning strategies for Duplicator can be translated back and forth between uniform and non-uniform games.

Lemma 5.5.2 (Bridge Lemma)**.** *For any simulation game $\mathcal{G}$, let $\mathcal{G}'$ be the associated uniform simulation game (as above). Then,*

- *If σ is a winning strategy for Duplicator in $\mathcal{G}$, then there exists a winning strategy σ' for Duplicator in $\mathcal{G}'$.*
- *Conversely, if σ' is a winning strategy for Duplicator in $\mathcal{G}'$, then there exists a winning strategy σ for Duplicator in $\mathcal{G}$.*

Consequently, Duplicator wins $\mathcal{G}$ iff she wins $\mathcal{G}'$.

The Bridge Lemma is proved in Section 5.5.2.

5.5.1 Translation to uniform automata

We explain the construction in an incremental fashion. The idea is to expand a single transition in $\mathcal{Q}$ into two sequential transitions in $\mathcal{U}(\mathcal{Q})$, plus some auxiliary control transitions. For the time being, we ignore the acceptance condition and we concentrate on the branching structure of the automaton. Accepting states will be considered later.

Existential transitions We explain the idea by first showing how to translate an existential transition in $\mathcal{Q}$ (i.e., a transition from an existential state). In order to satisfy (U1), the transition structure of $\mathcal{U}(\mathcal{Q})$ should be s.t. existential and universal states alternate in a fixed pattern. We use the transition pattern existential-universal: An existential transition $p \xrightarrow{a} q$ in $\mathcal{Q}$ with $p \in E$ is translated into two transitions in $\mathcal{U}(\mathcal{Q})$:

$$(\!|p|\!)^I \xrightarrow{a} [\![p, a, q]\!]^{II} \xrightarrow{a} (\!|q|\!)^I$$

where $(\!|p|\!)^I$ is existential and $[\![p, a, q]\!]^{II}$ is universal. We say that a pebble on $(\!|p|\!)^I$ is in *stage I*, and that a pebble on $[\![p, a, q]\!]^{II}$ is in *stage II*.

To accommodate (U2), we introduce an inert sink state $\perp$, and when Duplicator wishes to take pebbles away in $\mathcal{Q}$, we instead allow her to move pebbles to $\perp$ in $\mathcal{U}(\mathcal{Q})$. This also addresses (U3), since Duplicator can always avoid pebbles to merge by moving some of them to $\perp$. Since both left and right pebbles are generally taken away by Duplicator, we add transitions to $\perp$ from both stage I and stage II states:

$$\begin{tikzcd}
(\!|p|\!)^I \arrow[r, "a"] \arrow[drrr, "a"'] & {[\![p, a, q]\!]^{II}} \arrow[r, "a"] \arrow[dr, "a"] & (\!|q|\!)^I & \\
 & & \perp \arrow[loop below, "\Sigma"] &
\end{tikzcd}$$

So far, nothing prevents Duplicator from moving all right pebbles to $\perp$ and trivially win from there (since $\perp$ can perform any action in Σ). We need a way to ensure that at least one right pebble is *not on* $\perp$. We introduce a new input symbol $\checkmark$, which is a control symbol that is available only from stage II states; in particular, $\checkmark$ is not available from $\perp$:

$$\begin{tikzcd}
(\!|p|\!)^I \arrow[r, "a"] \arrow[drrr, "a"'] & {[\![p, a, q]\!]^{II}} \arrow[r, "a"] \arrow[dr, "a, \checkmark"] & (\!|q|\!)^I & \\
 & & \perp \arrow[loop below, "\Sigma"] &
\end{tikzcd}$$

In this way, if Duplicator moves every right pebble on $\perp$ at stage I, then Spoiler would play $\checkmark$ at stage II and immediately win, since no pebble on the right could do $\checkmark$. Therefore, this effectively prevents Duplicator from moving every right pebble to $\perp$.

However, Duplicator could still move some left pebble to $\perp$ at stage II, thus inhibiting Spoiler from future uses of $\checkmark$—since $\perp$ cannot do $\checkmark$. We prevent this by splitting the sink state into a left $\perp_L$ and right $\perp_R$ copy, and by further allowing the left copy $\perp_L$ to perform a $\checkmark$ action:

$$\begin{tikzcd}
(\!|p|\!)^I \arrow[r, "a"] \arrow[dr, "a"'] & {[\![p, a, q]\!]^{II}} \arrow[r, "a"] \arrow[dr, "a, \checkmark"] & (\!|q|\!)^I & \\
 & \perp_R \arrow[loop below, "\Sigma"] & \perp_L \arrow[loop below, "\Sigma, \checkmark"] &
\end{tikzcd}$$

Now, action ✓ is no longer disabled by the presence of any left pebble on $\bot_L$. Incidentally, this also prevents Duplicator from moving *all* left pebbles to $\bot_L$: Should this happen, action ✓ would be enabled in any following round; in particular, Spoiler could play ✓ in stage I of the next round, and Duplicator would lose since no state on the right can do ✓ at stage I. This concludes the description for translating existential transitions.

Universal transitions The translation of a universal transition $p \xrightarrow{a} q$ with $p \in U$ is the same, with the only difference that stage II states are of the form $[\![p, a]\!]^{II}$, and thus independent from q:

$$\begin{tikzcd}
(\!|p|\!)^I \arrow[r, "a"] \arrow[dr, "a"'] & {[\![p,a]\!]^{II}} \arrow[r, "a", "a, \checkmark"'] \arrow[dr] & (\!|q|\!)^I \\
& \bot_R \arrow[loop below, "\Sigma"] & \bot_L \arrow[loop below, "\Sigma, \checkmark"]
\end{tikzcd}$$

Branching structure Summing up, the translation preserves the branching structure in the following sense:

- For existential transitions, choice happens at stage I states, while stage II states are essentially deterministic (except for control transitions). Indeed, stage I states $(\!|p|\!)^I$ have one successor $[\![p, a, q]\!]^{II}$ for each transition $p \xrightarrow{a} q$. On the other hand, stage II states $[\![p, a, q]\!]^{II}$ are deterministic (except for control transitions to $\bot_L$), since the successor state $(\!|q|\!)^I$ is uniquely determined.

- For universal transitions, the situation is symmetric: Choice happens at stage II states, while stage I state are essentially deterministic. Indeed, stage II states $[\![p, a]\!]^{II}$ have one successor state $(\!|q|\!)^I$ for each transition $p \xrightarrow{a} q$. On the other hand, stage I states $(\!|p|\!)^I$ are deterministic (except for the control transition to $\bot_R$), since the successor state $[\![p, a]\!]^{II}$ is uniquely determined by state p and action a.

Accepting states For delayed and fair simulation, both $\bot_L$ and $\bot_R$ are accepting. Indeed, for pebbles which are taken away to be harmless, they need to go to an accepting state and become good. For direct simulation, this is not correct, and the left sink state $\bot_L$ needs to be rejecting. Indeed, if a left pebble is thrown away to $\bot_L$, this should not trigger any obligation for Duplicator. This is the sole difference between the reduction for delayed/fair simulation and direct simulation.

For non-sink states, the acceptance condition is dealt with at stage II states: all stage I states are rejecting, and stage II states $[\![p, a, q]\!]^{II}$ and $[\![p, a]\!]^{II}$ are accepting iff the corresponding source state p is accepting.

Remark 5.5.3. For delayed and fair simulation, the exact time when pebbles are accepting is irrelevant (provided they are eventually accepting), and we could have given an equally valid definition by having the acceptance condition on stage I states. However, for direct simulation, having the acceptance condition shifted from stage I to stage II states is crucial. Indeed, since accepting states have to be matched instantaneously in direct simulation, if the acceptance condition were on stage I states and some left pebble were accepting, then Duplicator would need to throw away some right stage II pebble in order to have only accepting ones. Since right stage II pebbles are universal, Duplicator does not control them, and thus she cannot move them to a sink state— throwing them away would contradict requirement (U2).

Spoiler's point of view By the above discussion, we have shown that Duplicator gains no extra power in the uniform simulation game. We now argue similarly for Spoiler.

- Spoiler never moves any *right* pebble in $\perp_L$ at stage II. Should this happen, Duplicator would win as follows. Duplicator simultaneously moves all left pebbles on $\perp_L$, and all right pebbles on $\perp_R$ at stage I of the next round. From the latter position, the right pebble on $\perp_L$ can now perform any action, so Spoiler cannot anymore win by playing ✓. For the acceptance condition, we have two cases.

  1. For delayed and fair simulation, all pebbles on sink states are accepting. Therefore, the winning condition is satisfied.
  2. For direct simulation, all left pebbles are on $\perp_L$, which is not accepting in this case. Thus, no obligation is triggered at all.

- Symmetrically, Spoiler never moves any *left* pebble on $\perp_R$ at stage I. Should this happen, action ✓ will be permanently disabled and Duplicator would win as follows. Duplicator moves all right pebbles to $\perp_R$, which is usually losing for Duplicator when Spoiler can play ✓ at stage II. But the left pebble on $\perp_R$ disables ✓ from being played, and Duplicator wins by safely staying with all right pebbles in $\perp_R$. The winning condition is satisfied for both delayed/fair and direct simulation, since all right pebbles are on the accepting sink $\perp_R$.

- Spoiler can play the control action ✓ to punish Duplicator only if all right pebbles are on $\bot_R$. Otherwise, if Spoiler plays ✓ while some right stage II pebble is not on $\bot_R$, at least one right pebble ends up in $\bot_L$. This last event, as we have already seen in the first point above, is winning for Duplicator.

This concludes the informal description of the translation. We now formally describe the translation; see also Figure 5.2.

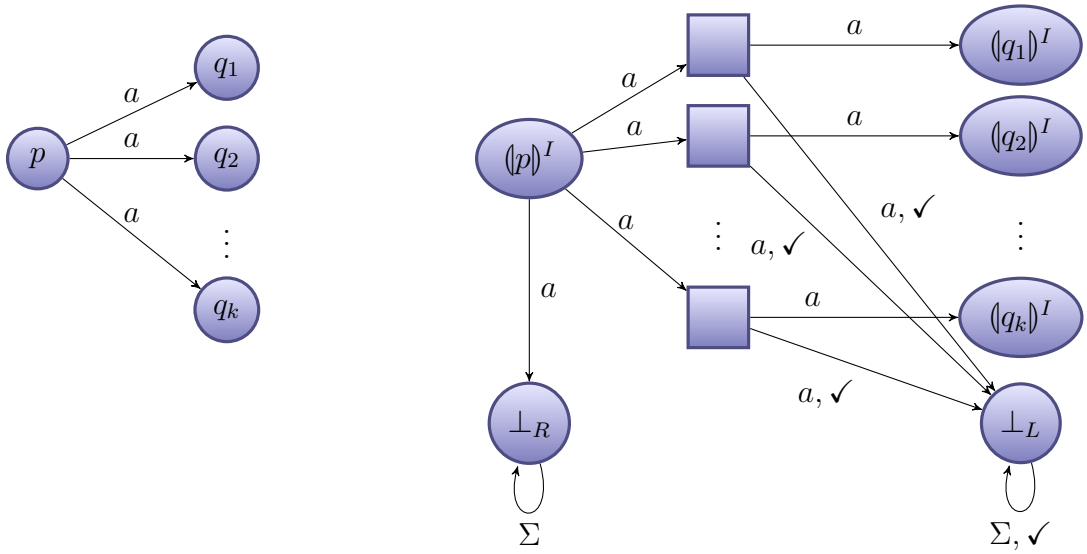
Definition 5.5.4. *For an ABA* $\mathcal{Q} = (Q, \Sigma, q_I, \Delta, E, U, F)$ *and* $x = \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$*, we define its* x-uniformization *as the ABA* $\mathcal{U}^x(\mathcal{Q}) = (Q', \Sigma', q'_I, \Delta', E', U', F'_x)$*, where*

$$
\begin{aligned}
\Sigma' &= \Sigma \cup \{\checkmark\} \\
E' &= \{(\!|p|\!)^I \mid p \in Q\} \cup \{\bot_R, \bot_L\} \\
U' &= \{[\![p, a, q]\!]^{II} \mid p \in E \textit{ and } (p, a, q) \in \Delta\} \cup \\
&\quad \cup \{[\![p, a]\!]^{II} \mid p \in U, a \in \Sigma\} \\
Q' &= E' \cup U' \\
q'_I &= (\!|q_I|\!)^I \\
F'_{\mathrm{de}} &= F'_{\mathrm{f}} = \{[\![p, a, q]\!]^{II}, [\![p, a]\!]^{II} \mid p \in F, a \in \Sigma\} \cup \{\bot_R, \bot_L\} \\
F'_{\mathrm{di}} &= \{[\![p, a, q]\!]^{II}, [\![p, a]\!]^{II} \mid p \in F, a \in \Sigma\} \cup \{\bot_R\}
\end{aligned}
$$

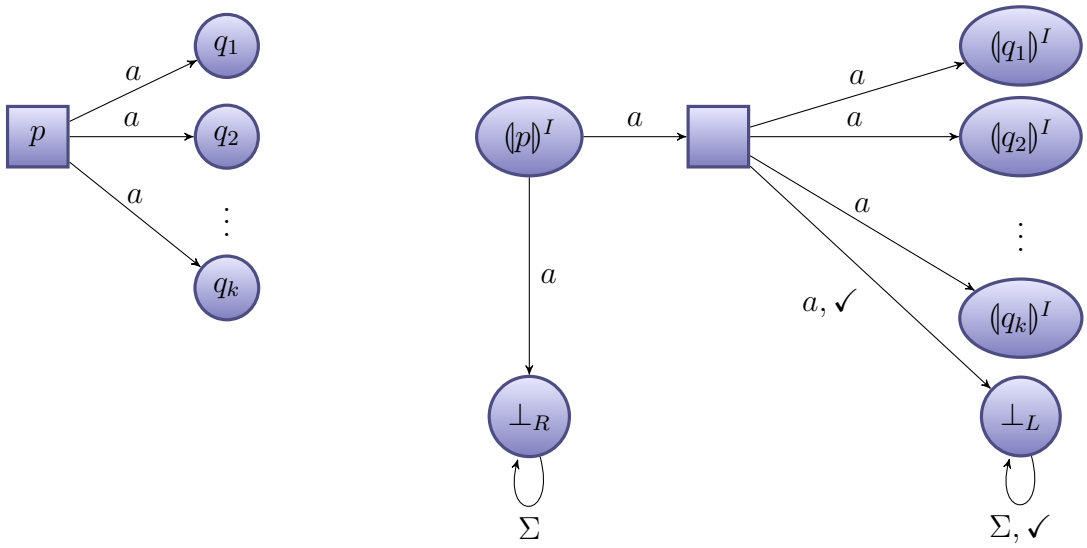
$$
\begin{aligned}
\Delta' = \{&((\!|p|\!)^I, a, [\![p, a, q]\!]^{II}), \\
&([\![p, a, q]\!]^{II}, a, (\!|q|\!)^I), \\
&((\!|p|\!)^I, a, \bot_R), \\
&([\![p, a, q]\!]^{II}, a, \bot_L), ([\![p, a, q]\!]^{II}, \checkmark, \bot_L) \mid p \in E \textit{ and } (p, a, q) \in \Delta\} \cup
\end{aligned}
$$

$$
\begin{aligned}
&\{((\!|p|\!)^I, a, [\![p, a]\!]^{II}), \\
&([\![p, a]\!]^{II}, a, (\!|q|\!)^I), \\
&((\!|p|\!)^I, a, \bot_R), \\
&([\![p, a]\!]^{II}, a, \bot_L), ([\![p, a]\!]^{II}, \checkmark, \bot_L) \mid p \in U \textit{ and } (p, a, q) \in \Delta\} \cup
\end{aligned}
$$

$$
\{(\bot_R, a, \bot_R), (\bot_L, a, \bot_L) \mid a \in \Sigma\} \ \cup \ \{(\bot_L, \checkmark, \bot_L)\}
$$



(a) Existential transitions



(b) Universal transitions

Figure 5.2: Translation to uniform automata and games

5.5.2 Correctness of the translation

The translation to uniform automata does not preserve the language—in fact, the uniformized automaton accepts Σ^ω for $x = \mathrm{di}$, and even more in the other cases. Nonetheless, the translation preserves the winner of simulation games: In Lemma 5.5.5, we show that winning strategies for Duplicator can be "ported" from the original simulation game to its uniform version. In Lemma 5.5.6, we show the reverse direction. Together, they prove the Bridge Lemma 5.5.2.

Lemma 5.5.5. *For two ABAs $\mathcal{Q}$ and $\mathcal{S}$, let $n_2 = |S|$. For $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$ and $k_1, k_2 > 0$, if σ is a winning strategy for Duplicator in the (k_1, k_2) x-simulation game $\mathbb{G}^x_{(k_1,k_2)}(\mathcal{Q}, \mathcal{S})$, then there exists a winning strategy $\mathcal{U}^x(\sigma)$ for Duplicator in the uniform (k_1+1, n_2+1) x-simulation game $\mathbb{G}^x_{(k_1+1,n_2+1)}(\mathcal{U}^x(\mathcal{Q}), \mathcal{U}^x(\mathcal{S}))$.*

One more pebble is needed in the uniform game on the left hand side since pebbles may be on the sink state. We also require $n_2 + 1$ pebbles on the right hand side, which allows us to use a simplified winning strategy for Duplicator in the uniform game.

Proof. Let $\mathcal{G} = \mathbb{G}^x_{(k_1,k_2)}(\mathcal{Q}, \mathcal{S})$ be the original game, and let $\mathcal{G}'$ be the associated uniform game, where $\mathcal{G}' = \mathbb{G}^x_{(k_1+1,n_2+1)}(\mathcal{U}^x(\mathcal{Q}), \mathcal{U}^x(\mathcal{S}))$. Given a winning strategy σ for Duplicator in $\mathcal{G}$, we define a winning strategy $\mathcal{U}^x(\sigma)$ for Duplicator in $\mathcal{G}'$. Each round in $\mathcal{G}$ is simulated by two rounds in $\mathcal{G}'$. Moreover, since $\mathcal{G}'$ is a uniform simulation game, every stage I round, where all pebbles are existential, is followed by a stage II round, where all pebbles are universal. When left pebbles in $\mathcal{G}$ are discarded, then at stage II of the current round they are moved to the left sink state $\perp_L$. When right pebbles are discarded, they are moved to $\perp_R$ at stage I of *the next round*.

Intuitively, pebbles are synchronized between $\mathcal{G}$ and $\mathcal{G}'$ as follows.

- At stage I, Spoiler moves left existential pebbles, those of which represent existential pebbles in $\mathcal{G}$ do induce a corresponding move by Spoiler in the latter game, and the others just go to the unique non-sink successor.

- Duplicator's reply at stage I is to just move right pebbles to every non-sink successor (that's why we need n_2 pebbles on the right in $\mathcal{G}'$), except for pebbles in $\mathcal{G}'$ which do not represent any pebble in $\mathcal{G}$: those are pebbles which where discarded in the previous round, and now they are moved to the right sink state $\perp_R$. This defines $\mathcal{U}^x(\sigma)$ on stage I rounds, and the game $\mathcal{G}'$ goes to stage II.

- At stage II, Spoiler moves right universal pebbles: Those representing universal states in $\mathcal{G}$ induce a corresponding move there, and the others just go to the unique non-sink successor.
- Spoiler mimics in $\mathcal{G}$ stages I and II of $\mathcal{G}'$.
- Duplicator replies in $\mathcal{G}$ according to σ.
- Duplicator's reply at stage II it so move to $\perp_L$ those left pebbles whose corresponding pebbles in $\mathcal{G}$ where thrown away by σ in the previous step. The remaining pebbles are moved to the corresponding successors, as dictated by σ. This defines $\mathcal{U}^x(\sigma)$ on stage II rounds.

We proceed with a formal exposition. Without no loss of generality, we assume that there initially is one pebble on a sink state on either side. Let π_k be the current σ-conform partial play in $\mathcal{G}$, and let π'_k be the current $\mathcal{U}^x(\sigma)$-conform partial play in $\mathcal{G}'$, where

$$\pi_k = \langle \mathbf{q}_0, \mathbf{s}_0 \rangle \overset{a_0}{\Longrightarrow} \langle \mathbf{q}_1, \mathbf{s}_1 \rangle \overset{a_1}{\Longrightarrow} \cdots \overset{a_{k-1}}{\Longrightarrow} \langle \mathbf{q}_k, \mathbf{s}_k \rangle$$
$$\pi'_k = \langle \mathbf{q}_0^I, \mathbf{s}_0^I \rangle \overset{a_0}{\Longrightarrow} \langle \mathbf{q}_0^{II}, \mathbf{s}_0^{II} \rangle \overset{a_0}{\Longrightarrow} \langle \mathbf{q}_1^I, \mathbf{s}_1^I \rangle \overset{a_1}{\Longrightarrow} \langle \mathbf{q}_1^{II}, \mathbf{s}_1^{II} \rangle \overset{a_1}{\Longrightarrow} \cdots \overset{a_{k-1}}{\Longrightarrow} \langle \mathbf{q}_k^I, \mathbf{s}_k^I \rangle$$

with $\mathbf{q}_0 = \{q_I\}$, $\mathbf{s}_0 = \{s_I\}$, $\mathbf{q}_0^I = \{(\!|q_I|\!)^I, \perp_R\}$ and $\mathbf{s}_0^I = \{(\!|s_I|\!)^I, \perp_L\}$. We maintain the following invariant:

$$\mathbf{q}_k^I = (\!|\mathbf{q}_k|\!)^I \cup \{\perp_L\} \quad \text{and} \quad (\!|\mathbf{s}_k|\!)^I \cup \{\perp_R\} \subseteq \mathbf{s}_k^I$$

The extra pebbles in $\mathbf{s}_k^I \setminus (\!|\mathbf{s}_k|\!)^I$ and not in $\perp_R$ represent those that where discarded in the previous round, and they are meant to be moved by Duplicator to $\perp_R$ at stage I of the current round.

The invariant immediately implies that, if σ is winning for Duplicator in $\mathcal{G}$, then σ' is winning for Duplicator in $\mathcal{G}'$: Indeed, whenever a pebble is accepting in $\mathcal{G}$, the corresponding stage I pebble in $\mathcal{G}'$ has only non-sink stage II accepting successors. Vice versa, the only non-sink accepting pebbles in $\mathcal{G}'$ are those at stage II which have a stage I predecessor pebble whose corresponding one in $\mathcal{G}$ is accepting. Therefore, the accepting behavior of pebbles in $\mathcal{G}$ at any round is matched exactly in $\mathcal{G}'$ *at stage II* of the same round.

First, assume Spoiler selects a_k as the next input symbol. We represent Spoiler's movement of pebbles in $\mathcal{G}'$ as a function g' mapping each pebble to his successor; the corresponding move in $\mathcal{G}$ is indicated by function g. By the previous discussions, we

assume, with no loss of generality, that Spoiler does not put any left pebble in $\bot_R$. At stage I, all pebbles are existential in $\mathcal{G}'$. For each left pebble $(\!|q|\!)^I \in (\!|\mathbf{q}_k^E|\!)^I$ which corresponds to an existential pebble $q \in E$ in $\mathcal{G}$, assume Spoiler selects a successor $g'((\!|q|\!)^I) := [\![q, a_k, q']\!]^{II}$ s.t. $q \xrightarrow{a_k} q'$. The corresponding pebble $q \in \mathbf{q}_k^E$ in $\mathcal{G}$ is moved to $g(q) := q'$. For each left pebble $q \in (\!|\mathbf{q}_k^U|\!)^I$ which corresponds to a universal pebble $q \in U$ in $\mathcal{G}$, with no loss of generality, we assume that Spoiler selects the unique non-sink successor $g'((\!|q|\!)^I) := [\![q, a_k]\!]^{II}$. And the pebble on $\bot_L$ stays there, i.e., $g'(\bot_L) := \bot_L$. By the invariant, there is no other pebble. Let $\mathbf{q}^{II} := g'(\mathbf{q}_k^I)$. Spoiler plays action

$$(\mathbf{q}_k^I, \mathbf{s}_k^I, a_k, \mathbf{q}^{II}, \emptyset) \in \Gamma_{\mathcal{G}'}^{\mathrm{Sp}}$$

Let $\mathbf{s}_E^I := (\!|\mathbf{s}_k^E|\!)^I$ be the set of right pebbles in $\mathcal{G}'$ which represent existential pebbles in $\mathcal{G}$, let $\mathbf{s}_U^I := (\!|\mathbf{s}_k^U|\!)^I$ be the universal ones, and let $\mathbf{s}_{\text{sink}}^I := \mathbf{s}_k^I \setminus (\mathbf{s}_E^I \cup \mathbf{s}_U^I)$ be the remaining pebbles. Duplicator's move at stage I is independent of σ:

1. Duplicator moves pebbles in $\mathbf{s}_E^I$ to every successor:

$$\mathbf{s}_E^{II} := \Delta'(\mathbf{s}_E^I, a_k) = \{[\![s, a_k, s']\!]^{II} \mid (\!|s|\!)^I \in \mathbf{s}_E^I, s \xrightarrow{a_k} s'\}$$

2. Duplicator move pebbles in $\mathbf{s}_U^I$ to the unique non-sink successor:

$$\mathbf{s}_U^{II} := \{[\![s, a_k]\!]^{II} \mid (\!|s|\!)^I \in \mathbf{s}_U^I\}$$

3. Duplicator move pebbles in $\mathbf{s}_{\text{sink}}^I$ to the sink state $\bot_R$.

We have $\mathbf{s}_E^I \cup \mathbf{s}_U^I = (\!|\mathbf{s}_k|\!)^I$. Define $\mathbf{s}^{II} := \mathbf{s}_E^{II} \cup \mathbf{s}_U^{II} \cup \{\bot_R\}$, and let

$$\mathcal{U}^x(\sigma)(\pi'_k, (\mathbf{q}_k^I, \mathbf{s}_k^I, a_k, \mathbf{q}^{II}, \emptyset)) := \langle \mathbf{q}^{II}, \mathbf{s}^{II} \rangle$$

which brings the uniform game to stage II, where Spoiler has to play again.

Clearly, Spoiler has to play action a_k again, since otherwise all left pebbles would be stuck. Moreover, we assume, with no loss of generality, that Spoiler does not put any right pebble in $\bot_L$. At stage II, all pebbles are universal in $\mathcal{G}'$. For each pebble $[\![s, a_k, s']\!]^{II} \in \mathbf{s}_E^{II}$ which corresponds to an existential pebble $s \in E$ in $\mathcal{G}$, Spoiler is forced to select the unique non-sink successor $g'([\![s, a_k, s']\!]^{II}) := (\!|s'|\!)^I$. For each pebble $[\![s, a_k]\!]^{II} \in \mathbf{s}_U^{II}$ which corresponds to a universal pebble $s \in U$ in $\mathcal{G}$, Spoiler selects a successor $g'([\![s, a_k]\!]^{II}) := (\!|s'|\!)^I$ s.t. $s \xrightarrow{a_k} s'$. Spoiler moves the corresponding

pebble $s \in \mathbf{s}_k^U$ in $\mathcal{G}$ to $g(s) := s'$. The pebble on $\perp_R$ stays there: $g'(\perp_R) = \perp_R$. Let $\mathbf{s}_{k+1}^I := g'(\mathbf{s}^{II})$, and at stage II Spoiler plays

$$(\mathbf{q}^{II}, \mathbf{s}^{II}, a_k, \emptyset, \mathbf{s}_{k+1}^I) \in \Gamma_{\mathcal{G}'}^{\mathrm{Sp}}$$

We now have enough information to define Spoiler's move in $\mathcal{G}$. Let $\mathbf{q}' = g(\mathbf{q}_k^E)$ and $\mathbf{s}' = g(\mathbf{s}_k^U)$ as defined by the two consecutive Spoiler's moves in $\mathcal{G}'$ above. Then, Spoiler in $\mathcal{G}$ plays

$$(\mathbf{q}_k, \mathbf{s}_k, a_k, \mathbf{q}', \mathbf{s}') \in \Gamma_{\mathcal{G}}^{\mathrm{Sp}}$$

We apply Duplicator's winning strategy σ in $\mathcal{G}$, to obtain

$$\sigma(\pi_k, (\mathbf{q}_k, \mathbf{s}_k, a_k, \mathbf{q}', \mathbf{s}')) = \langle \mathbf{q}_{k+1}, \mathbf{s}_{k+1} \rangle$$

This directly induces the long sought move for Duplicator in $\mathcal{G}'$ at round II. Let $\mathbf{q}_{k+1}^I := (\!|\mathbf{q}_{k+1}|\!)^I \cup \{\perp_L\}$. Let $\pi'' = \pi'_k \xRightarrow{a_k} \langle \mathbf{q}^{II}, \mathbf{s}^{II} \rangle$ and define $\mathcal{U}^x(\sigma)$ as

$$\mathcal{U}^x(\sigma)(\pi'', (\mathbf{q}^{II}, \mathbf{s}^{II}, a_k, \emptyset, \mathbf{s}_{k+1}^I)) := \langle \mathbf{q}_{k+1}^I, \mathbf{s}_{k+1}^I \rangle$$

No right pebble is thrown away, and, at this point, right pebbles are moved independently of $\mathbf{s}_{k+1}$ above. The invariant holds: On the right, it suffices to note that Duplicator at stage I moved pebbles to every non-sink successor. On the left, it holds by the very definition of $\mathbf{q}_{k+1}^I$. □

Lemma 5.5.6. *Let $\mathcal{Q}$ and $\mathcal{S}$ be two ABAs, and let $k_1, k_2 > 0$. For $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$, if σ is a winning strategy for Duplicator in the uniform (k_1, k_2) x-simulation game $\mathbb{G}^x_{(k_1,k_2)}(\mathcal{U}^x(\mathcal{Q}), \mathcal{U}^x(\mathcal{S}))$, then there exists a winning strategy σ' for Duplicator in the (k_1, k_2) x-game $\mathbb{G}^x_{(k_1,k_2)}(\mathcal{Q}, \mathcal{S})$.*

Proof. Let $\mathcal{G} = \mathbb{G}^x_{(k_1,k_2)}(\mathcal{U}^x(\mathcal{Q}), \mathcal{U}^x(\mathcal{S}))$ and $\mathcal{G}' = \mathbb{G}^x_{(k_1,k_2)}(\mathcal{Q}, \mathcal{S})$. Given a winning strategy σ for Duplicator in $\mathcal{G}$, we define a winning strategy σ' for Duplicator in $\mathcal{G}'$. Without no loss of generality, we assume that σ never moves pebbles to sink states, since throwing pebbles away has, in fact, the same effect; in this way, we save one pebble on each side of $\mathcal{G}'$. Two rounds in $\mathcal{G}$ (stage I + stage II) correspond to a single round in $\mathcal{G}'$. Pebbles are synchronized between $\mathcal{G}$ and $\mathcal{G}'$ in the obvious way. In particular, when pebbles in $\mathcal{G}$ are discarded, the corresponding pebbles in $\mathcal{G}'$ are discarded as well.

Formally, let π_k be the current σ-conform partial play in $\mathcal{G}$, and let π'_k be the current σ'-conform partial play in $\mathcal{G}'$, where

$$\pi_k = \langle \mathbf{q}_0^I, \mathbf{s}_0^I \rangle \xRightarrow{a_0} \langle \mathbf{q}_0^{II}, \mathbf{s}_0^{II} \rangle \xRightarrow{a_0} \langle \mathbf{q}_1^I, \mathbf{s}_1^I \rangle \xRightarrow{a_1} \langle \mathbf{q}_1^{II}, \mathbf{s}_1^{II} \rangle \xRightarrow{a_1} \cdots \xRightarrow{a_{k-1}} \langle \mathbf{q}_k^I, \mathbf{s}_k^I \rangle$$
$$\pi'_k = \langle \mathbf{q}_0, \mathbf{s}_0 \rangle \xRightarrow{a_0} \langle \mathbf{q}_1, \mathbf{s}_1 \rangle \xRightarrow{a_1} \cdots \xRightarrow{a_{k-1}} \langle \mathbf{q}_k, \mathbf{s}_k \rangle$$

with $\mathbf{q}_0^I = \{(\!|q_I|\!)^I\}$, $\mathbf{s}_0^I = \{(\!|s_I|\!)^I\}$, $\mathbf{q}_0 = \{q_I\}$ and $\mathbf{s}_0 = \{s_I\}$. We maintain the following invariant: At round k,

$$\mathbf{q}_k^I = (\!|\mathbf{q}_k|\!)^I \quad \text{and} \quad \mathbf{s}_k^I = (\!|\mathbf{s}_k|\!)^I$$

Assume Spoiler in $\mathcal{G}'$ selects a_k as the next input symbol. We represent Spoiler's movement of pebbles in $\mathcal{G}'$ as a function g' mapping each pebble to his successor; the corresponding move in $\mathcal{G}$ is indicated by function g.

- For each existential left pebble $q \in \mathbf{q}_k^E$, assume Spoiler selects a successor $g'(q) := q'$ s.t. $q \xrightarrow{a_k} q'$.

  The corresponding pebble $(\!|q|\!)^I \in \mathbf{q}_k^I$ in $\mathcal{G}$ is moved at stage I to $g((\!|q|\!)^I) := [\![q, a_k, q']\!]^{II}$, and then at stage II to $(\!|q'|\!)^I$.

- For each universal right pebble $s \in \mathbf{s}_k^U$, assume Spoiler selects a successor $g'(s) := s'$ s.t. $s \xrightarrow{a_k} s'$.

  The corresponding pebble $(\!|s|\!)^I \in \mathbf{s}_k^I$ in $\mathcal{G}$ is moved at stage I to $[\![s, a_k]\!]^{II}$, and then at stage II to $g([\![s, a_k]\!]^{II}) := (\!|s'|\!)^I$.

Summing up, Spoiler in $\mathcal{G}'$ plays action

$$(\mathbf{q}_k, \mathbf{s}_k, a_k, \mathbf{q}', \mathbf{s}') \in \Gamma^{\mathrm{Sp}}_{\mathcal{G}'}, \qquad \text{where } \mathbf{q}' := g'(\mathbf{q}_k^E) \text{ and } \mathbf{s}' := g'(\mathbf{s}_k^U)$$

This induces a move for Spoiler in $\mathcal{G}$, as we have outlined above.

- For each pebble $(\!|q|\!)^I \in (\!|\mathbf{q}_k^E|\!)^I$, Spoiler selects the non-sink successor $g((\!|q|\!)^I)$.

- For each pebble $(\!|q|\!)^I \in (\!|\mathbf{q}_k^U|\!)^I$, Spoiler is forced to select the unique non-sink successor $[\![q, a_k]\!]^{II}$. In this case, we extend g by $g((\!|q|\!)^I) := [\![q, a_k]\!]^{II}$.

Therefore, Spoiler in $\mathcal{G}$ at stage I plays as

$$(\mathbf{q}_k^I, \mathbf{s}_k^I, a_k, \mathbf{q}_k^{II}, \emptyset) \in \Gamma^{\mathrm{Sp}}_{\mathcal{G}}, \qquad \text{where } \mathbf{q}^{II} = g(\mathbf{q}_k^I)$$

We assume that Duplicator in $\mathcal{G}$ never throws pebbles away at stage I. This is safe, since the only reason for throwing pebbles away is to satisfy the acceptance condition. Indeed, if a pebble is thrown away at stage I for such a reason, then the same pebble could have been thrown away at stage II of the *previous* round. This holds since accepting states at stage II are completely determined by states at stage I,

and Duplicator can decide in advance whether a pebble should be thrown away. Thus, assume Duplicator replies with

$$\sigma(\pi_k, (\mathbf{q}_k^I, \mathbf{s}_k^I, a_k, \mathbf{q}_k^{II}, \emptyset)) = \langle \mathbf{q}_k^{II}, \mathbf{s}_k^{II} \rangle$$

which brings $\mathcal{G}$ to stage II, where Spoiler has to play again. Let

$$\mathbf{s}_E^{II} = \{[\![s, a_k, s']\!]^{II} \mid [\![s, a_k, s']\!]^{II} \in \mathbf{s}_k^{II}\}$$
$$\mathbf{s}_U^{II} = \{[\![s, a_k]\!]^{II} \mid [\![s, a_k]\!]^{II} \in \mathbf{s}_k^{II}\}$$

Since we have assumed that no pebble is ever moved to a sink state by σ, $\mathbf{s}_k^{II} = \mathbf{s}_E^{II} \cup \mathbf{s}_U^{II}$, which is nonempty for σ is winning. The move of Spoiler at stage II of $\mathcal{G}$ is determined as follows.

- For each pebble $[\![s, a_k]\!]^{II} \in \mathbf{s}_U^{II}$, Spoiler selects the non-sink successor $g([\![s, a_k]\!]^{II})$, as defined above.
- For each pebble $[\![s, a_k, s']\!]^{II} \in \mathbf{s}_E^{II}$, Spoiler is forced to select the unique non-sink successor $(\!|s'|\!)^I$. In this case, we extend g by $g([\![s, a_k, s']\!]^{II}) := (\!|s'|\!)^I$.

Therefore, Spoiler plays

$$(\mathbf{q}_k^{II}, \mathbf{s}_k^{II}, a_k, \emptyset, \mathbf{s}^{III}) \in \Gamma_{\mathcal{G}}^{\mathrm{Sp}}, \qquad \text{where } \mathbf{s}^{III} = g(\mathbf{s}_k^{II})$$

and Duplicator replies with

$$\sigma(\pi_k \langle \mathbf{q}_k^{II}, \mathbf{s}_k^{II} \rangle, (\mathbf{q}_k^{II}, \mathbf{s}_k^{II}, a_k, \emptyset, \mathbf{s}^{III}) = \langle \mathbf{q}_{k+1}^I, \mathbf{s}_{k+1}^I \rangle$$

Notice that pebbles are allowed to be thrown away at stage II in $\mathcal{G}$.

This induces the long sought move for Duplicator in $\mathcal{G}'$. Let $\mathbf{q}_{k+1} = \{q' \mid (\!|q'|\!)^I \in \mathbf{q}_{k+1}^I\}$, $\mathbf{s}_{k+1} = \{s' \mid (\!|s'|\!)^I \in \mathbf{s}_{k+1}^I\}$, and define σ' as

$$\sigma'(\pi', (\mathbf{q}_k, \mathbf{s}_k, a_k, \mathbf{q}', \mathbf{s}')) := \langle \mathbf{q}_{k+1}, \mathbf{s}_{k+1} \rangle$$

The invariant holds by the very definition of $\mathbf{q}_{k+1}$ and $\mathbf{s}_{k+1}$.

We now argue about the winning condition. The fundamental observation is that accepting pebbles in $\mathcal{G}'$ are in one-to-one correspondence with accepting pebbles at stage II in $\mathcal{G}$. This holds since, if a pebble is accepting in $\mathcal{G}'$, then the corresponding stage I pebble in $\mathcal{G}$ has only accepting successors. Since, by assumption, no pebble is thrown away at stage I, an accepting pebble $\mathcal{G}'$ will induce exactly one accepting pebble at stage II in $\mathcal{G}$. And this is the only way in which pebbles can be accepting in $\mathcal{G}$. Therefore, the acceptance condition in $\mathcal{G}'$ is matched exactly (modulo 1-step) in $\mathcal{G}$. Since Duplicator wins $\mathcal{G}$, so does $\mathcal{G}'$. □

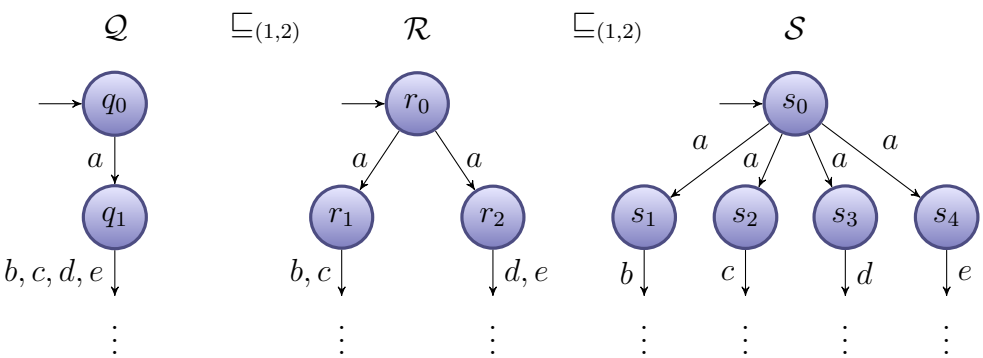
5.6 Transitivity

In this section, we establish that $(1, n)$, $(n, 1)$ and (n, n) simulations are transitive, where n is the number of states of the automaton. Transitivity is crucially used in the proofs for quotienting (cf. Section 5.8) and for the subset constructions (cf. Section 5.7). Thus, for this section, we have the following main result.

Theorem 5.6.1. *Let $\mathcal{Q}$ be an ABA with n states, and let $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$. Then, $\sqsubseteq^x_{(1,n)}$, $\sqsubseteq^x_{(n,1)}$ and $\sqsubseteq^x_{(n,n)}$ are transitive.*

While the limit multipebble simulations in the above theorem are transitive, in general (k_1, k_2)-simulation is not transitive, and this holds already for NBAs (cf. [42]).

Example 5.6.1 - $(1, k)$-simulation with $1 < k < n$ is not transitive



Consider the automata $\mathcal{Q}$, $\mathcal{R}$ and $\mathcal{S}$ above. With two pebbles, $\mathcal{R}$ can simulate $\mathcal{Q}$ and $\mathcal{S}$ can simulate $\mathcal{R}$. However, $\mathcal{S}$ cannot $(1, 2)$-simulate $\mathcal{Q}$:

$$\mathcal{Q} \not\sqsubseteq_{(1,2)} \mathcal{S}$$

Intuitively, the pebble in $\mathcal{Q}$ is simulated by two pebbles in $\mathcal{R}$, *each of which* is simulated by *two* pebbles in $\mathcal{S}$. Therefore, four pebbles in $\mathcal{S}$ suffice to simulate $\mathcal{Q}$, but not less.

We do not directly prove transitivity for arbitrary automata. Instead, we do it for uniform automata. In this way, we can exploit the special properties of the induced uniform simulation games (cf. Section 5.5). This makes the proofs easier; in the (n, n) case, it is not even clear how to define certain fundamental notions in the general case of non-uniform games.

Transitivity is translated back to arbitrary automata by using the Bridge Lemma 5.5.2: For example, assume $q \sqsubseteq^{\mathrm{f}}_{(n,n)} r \sqsubseteq^{\mathrm{f}}_{(n,n)} s$. By taking Lemma 5.5.2 forth, $\mathcal{U}(q) \sqsubseteq^{\mathrm{f}}_{(n,n)}$

$\mathcal{U}(r) \sqsubseteq^{\mathrm{f}}_{(n,n)} \mathcal{U}(s)$. Since simulation is transitive on uniform automata, $\mathcal{U}(q) \sqsubseteq^{\mathrm{f}}_{(n,n)} \mathcal{U}(s)$. By taking Lemma 5.5.2 back, $q \sqsubseteq^{\mathrm{f}}_{(n,n)} s$.

The rest of this section is organized as follows.

- In Section 5.6.1, we give an high level view of the proof idea. We describe composition of strategies and their properties.
- In Section 5.6.2, we prove transitivity of direct simulation for uniform games, by instantiating the framework from Section 5.6.1.
- In Section 5.6.3, we do the same for delayed and fair simulation.
- Finally, in Section 5.6.4, we note that the notion of delayed simulation from [29] is not transitive, and we explain why. The definition of delayed simulation in this thesis *is* transitive, as we prove.

5.6.1 Proof overview

The main idea is to define an associative *join* operation $\bowtie$ for Duplicator's strategies, and to show that $\bowtie$ preserves winning strategies. The approach is borrowed from [55].

All simulation games in this section are uniform (cf. Section 5.5). We consider (n, n)-simulation; the $(1, n)$- and $(n, 1)$-case arise as a special instance thereof, by restricting sets of pebbles to be singletons in the obvious way. Let $\mathbb{G}_L = \mathbb{G}_{(n,n)}(q, r)$ and $\mathbb{G}_R = \mathbb{G}_{(n,n)}(r, s)$ be the simulation games between q and r, and between r and s, respectively. $\mathbb{G}_L$ is the *left game* and $\mathbb{G}_R$ is the *right game*. Let σ_L (left strategy) and σ_R (right strategy) be two Duplicator's strategies in $\mathbb{G}_L$ and $\mathbb{G}_R$, respectively. We construct a *joint strategy* $\sigma_L \bowtie \sigma_R$ for Duplicator in the simulation game $\mathbb{G} = \mathbb{G}_{(n,n)}(q, s)$. $\mathbb{G}$ is the *outer game*. The crucial part of the following sections is to properly define $\sigma_L \bowtie \sigma_R$. The main result is that composition preserves winning strategies:

Lemma 5.6.2 (Preservation of winning strategies)**.** *If σ_L is a left winning strategy in $\mathbb{G}_L$ and σ_R is a right winning strategy in $\mathbb{G}_R$, then $\sigma_L \bowtie \sigma_R$ is winning in $\mathbb{G}$.*

This immediately implies that $\sqsubseteq_{(n,n)}$ is transitive: If $q \sqsubseteq_{(n,n)} r \sqsubseteq_{(n,n)} s$, then there exist two winning strategies σ_L and σ_R as above. By the lemma, $\sigma_L \bowtie \sigma_R$ is a winning strategy in $\mathbb{G}$. Thus, $q \sqsubseteq_{(n,n)} s$.

The definition of $\sigma_L \bowtie \sigma_R$ follows the same general pattern for all simulations. While playing the $\mathbb{G}$ game, we keep track of several left and right games. Duplicator's choices dictated by σ_R and σ_L in the right and left games will define the joint strategy

$\sigma_L \bowtie \sigma_R$ in $\mathbb{G}$. Since games are uniform, at each round pebbles are either all existential or all universal. Accordingly, we call a round existential/universal and, depending on which is the case, we have the following flow of information between the various games:

$$\begin{array}{rl} \text{Existential round:} & \mathbb{G} \xrightarrow{S-S} \mathbb{G}_L \xrightarrow{D-S} \mathbb{G}_R \xrightarrow{D-D} \mathbb{G} \\ \text{Universal round:} & \mathbb{G} \xrightarrow{S-S} \mathbb{G}_R \xrightarrow{D-S} \mathbb{G}_L \xrightarrow{D-D} \mathbb{G} \end{array}$$

The first row above means that, in an existential round, the flow of information goes left-to-right between games: $\mathbb{G} \xrightarrow{S-S} \mathbb{G}_L$ means that Spoiler's choice in $\mathbb{G}$ induces a corresponding move in left games. Duplicator's reply in those games is obtained by applying σ^L. In turn, this is interpreted as Spoiler's move in right games, $\mathbb{G}_L \xrightarrow{D-S} \mathbb{G}_R$. Finally, Duplicator's reply in those games is obtained by applying σ^R. This also defines a move for σ in $\mathbb{G}$: $\mathbb{G}_R \xrightarrow{D-D} \mathbb{G}$. The second row is symmetric, with the flow of information going right-to-left.

Remark 5.6.3. In non-uniform games, the flow of information would not be as simple as above (either from left to right, or from right to left). Indeed, in non-uniform games there might be pebbles on both existential and universal states at a given round. This creates problems since pebbles in the middle would have circular dependencies: For example, existential pebbles in the middle depend on the move of Duplicator in left games, which depends on the move of Spoiler, which needs to move universal pebbles in the middle, which, in turn, depend on the move of Duplicator in right games, which depends on the move of Spoiler, which needs to move existential pebbles in the middle, thus forming a loop. In non-uniform games, it is not clear how to break such dependencies. This technical difficulty prevented us from proving transitivity for (n, n)-simulations in [29].

The necessary bookkeeping for coordinating the various games and for carrying the required invariant is done with a *logbook*. We introduce some notation which is heavily used in the following sections. At round $k \geq 0$, the current logbook is a tuple $B_k = (B_k^L, B_k^R, w_k, \pi_k)$, where

- $w_k = a_0 a_1 \ldots a_{k-1}$ is the input word constructed so far by Spoiler, with $w_0 = \varepsilon$ for $k = 0$.
- π_k is the partial play of length $k+1$ of the outer game $\mathbb{G}$ between q and s, where

$$\begin{aligned} \pi_k &= \rho_k^L \times \rho_k^R, \text{ with} \\ \rho_k^L &= \mathbf{q}_0 \xRightarrow{a_0} \mathbf{q}_1 \xRightarrow{a_1} \cdots \xRightarrow{a_{k-1}} \mathbf{q}_k \\ \rho_k^R &= \mathbf{s}_0 \xRightarrow{a_0} \mathbf{s}_1 \xRightarrow{a_1} \cdots \xRightarrow{a_{k-1}} \mathbf{s}_k \ , \end{aligned}$$

- The *left logbook* $B_k^L = \{\pi_{k,0}^L, \pi_{k,1}^L, \ldots, \pi_{k,l(k)}^L\}$ is a set of σ_L-conform partial plays in $\mathbb{G}_L$, where each play $\pi_{k,i}^L \in B_k^L$ has length $k+1$ and is of the form

$$\begin{aligned}
\pi_{k,i}^L &= \rho_{k,i}^L \times \psi_{k,i}^L, \text{ with} \\
\rho_{k,i}^L &= \mathbf{q}_{k,i,0} \overset{a_0}{\Longrightarrow} \mathbf{q}_{k,i,1} \overset{a_1}{\Longrightarrow} \cdots \overset{a_{k-1}}{\Longrightarrow} \mathbf{q}_{k,i,k} \\
\psi_{k,i}^L &= \mathbf{r}_{k,i,0}^L \overset{a_0}{\Longrightarrow} \mathbf{r}_{k,i,1}^L \overset{a_1}{\Longrightarrow} \cdots \overset{a_{k-1}}{\Longrightarrow} \mathbf{r}_{k,i,k}^L
\end{aligned}$$

  $\rho_{k,i}^L$ and $\psi_{k,i}^L$ are the projections of $\pi_{k,i}^L$ to the first and second component, respectively.

- Similarly, the *right logbook* $B_k^R = \{\pi_{k,0}^R, \pi_{k,1}^R, \ldots, \pi_{k,r(k)}^R\}$ is a set of σ_R-conform partial plays in $\mathbb{G}_R$ of length $k+1$, of the form

$$\begin{aligned}
\pi_{k,j}^R &= \psi_{k,j}^R \times \rho_{k,j}^R, \text{ with} \\
\psi_{k,j}^R &= \mathbf{r}_{k,j,0}^R \overset{a_0}{\Longrightarrow} \mathbf{r}_{k,j,1}^R \overset{a_1}{\Longrightarrow} \cdots \overset{a_{k-1}}{\Longrightarrow} \mathbf{r}_{k,j,k}^R \\
\rho_{k,j}^R &= \mathbf{s}_{k,j,0} \overset{a_0}{\Longrightarrow} \mathbf{s}_{k,j,1} \overset{a_1}{\Longrightarrow} \cdots \overset{a_{k-1}}{\Longrightarrow} \mathbf{s}_{k,j,k}
\end{aligned}$$

  $\psi_{k,j}^R$ and $\rho_{k,j}^R$ are the projections of $\pi_{k,j}^R$ to the first and second component, respectively.

Pebbles in $\mathbf{q}_k$ are called *left pebbles*, those in $\mathbf{s}_k$ are called *right pebbles*, and the remaining pebbles are called *middle pebbles*.

Not every logbook describes a legal interaction between $\mathbb{G}$, $\mathbb{G}_L$ and $\mathbb{G}_R$. We say that a logbook B_k is *valid* iff it satisfies certain *logbook properties*. These properties depend on the specific simulation game, and they will be instantiated in the following sections.

Main technical difficulties The main issue in properly combining left and right games is that middle pebbles are shared in non-trivial patterns between them. In $(1,1)$-simulation, there is just one left and one right game, and the standard argument is to interpret Duplicator's move in the left game as Spoiler's move in the right game (and vice versa). But now there are several left and several right games. Consequently, a number of difficulties arise. For example, in an existential round:

(D1) Pebbles might be dropped, thus making it difficult to keep track of intermediate games and to prove invariants.

(D2) For both delayed and fair simulation, we even have an acceptance condition mismatch between right and left games. Indeed, at some point one needs to

translate existentially fair paths in left games into universally fair paths in right games, and the former condition does not imply the latter in general.

(D3) The same middle pebble might be shared among several left games. For each left game, Duplicator might prescribe different successors to these pebbles. These conflicts have to be resolved.

Difficulties (D1) and (D2) are solved by considering only uniform games. Specifically, (D1) is solved by (U2), and (D2) is solved by (U4). Difficulty (D3) is handled differently between direct and delayed/fair simulation, as we remark in the construction.

5.6.2 Direct simulation

We consider (n, n)-direct simulation; the $(1, n)$- and $(n, 1)$-cases are just a special case thereof. We instantiate the framework of Section 5.6.1 to direct simulation. At round $k \geq 0$, the current logbook is a tuple $B_k = (B_k^L, B_k^R, w_k, \pi_k)$ (see Section 5.6.1), and it is *valid* iff it satisfies the logbook properties below.

$$\mathbf{q}_k = \bigcup_i \mathbf{q}_{k,i,k} \tag{DiLeft}$$

$$\mathbf{s}_k = \bigcup_j \mathbf{s}_{k,j,k} \tag{DiRight}$$

$$\forall (r \in \mathbf{r}^L_{k,i,k}) \cdot \exists j \cdot (r \in \mathbf{r}^R_{k,j,k}) \tag{DiMiddle$_\mathrm{L}$}$$

$$\forall (r \in \mathbf{r}^R_{k,j,k}) \cdot \exists i \cdot (r \in \mathbf{r}^L_{k,i,k}) \tag{DiMiddle$_\mathrm{R}$}$$

$$\forall (q \in \mathbf{q}_k, s \in \mathbf{s}_k) \cdot \exists (i, j) \cdot \begin{cases} (q \in \mathbf{q}_{k,i,k}) \wedge (s \in \mathbf{s}_{k,j,k}) \\ \wedge \\ (\mathbf{r}^L_{k,i,k} \cap \mathbf{r}^R_{k,j,k} \neq \emptyset) \end{cases} \tag{Middle}$$

Intuitively, property (DiLeft) states that left pebbles in left games cover left pebbles in $\mathbb{G}$, and dually for property (DiRight). These two properties holds by the very way sets $\mathbf{q}_k$ and $\mathbf{s}_k$ are constructed. Property (DiMiddle$_\mathrm{L}$) states that middle pebbles in left games are also (middle pebbles) in some right game, and dually for property (DiMiddle$_\mathrm{R}$). Taken together, these two properties state that middle pebbles are always shared between left and right games; i.e., no middle pebble exists on one side which does not belong to the other side as well. They are established inductively, and they are used to ensure that certain sets defined during the construction are non-empty. Finally, property (Middle) says that for any left and right pebble in $\mathbb{G}$, there exist a left and right game containing those two pebbles, respectively; moreover, these two games share a middle pebble. This

property is established inductively, and it is used to deal with the winning condition for direct simulation.

We inductively show how to build a valid logbook and simultaneously define the joint strategy $\sigma_0 \bowtie \sigma_1$. For the initial round $k = 0$, the initial configuration in $\mathbb{G}_0$ is $\langle q, r\rangle$, the one in $\mathbb{G}_1$ is $\langle r, s\rangle$, and the one in $\mathbb{G}$ is $\langle q, s\rangle$. Therefore, the initial logbook $B_0 := (\{\langle\{q\},\{r\}\rangle\}, \{\langle\{r\},\{s\}\rangle\}, \varepsilon, \langle\{q\},\{s\}\rangle)$ is clearly valid.

Inductively assume that, at round k, B_k is a valid logbook. We show how to build a new logbook $B_{k+1} = (B^L_{k+1}, B^R_{k+1}, w_{k+1}, \pi_{k+1})$ for the next round, and we show that it is valid. We distinguish two cases, based on whether we are in an existential or in a universal round.

Existential round Assume that Spoiler moves as follows:

$$(\mathbf{q}_k, \mathbf{s}_k, a_k, \mathbf{q}', \emptyset) \in \Gamma^{\mathrm{Sp}}_{\mathbb{G}} \ , \tag{S$_{\mathrm{di}}$}$$

i.e., the next input symbol is a_k, and (existential) left pebbles are moved from $\mathbf{q}_k$ to $\mathbf{q}'$. Notice that all pebbles are existential by assumption, so Spoiler does not move any pebble in $\mathbf{s}_k$, which explains the empty set in the last component. Let the input word for the next logbook be $w_{k+1} := w_k \cdot a_k$.

Move (S)$_{\mathrm{di}}$ above immediately induces a move by Spoiler in every left game: For any left partial play $\pi^L_{k,i}$ ending in configuration $\langle \mathbf{q}_{k,i,k}, \mathbf{r}^L_{k,i,k}\rangle$ with $\mathbf{q}_{k,i,k} \subseteq \mathbf{q}_k$, let $\mathbf{q}'_i = \mathbf{q}' \downarrow_{\Delta(\mathbf{q}_{k,i,k}, a_k)}$ be the choice of Spoiler (S)$_{\mathrm{di}}$ restricted to $\mathbf{q}_{k,i,k}$ (cf. Section 2.2). Accordingly, we let the left Spoiler in the i-th left game play

$$(\mathbf{q}_{k,i,k}, \mathbf{r}^L_{k,i,k}, a_k, \mathbf{q}'_i, \emptyset) \in \Gamma^{\mathrm{Sp}}_{\mathbb{G}_L} \ , \tag{S-L$_{\mathrm{di}}$}$$

and we then apply left Duplicator's strategy σ_L thereto, obtaining

$$\sigma_L(\pi^L_{k,i})(a_k, \mathbf{q}'_i, \emptyset) = \langle \mathbf{q}_{k+1,i,k+1}, \mathbf{r}^L_{k+1,i,k+1}\rangle \ . \tag{D-L$_{\mathrm{di}}$}$$

By definition, $\mathbf{r}^L_{k+1,i,k+1}$ is obtained by selecting a set of successors in $\Delta(r, a_k)$ for each $r \in \mathbf{r}^L_{k,i,k}$. Let $c_i(r)$ be this set of successors, and let $c_i(r) = \emptyset$ for $r \notin \mathbf{r}^L_{k,i,k}$. We call $c_i : Q \mapsto 2^Q$ a choice function; by definition, $\mathbf{r}^L_{k+1,i,k+1} = \bigcup_{r\in Q} c_i(r)$. Since the game is uniform, pebbles are not thrown away (U2), and $c_i(r) \neq \emptyset$ for every $r \in \mathbf{r}^L_{k,i,k}$. Difficulty (D3) is solved by considering all possible successors of a pebble over all left games it might belong to: We collapse all these choices c_i's into a unique function $c : Q \mapsto 2^Q$, by defining $c(r) = \bigcup_i c_i(r)$ for any $r \in Q$.

The next configuration of the i-th left game is $\langle \mathbf{q}_{k+1,i,k+1}, \mathbf{r}^L_{k+1,i,k+1} \rangle$, which induces the corresponding partial play at round $k+1$ $\pi^L_{k+1,i} = \pi^L_{k,i} \xRightarrow{a_k} \langle \mathbf{q}_{k+1,i,k+1}, \mathbf{r}^L_{k+1,i,k+1} \rangle$. Accordingly, the new left logbook B^L_{k+1} is defined as $B^L_{k+1} := \{\pi^L_{k+1,0}, \dots, \pi^L_{k+1,l(k)}\}$. The number of left games is preserved in an existential round.

We now propagate the choices of left Duplicator to the right. Initially, the new right logbook B^R_{k+1} is empty. Each right game spawns several new right games.

- Let $\langle \mathbf{r}^R_{k,j,k}, \mathbf{s}_{k,j,k} \rangle$ be the current configuration of the j-th right game. By property (DiMiddle$_\mathrm{R}$), each $r \in \mathbf{r}^R_{k,j,k}$ belongs to some $\mathbf{r}^L_{k,i,k}$. But c_i is non-empty on $\mathbf{r}^L_{k,i,k}$, therefore c is non-empty on $\mathbf{r}^R_{k,j,k}$.

  Each pebble r induces a global set of successors $c(r)$. Let $\mathcal{X}_j = c(\mathbf{r}^R_{k,j,k}) = \{c(r) \mid r \in \mathbf{r}^R_{k,j,k}\}$ be the set of all possible successors of $\mathbf{r}^R_{k,j,k}$, and let $Y_j = \otimes \mathcal{X}_j$ be the set of all possible combinations of successors. Each set in Y_j is obtained by fixing a representative for each set $c(r)$. By the discussion above, X_j does not contain the empty set, therefore $Y_j \neq \emptyset$.

- For every choice of successors $\mathbf{r}^R_{k+1,h,k+1} \in Y_j$, create a new right game:

  - Set $\mathbf{r}^R_{k+1,h,k+1}$ induces a move by Spoiler in the j-th right game:

    $$(\mathbf{r}^R_{k,j,k}, \mathbf{s}_{k,j,k}, a_k, \mathbf{r}^R_{k+1,h,k+1}, \emptyset) \in \Gamma^{\mathrm{Sp}}_{\mathbb{G}_R} \ . \qquad \text{(S-R)}_{\mathrm{di}}$$

    We apply Duplicator's winning strategy σ_R

    $$\sigma_R(\pi^R_{k,j})(a_k, \mathbf{r}^R_{k+1,h,k+1}, \emptyset) = \langle \mathbf{r}^R_{k+1,h,k+1}, \mathbf{s}_{k+1,h,k+1} \rangle \qquad \text{(D-R)}_{\mathrm{di}}$$

    Since the game is uniform, by (U2) Duplicator does not drop any middle pebble from $\mathbf{r}^R_{k+1,h,k+1}$.

  - Create a new partial play

    $$\pi^R_{k+1,h} = \pi^R_{k,j} \xRightarrow{a_k} \langle \mathbf{r}^R_{k+1,h,k+1}, \mathbf{s}_{k+1,h,k+1} \rangle$$

    and add it to the new right logbook B^R_{k+1}.

The number of right games is *not preserved* in an existential round, since each right game might create several new right games.

Let $\mathbf{s}_{k+1} = \bigcup_h \mathbf{s}_{k+1,h,k+1}$ (cf. (D-R)$_{\mathrm{di}}$) and $\mathbf{q}_{k+1} = \bigcup_i \mathbf{q}_{k+1,i,k+1}$ (cf. (D-L)$_{\mathrm{di}}$). We define Duplicator's move in $\mathbb{G}$ as

$$(\sigma_L \bowtie \sigma_R)(\pi_k)(a_k, \mathbf{q}', \emptyset) := \langle \mathbf{q}_{k+1}, \mathbf{s}_{k+1} \rangle \ . \qquad \text{(D)}_{\mathrm{di}}$$

Invariant Conditions (DiLeft) and (DiRight) are immediately satisfied by the definition of $\mathbf{q}_{k+1}$ and $\mathbf{s}_{k+1}$.

Condition (DiMiddle_L) is established inductively. Consider any i-th left game, and take any middle pebble $r' \in \mathbf{r}^L_{k+1,i,k+1}$. We have to show that it belongs to some right game as well, i.e., that there exists h s.t. $r \in \mathbf{r}^R_{k+1,h,k+1}$. By construction, there exists a predecessor $r \in \mathbf{r}^L_{k,i,k}$ s.t. $r' \in c_i(r) \subseteq c(r)$. By induction hypothesis, there exists j s.t. the predecessor r is in $\mathbf{r}^R_{k,j,k}$. Since $r' \in c(r)$ is a successor of r, and for any choice of successors a new right game is created, by construction there exists some h s.t. $r' \in \mathbf{r}^R_{k+1,h,k+1}$.

In an existential step, condition (DiMiddle_R) holds by construction. Take any middle pebble $r \in \mathbf{r}^R_{k+1,h,k+1}$ in any right game, and we have to show that it belongs to some left game as well, i.e., that there exists i s.t. $r \in \mathbf{r}^L_{k+1,i,k+1}$. By construction, there exists j s.t. $\mathbf{r}^R_{k+1,h,k+1} \in \bigotimes c(\mathbf{r}^R_{k,j,k})$. This implies that there exists $r' \in \mathbf{r}^R_{k,j,k}$ s.t. $r \in c(r')$, which is to say $r \in c_i(r')$ for some i. Consequently, by the definition of c_i, $r \in \mathbf{r}^L_{k+1,i,k+1}$.

Finally, the rectangularity condition (Middle) is established inductively. Take any $q \in \mathbf{q}_{k+1}$ and $s \in \mathbf{s}_{k+1}$. We have to find i, h s.t. $q \in \mathbf{q}_{k+1,i,k+1}$, $s \in \mathbf{s}_{k+1,h,k+1}$, and $\mathbf{r}^L_{k+1,i,k+1} \cap \mathbf{r}^R_{k+1,h,k+1} \neq \emptyset$. By construction, there exist $q' \in \mathbf{q}_k$ and $s' \in \mathbf{s}_k$ s.t. $q' \xrightarrow{a_k} q$ and $s' \xrightarrow{a_k} s$. By induction hypothesis, there exist i, j s.t. $q' \in \mathbf{q}_{k,i,k}$, $s' \in \mathbf{s}_{k,j,k}$, and $\mathbf{r}^L_{k,i,k} \cap \mathbf{r}^R_{k,j,k} \neq \emptyset$. So, take some $r \in \mathbf{r}^L_{k,i,k} \cap \mathbf{r}^R_{k,j,k}$, and consider the set $c_i(r)$. By definition, $\mathbf{r}^L_{k+1,i,k+1} = \bigcup c_i(\mathbf{r}^L_{k,i,k})$. Since $r \in \mathbf{r}^L_{k,i,k}$, we have $c_i(r) \subseteq \mathbf{r}^L_{k+1,i,k+1}$. Consider now the set $Y_j = \bigotimes c(\mathbf{r}^R_{k,j,k})$. Since $r \in \mathbf{r}^R_{k,j,k}$ and $c_i(r)$ is a non-empty subset of $c(r)$, by the definition of unordered product there exists h s.t. $\mathbf{r}^R_{k+1,h,k+1} \in Y_j$ and $\mathbf{r}^R_{k+1,h,k+1}$ selects some element in $c_i(r)$. That is, $\mathbf{r}^R_{k+1,h,k+1} \cap c_i(r) \neq \emptyset$. But $c_i(r) \subseteq \mathbf{r}^L_{k+1,i,k+1}$, therefore $\mathbf{r}^L_{k+1,i,k+1} \cap \mathbf{r}^R_{k+1,h,k+1} \neq \emptyset$.

Universal round Assume that Spoiler moves as follows:

$$(\mathbf{q}_k, \mathbf{s}_k, a_k, \emptyset, \mathbf{s}') \in \Gamma^{\text{Sp}}_{\mathbb{G}} \,, \tag{S}$$

i.e., the next input symbol is a_k, and universal right pebbles are moved from $\mathbf{s}_k$ to $\mathbf{s}'$. Accordingly, we define the input word for the next logbook as $w_{k+1} := w_k \cdot a_k$.

Move $(\text{S})_{\text{di}'}$ above immediately induces a move by Spoiler in every right game: For any right partial play $\pi^R_{k,j}$ ending in configuration $\langle \mathbf{r}^R_{k,j,k}, \mathbf{s}_{k,j,k} \rangle$ with $\mathbf{s}_{k,j,k} \subseteq \mathbf{s}_k$, let $\mathbf{s}'_j = \mathbf{s}' \downarrow_{\Delta(\mathbf{s}_{k,j,k}, a_k)}$ be the choice of Spoiler $(\text{S})_{\text{di}'}$ restricted to a_k-successors of $\mathbf{s}_{k,j,k}$ (cf.

Section 2.2). Accordingly, we let the right Spoiler in the j-th right game play

$$(\mathbf{r}^R_{k,j,k}, \mathbf{s}_{k,j,k}, a_k, \emptyset, \mathbf{s}'_j) \in \Gamma^{\mathrm{Sp}}_{\mathbb{G}_R} , \qquad \text{(S-R)}_{\mathrm{di}}$$

and we then apply right Duplicator's strategy σ_R thereto, obtaining

$$\sigma_L(\pi^R_{k,j})(a_k, \emptyset, \mathbf{s}'_j) = \langle \mathbf{r}^R_{k+1,j,k+1}, \mathbf{s}_{k+1,j,k+1} \rangle . \qquad \text{(D-R)}_{\mathrm{di}'}$$

That is, Duplicator selects, for each state $r \in \mathbf{r}^R_{k,j,k}$, a set of successors $c_j(r) \subseteq \Delta(r, a_k)$. We extend c_j to elements outside $\mathbf{r}^R_{k,j,k}$ by just setting $c_j(r) = \emptyset$ for $r \notin \mathbf{r}^R_{k,j,k}$. Then, by definition, $\mathbf{r}^R_{k+1,j,k+1} = \bigcup c_j(r^R_{k,j,k}) \neq \emptyset$. Notice that, since the game is uniform, no pebble is thrown away, that is, $c_j(r) \neq \emptyset$ for any $r \in \mathbf{r}^R_{k,j,k}$. Difficulty (D3) is solved as in existential rounds, by considering all possible successors. Thus, let $c(r) = \bigcup_j c_j(r)$.

Let $\pi^R_{k+1,j} = \pi^R_{k,j} \xRightarrow{a_k} \langle \mathbf{r}^R_{k+1,j,k+1}, \mathbf{s}_{k+1,j,k+1} \rangle$, and define the new right logbook B^R_{k+1} as $B^R_{k+1} := \{\pi^R_{k+1,0}, \pi^L_{k+1,1}, \ldots, \pi^R_{k+1,r(k)}\}$. The number of right games is preserved in a universal round.

Initially, the new left logbook B^L_{k+1} is empty. Each left game induces several new left games.

- Let $\langle \mathbf{q}_{k,i}, \mathbf{r}^L_{k,i,k} \rangle$ be the current configuration of the i-th left game. By (DiMiddle$_{\mathrm{L}}$), each $r \in \mathbf{r}^L_{k,i,k}$ belongs to some $\mathbf{r}^R_{k,j,k}$. But c_j is non-empty on $\mathbf{r}^R_{k,j,k}$, therefore, c is non-empty on $\mathbf{r}^L_{k,i,k}$.

  Let $\mathcal{X}_i = c(\mathbf{r}^L_{k,i,k}) = \{c(r) \mid r \in \mathbf{r}^L_{k,i,k}\}$ be the set of potential successors of states r in $\mathbf{r}^L_{k,i,k}$ as dictated in (D-R)$_{\mathrm{di}'}$, and let $Y_i = \otimes \mathcal{X}_i$ be the set of all possible combinations of successors. By the discussion above, $Y_i \neq \emptyset$.

- For each $\mathbf{r}^L_{k+1,h_k+1} \in Y_i$, create a new left game:

  - Set $\mathbf{r}^L_{k+1,h_k+1}$ induces the following move by Spoiler:

    $$(\mathbf{q}_{k,i}, \mathbf{r}^L_{k,i,k}, a_k, \emptyset, \mathbf{r}^L_{k+1,h_k+1}) \in \Gamma^{\mathrm{Sp}}_{\mathbb{G}_L} . \qquad \text{(S-L)}_{\mathrm{di}'}$$

    We apply the left Duplicator's winning strategy σ_L:

    $$\sigma_L(\pi^L_{k,i})(a_k, \emptyset, \mathbf{r}^L_{k+1,h_k+1}) = \langle \mathbf{q}_{k+1,h_k+1}, \mathbf{r}^L_{k+1,h_k+1} \rangle , \qquad \text{(D-L)}_{\mathrm{di}'}$$

    Since the game is uniform, by (U2) Duplicator does not drop any right pebble.

  - Create a new partial play $\pi'_{k+1} = \pi^L_{k,i} \xRightarrow{a_k} \langle \mathbf{q}_{k+1,h_k+1}, \mathbf{r}^L_{k+1,h_k+1} \rangle$ and add it to the new left logbook B^L_{k+1}.

The number of left games is *not preserved* in a universal round.

Let $\mathbf{q}_{k+1} = \bigcup_h \mathbf{q}_{k+1,h,k+1}$ (cf. (D-L)$_{\text{di}'}$) and $\mathbf{s}_{k+1} = \bigcup_j \mathbf{s}_{k+1,j,k+1}$ (cf. (D-R)$_{\text{di}'}$). Finally, we define Duplicator's move in $\mathbb{G}$ as

$$(\sigma_L \bowtie \sigma_R)(\pi_k)(a_k, \emptyset, \mathbf{s}') := \langle \mathbf{q}_{k+1}, \mathbf{s}_{k+1} \rangle \ . \qquad \text{(D)}_{\text{di}'}$$

Invariant Conditions (DiLeft) and (DiRight) are immediately satisfied by the definition of $\mathbf{q}_{k+1}$ and $\mathbf{s}_{k+1}$.

Condition (DiMiddle$_{\text{R}}$) is established inductively. Take any middle pebble $r \in \mathbf{r}^R_{k+1,j,k+1}$ in any right game, and we have to show that it belongs to some left game as well, i.e., that there exists h s.t. $r \in \mathbf{r}^L_{k+1,h,k+1}$. By construction, there exists a predecessor $r' \in \mathbf{r}^R_{k,j,k}$ s.t. $r \in c_j(r') \subseteq c(r')$. By induction hypothesis, there exists i s.t. the predecessor r' is in $\mathbf{r}^L_{k,i,k}$. Since $r \in c(r')$ is a successor of r', by construction we have that there exists h s.t. r is chosen as a successor, i.e., $r \in \mathbf{r}^L_{k+1,h,k+1}$.

In a universal round, condition (DiMiddle$_{\text{L}}$) holds by construction. Take any middle pebble $r \in \mathbf{r}^L_{k+1,h,k+1}$ in any right game, and we have to show that it belongs to some left game as well, i.e., that there exists i s.t. $r \in \mathbf{r}^R_{k+1,i,k+1}$. By construction, there exists i s.t. $\mathbf{r}^L_{k+1,h,k+1} \in \bigotimes c(\mathbf{r}^L_{k,i,k})$. This implies that there exists $r' \in \mathbf{r}^L_{k,i,k}$ s.t. $r \in c(r')$, which is to say $r \in c_j(r')$ for some j. Consequently, by the definition of c_j, $r \in \mathbf{r}^R_{k+1,j,k+1}$.

Finally, the rectangularity condition (Middle) is established inductively. Take any $q \in \mathbf{q}_{k+1}$ and $s \in \mathbf{s}_{k+1}$. We have to find h, j s.t. $q \in \mathbf{q}_{k+1,h,k+1}$, $s \in \mathbf{s}_{k+1,j,k+1}$, and $\mathbf{r}^L_{k+1,h,k+1} \cap \mathbf{r}^R_{k+1,j,k+1} \neq \emptyset$. By construction, there exist $q' \in \mathbf{q}_k$ and $s' \in \mathbf{s}_k$ s.t. $q' \xrightarrow{a_k} q$ and $s' \xrightarrow{a_k} s$. By induction hypothesis, there exist i, j s.t. $q' \in \mathbf{q}_{k,i,k}$, $s' \in \mathbf{s}_{k,j,k}$, and $\mathbf{r}^L_{k,i,k} \cap \mathbf{r}^R_{k,j,k} \neq \emptyset$. So, take some $r \in \mathbf{r}^L_{k,i,k} \cap \mathbf{r}^R_{k,j,k}$, and consider the set $c_j(r)$. By definition, $\mathbf{r}^R_{k+1,j,k+1} = \bigcup c_j(\mathbf{r}^R_{k,j,k})$. Since $r \in \mathbf{r}^R_{k,j,k}$, we have $c_j(r) \subseteq \mathbf{r}^R_{k+1,j,k+1}$. Consider now the set $Y_i = \bigotimes c(\mathbf{r}^L_{k,i,k})$. Since $r \in \mathbf{r}^L_{k,i,k}$ and $c_j(r)$ is a non-empty subset of $c(r)$, by the definition of unordered product there exists h s.t. $\mathbf{r}^L_{k+1,h,k+1} \in Y_i$ and $\mathbf{r}^L_{k+1,h,k+1}$ selects some element in $c_j(r)$. That is, $\mathbf{r}^L_{k+1,h,k+1} \cap c_j(r) \neq \emptyset$. But $c_j(r) \subseteq \mathbf{r}^R_{k+1,j,k+1}$, therefore $\mathbf{r}^L_{k+1,h,k+1} \cap \mathbf{r}^R_{k+1,j,k+1} \neq \emptyset$.

This concludes the description of the joint strategy $\sigma_L \bowtie \sigma_R$.

Lemma 5.6.4. $\sqsubseteq^{\text{di}}_{(1,n)}$, $\sqsubseteq^{\text{di}}_{(n,1)}$ *and* $\sqsubseteq^{\text{di}}_{(n,n)}$ *are transitive.*

Proof. We consider (n, n)-simulation; the other cases are similar. We refer to the logbook B_k at round k as defined above. Assume that some left pebble $q \in \mathbf{q}_k \cap F$ is accepting, and take an arbitrary pebble $s \in \mathbf{s}_k$. By (Middle), there exist i, j s.t. $q \in \mathbf{q}_{k,i,k}$, $s \in \mathbf{s}_{k,j,k}$ and there exists a common middle pebble $r \in \mathbf{r}^L_{k,i,k} \cap \mathbf{r}^R_{k,j,k}$. Since

q is accepting and σ_L is winning, r is accepting as well. But σ_R is winning, therefore s is accepting. Since s was arbitrary, all right pebbles are accepting in $\mathbf{s}_k$. □

5.6.3 Delayed and fair simulation

We instantiate the framework from Section 5.6.1. For fair simulation, we can make the following two simplifying assumptions:

(FA1) Duplicator always splits pebbles maximally on universal left states. That is, she uses only *left-blind strategies* (cf. Definition 5.4.14). This solves difficulty (D3) for universal pebbles.

As a consequence, left games all share the same left pebbles: I.e., if left pebbles in the outer game are those in

$$\rho_k^L = \mathbf{q}_0 \overset{a_0}{\Longrightarrow} \mathbf{q}_1 \overset{a_1}{\Longrightarrow} \cdots \overset{a_{k-1}}{\Longrightarrow} \mathbf{q}_k$$

then $\rho_{k,i}^L = \rho_k^L$ for any i.

(FA2) Duplicator splits pebbles uniformly on existential right states, although not necessarily maximally. That is, if an existential right pebble belongs to several games, then the same successors are selected in all these games. This solves difficulty (D3) for existential pebbles.

This is achieved by taking the union over all successors defined by the original winning strategy in the different games. This is correct only if there are uniformly finitely many different games at any round, since winning plays for Duplicator in delayed and fair simulation are closed only under *finite* union (cf. Lemma 5.4.5). We ensure this by bookkeeping at most one game for any given configuration.

Consequently, all right games share the same right pebbles: I.e., if right pebbles in the outer game are those in

$$\rho_k^R = \mathbf{s}_0 \overset{a_0}{\Longrightarrow} \mathbf{s}_1 \overset{a_1}{\Longrightarrow} \cdots \overset{a_{k-1}}{\Longrightarrow} \mathbf{s}_k$$

then $\rho_{k,j}^R = \rho_k^R$ for any j.

Remark 5.6.5. Neither of the two assumptions above is correct for direct simulation. The reason is that the acceptance condition for direct simulation is not "monotone" w.r.t. having neither more left nor more right pebbles. For example, (FA1) is incorrect since having more left pebbles is more likely to cause a visit to an accepting state, and

this might force Duplicator to possibly drop some right pebble in order to satisfy the winning condition. Similarly, (FA2) is incorrect since more right pebbles might fall outside accepting states, thus possibly violating the acceptance condition.

Thanks to the assumptions above, we just need two logbook properties: The first property (Middle_R) is a refinement of (DiMiddle_R) for direct simulation, and it says that "every left every path in any right game is also a right path in some left game". We use it to ensure that certain sets defined during the construction are non-empty, and to propagate the acceptance condition from left games to right games. The second property ($\text{Extension}_\text{L}$) is "new right paths in left games from $\psi^L_{k,i}$ always arise as an extension of old ones from $\psi^L_{k',l}$".

$$\forall(\rho \in \psi^R_{k,j}) \cdot \exists i \cdot (\rho \in \psi^L_{k,i}) \tag{Middle_R}$$

$$\forall(\psi^L_{k,i} = r^L_{k,i,0} \xRightarrow{a_0} \cdots \xRightarrow{a_{k-1}} r^L_{k,i,k}) \cdot \forall(k' \leq k) \cdot \exists l \cdot \psi^L_{k',l} = r^L_{k,i,0} \xRightarrow{a_0} \cdots \xRightarrow{a_{k-1}} r^L_{k,i,k'} \tag{$\text{Extension}_\text{L}$}$$

Inductively assume that, at round k, B_k is a valid logbook, and we show how to build a new logbook $B_{k+1} = (B^L_{k+1}, B^R_{k+1}, w_{k+1}, \pi_{k+1})$ for the next round.

Existential round Assume that Spoiler moves as follows:

$$(\mathbf{q}_k, \mathbf{s}_k, a_k, \mathbf{q}_{k+1}, \emptyset) \in \Gamma^{\text{Sp}}_{\mathbb{G}} \ . \tag{S$)_\text{f}$}$$

Let the input word for the next logbook be $w_{k+1} := w_k \cdot a_k$. This immediately induces a move by Spoiler in every left game: Indeed, any left partial play $\pi^L_{k,i}$ ends in a configuration of the form $\langle \mathbf{q}_k, \mathbf{r}^L_{k,i,k} \rangle$, thus, we let the left Spoiler in the i-th left game play

$$(\mathbf{q}_k, \mathbf{r}^L_{k,i,k}, a_k, \mathbf{q}_{k+1}, \emptyset) \in \Gamma^{\text{Sp}}_{\mathbb{G}_L} \ , \tag{S-L$)_\text{f}$}$$

and we then apply left Duplicator's strategy σ_L thereto, obtaining

$$\sigma_L(\pi^L_{k,i})(a_k, \mathbf{q}_{k+1}, \emptyset) = \langle \mathbf{q}_{k+1}, \mathbf{r}^L_{k+1,i,k+1} \rangle \ . \tag{D-L$)_\text{f}$}$$

Since the game is uniform, Duplicator does not throw any left pebble in $\mathbf{q}_{k+1}$ away. By assumption (FA2), Duplicator selects successors by means of a *uniform* choice function $c : Q \mapsto 2^Q$ (i.e., independent of i) s.t., for each state $r \in \mathbf{r}^L_{k,i,k}$, $c(r)$ is a subset of $\Delta(r, a_k)$, and $c(r) = \emptyset$ for $r \notin \mathbf{r}^L_{k,i,k}$. In the former case, $c(r)$ is non-empty by (U2). By construction, $\mathbf{r}^L_{k+1,i,k+1} = \bigcup c(\mathbf{r}^L_{k,i,k})$. The next configuration of the i-th left game is $\langle \mathbf{q}_{k+1}, \mathbf{r}^L_{k+1,i,k+1} \rangle$ which naturally defines the corresponding partial play at round $k+1$

$\pi^L_{k+1,i} = \pi^L_{k,i} \xRightarrow{a_k} \langle \mathbf{q}_{k+1}, \mathbf{r}^L_{k+1,i,k+1} \rangle$. Then, $\pi^L_{k+1,i}$ is added to B_{k+1^L} only if there exists no other path already in B^L_{k+1} ending in the same configuration $\langle \mathbf{q}_{k+1}, \mathbf{r}^L_{k+1,i,k+1} \rangle$.

Initially, the new right logbook B^R_{k+1} is empty. For each right game j, we do the following.

- Let $\langle \mathbf{r}^R_{k,j,k}, \mathbf{s}_k \rangle$ be the current configuration of the j-th right game. Since c is non-empty on $\mathbf{r}^L_{k,i,k}$, for any i, by property (Middle$_\mathrm{R}$), c is non-empty on $\mathbf{r}^R_{k,j,k}$.

  Let $\mathcal{X}_j = c(\mathbf{r}^R_{k,j,k}) = \{c(r) \mid r \in \mathbf{r}^R_{k,j,k}\}$ be the set of potential successors of states r in $\mathbf{r}^R_{k,j,k}$ as dictated by (D-L)$_\mathrm{di}$, and take $Y_j = \otimes \mathcal{X}_j$ to be the set of all possible combinations of successors. X_j does not contain the empty set, thus $Y_j \neq \emptyset$.

- For every $\mathbf{r}^R_{k+1,h,k+1} \in Y_j$, create a new right game:

  - First, make the right Spoiler select transition

    $$(\mathbf{r}^R_{k,j,k}, \mathbf{s}_k, a_k, \mathbf{r}^R_{k+1,h,k+1}, \emptyset) \in \Gamma^{\mathrm{Sp}}_{\mathbb{G}_R} \qquad \text{(S-R)}_\mathrm{f}$$

    Then, we apply the Duplicator's winning strategy σ_R:

    $$\sigma_R(\pi^R_{k,j})(a_k, \mathbf{r}^R_{k+1,h,k+1}, \emptyset) = \langle \mathbf{r}^R_{k+1,h,k+1}, \mathbf{s}_{k+1} \rangle \qquad \text{(D-R)}_\mathrm{f}$$

    Since the game is uniform, by (U2) Duplicator does not drop any left pebble. Moreover, by assumption (FA2), Duplicator splits pebbles on the right uniformly across different games: Consequently, the set $\mathbf{s}_{k+1}$ does not depend on the current right game h.

  - Create a new partial play $\pi^R_{k+1,h} = \pi^R_{k,j} \xRightarrow{a_k} \langle \mathbf{r}^R_{k+1,h,k+1}, \mathbf{s}_{k+1} \rangle$, and, if there exists no other play in B^R_{k+1} ending in $\langle \mathbf{r}^R_{k+1,h,k+1}, \mathbf{s}_{k+1} \rangle$, then add it to B^R_{k+1}.

We define Duplicator's move in $\mathbb{G}$ as

$$(\sigma_L \bowtie \sigma_R)(\pi_k)(a_k, \mathbf{q}_{k+1}, \emptyset) := \langle \mathbf{q}_{k+1}, \mathbf{s}_{k+1} \rangle \qquad \text{(D)}_\mathrm{f}$$

Again, by (U2) Duplicator does not throw left pebbles away. The last component of the new logbook is $\pi_{k+1} = \pi_k \xRightarrow{a_k} \langle \mathbf{q}_{k+1}, \mathbf{s}_{k+1} \rangle$.

Invariant Condition (Middle$_\mathrm{R}$) is established inductively. Take any middle path $\rho' \in \psi^R_{k+1,h}$. By construction, ρ' ends in some state $r' \in \mathbf{r}^R_{k+1,h,k+1}$, and there exist j and $\rho \in \psi^R_{k,j}$ s.t. $\rho' = \rho \xrightarrow{a_k} r'$. Assume ρ ends in state r. By construction, $r' \in c(r)$. By induction hypothesis, there exists i s.t. $\rho \in \psi^L_{k,i}$. We recall that Duplicator uses the

same uniform choice function c across different left games. But $r' \in c(r)$, therefore r' is selected as a successor of r also in the i-th left game. Thus, $r' \in \mathbf{r}^L_{k+1,i,k+1}$, implying $\rho \in \psi^L_{k+1,i}$.

Condition (Extension$_\mathrm{L}$) holds since, by definition, $\pi^L_{k+1,i}$ extends $\pi^L_{k,i}$, therefore, $\psi^L_{k+1,i}$ extends $\psi^L_{k,i}$.

Universal round Assume that Spoiler moves as follows:

$$(\mathbf{q}_k, \mathbf{s}_k, a_k, \emptyset, \mathbf{s}_{k+1}) \in \Gamma^{\mathrm{Sp}}_{\mathbb{G}} \ , \tag{S}$$

i.e., the next input symbol is a_k, and right pebbles are moved from $\mathbf{s}_k$ to $\mathbf{s}_{k+1}$. Accordingly, we define the input word for the next logbook as $w_{k+1} := w_k \cdot a_k$. This immediately induces a move by Spoiler in every right game: Let the right Spoiler in the j-th right game play

$$(\mathbf{r}^R_{k,j,k}, \mathbf{s}_k, a_k, \emptyset, \mathbf{s}_{k+1}) \in \Gamma^{\mathrm{Sp}}_{\mathbb{G}_R} \ , \tag{S-R}$$

By assumption (FA1), Duplicator's strategy σ_R is left-blind:

$$\sigma_R(\pi^R_{k,j})(a_k, \emptyset, \mathbf{s}_{k+1}) = \langle \mathbf{r}^R_{k+1,j,k+1}, \mathbf{s}_{k+1} \rangle \tag{D-R}$$

$$\mathbf{r}^R_{k+1,j,k+1} = \Delta(\mathbf{r}^R_{k,j,k}, a_k) \tag{5.1}$$

Let $\pi^R_{k+1,j} = \pi^R_{k,j} \overset{a_k}{\Longrightarrow} \langle \mathbf{r}^R_{k+1,j,k+1}, \mathbf{s}_{k+1} \rangle$, and add it to the new right logbook B^R_{k+1} only if there exists no other play in B^R_{k+1} ending in $\langle \mathbf{r}^R_{k+1,j,k+1}, \mathbf{s}_{k+1} \rangle$.

Initially, the new left logbook B^L_{k+1} is empty. For each left game i,

- Let $\langle \mathbf{q}_k, \mathbf{r}^L_{k,i,k} \rangle$ be the current configuration of the i-th left game.

  Let $\mathcal{X}_i = \Delta(\mathbf{r}^L_{k,i,k}, a_k)$ be the set of all a_k-successors of states from $\mathbf{r}^L_{k,i,k}$, as dictated in (D-R)$_{\mathrm{f}'}$, and let $Y_i = \otimes \mathcal{X}_i$ be the set of all possible combinations of successors. Clearly, $Y_i \neq \emptyset$.

- For each $\mathbf{r}^L_{k+1,h,k+1} \in Y_i$, create a new left game:

  – First, we have the left Spoiler select transition

  $$(\mathbf{q}_k, \mathbf{r}^L_{k,i,k}, a_k, \emptyset, \mathbf{r}^L_{k+1,h,k+1}) \in \Gamma^{\mathrm{Sp}}_{\mathbb{G}_L} \tag{S-L}$$

  By assumption (FA1), σ_L is left-blind:

  $$\sigma_L(\pi^L_{k,i})(a_k, \emptyset, \mathbf{r}^L_{k+1,h,k+1}) = \langle \mathbf{q}_{k+1}, \mathbf{r}^L_{k+1,h,k+1} \rangle \tag{D-L}$$

  $$\mathbf{q}_{k+1} = \Delta(\mathbf{q}_k, a_k) \tag{5.2}$$

Also, since the game is uniform, by (U2) Duplicator does not drop any right pebble.

– Create a new partial play $\pi^L_{k+1,h} = \pi^L_{k,i} \xRightarrow{a_k} \langle \mathbf{q}_{k+1}, \mathbf{r}^L_{k+1,h,k+1} \rangle$ and, if there exists no other play in B^L_{k+1} ending in $\langle \mathbf{q}_{k+1}, \mathbf{r}^L_{k+1,h,k+1} \rangle$, then add it to B^L_{k+1}.

We define Duplicator's move in $\mathbb{G}$ as

$$(\sigma_L \bowtie \sigma_R)(\pi_k)(a_k, \emptyset, \mathbf{s}_{k+1}) := \langle \mathbf{q}_{k+1}, \mathbf{s}_{k+1} \rangle \qquad \text{(D)}_{\text{f}'}$$

The last component of the new logbook is $\pi_{k+1} = \pi_k \xRightarrow{a_k} \langle \mathbf{q}_{k+1}, \mathbf{s}_{k+1} \rangle$.

Invariant Condition (Middle$_\text{R}$) is established inductively. Take any middle path $\rho' \in \psi^R_{k+1,j}$ in any right game. By construction, $\psi^R_{k+1,j} = \psi^R_{k,j} \xRightarrow{a_k} \mathbf{r}^R_{k+1,j,k+1}$. Therefore, there exists a prefix $\rho \in \psi^R_{k,j}$ of ρ'. By induction hypothesis, there exists i s.t. $\rho \in \psi^L_{k,i}$. Since ρ ends in a universal state, and Duplicator uses a left-blind strategy σ_R, there exists h s.t. $\rho \in \psi^L_{k+1,h}$.

Condition (Extension$_\text{L}$) holds since, by construction, $\pi^L_{k+1,h}$ extends $\pi^L_{k,i}$, for some i. Therefore, $\psi^L_{k+1,h}$ extends $\psi^L_{k,i}$.

This concludes the description of the joint strategy $\sigma_L \bowtie \sigma_R$.

The limit logbook Before turning to the proof of transitivity for fair simulation, we define a limit version of the logbook, and study its properties. We introduce the *limit logbook* $B_\omega = (B^L_\omega, B^R_\omega, w, \pi_\omega)$, which is constructed as follows. The infinite word $w = a_0 a_1 \cdots$ is just the one built by Spoiler, and it is equal to the limit of the sequence of finite words $w_0, w_1, \ldots$. Similarly, π_ω is just the limit of $\pi_0, \pi_1, \ldots$. Therefore,

$$\begin{aligned}
\pi_\omega &= \rho^L_\omega \times \rho^R_\omega, \text{ with} \\
\rho^L_\omega &= \mathbf{q}_0 \xRightarrow{a_0} \mathbf{q}_1 \xRightarrow{a_1} \cdots \\
\rho^R_\omega &= \mathbf{s}_0 \xRightarrow{a_0} \mathbf{s}_1 \xRightarrow{a_1} \cdots .
\end{aligned}$$

The left limit logbook $B^L_\omega = \{\pi^L_{\omega,h_0}, \pi^L_{\omega,h_1}, \ldots\}$ contains all infinite sequences of configurations which are limit of finite sequences in $B^L_0, B^L_1, \ldots$. That is, $\pi^L_{\omega,h}$ is in B^L_ω iff there exists a strongly coherent sequence of partial plays $\pi^L_{0,h(0)} \in B^L_0, \pi^L_{1,h(1)} \in B^L_1, \ldots$ s.t. $\pi^L_{\omega,h} = \lim_k \pi^L_{k,h(k)}$ is the limit of this sequence. We have:

$$\begin{aligned}
\pi^L_{\omega,h} &= \rho^L_\omega \times \psi^L_{\omega,h}, \text{ with} \\
\psi^L_{\omega,h} &= \mathbf{r}^L_{0,h(0),0} \xRightarrow{a_0} \mathbf{r}^L_{1,h(1),1} \xRightarrow{a_1} \cdots .
\end{aligned}$$

The right limit logbook $B^R_\omega = \{\pi^R_{\omega,g_0}, \pi^R_{\omega,g_1}, \dots\}$ is defined in an analogous way w.r.t. $B^R_0, B^R_1, \dots$: $\pi^R_{\omega,g} \in B^R_\omega$ iff there exists a strongly coherent sequence of partial plays $\pi^R_{0,g(0)} \in B^R_0, \pi^R_{1,g(1)} \in B^R_1, \dots$ s.t. $\pi^R_{\omega,g} = \lim_k \pi^R_{k,g(k)}$. We write:

$$\pi^R_{\omega,g} = \psi^R_{\omega,g} \times \rho^R_\omega, \text{ with}$$
$$\psi^R_{\omega,g} = \mathbf{r}^R_{0,g(0),0} \overset{a_0}{\Longrightarrow} \mathbf{r}^R_{1,g(1),1} \overset{a_1}{\Longrightarrow} \cdots$$

A limit version of the logbook property (Middle$_\mathrm{R}$) holds for the limit logbook:

Lemma 5.6.6. *Let $B_\omega = (B^L_\omega, B^R_\omega, w, \pi_\omega)$ be the limit logbook as defined above. Then,*

$$\forall(\rho \in \psi^R_{\omega,g}) \cdot \exists h \cdot (\rho \in \psi^L_{\omega,h}) \qquad \text{(LimMiddle}_\mathrm{R}\text{)}$$

Proof. Assume ρ is any path in $\psi^R_{\omega,g}$, and let ρ_k be the finite prefix of ρ of length $k+1$. Consider the set of multipaths $\Psi = \{\psi^L_{k,i} \mid \rho_k \in \psi^L_{k,i}\}$ containing prefixes of ρ. By (Extension$_\mathrm{L}$), every multipath $\psi^L_{k,i}$ extends a previous multipath $\psi^L_{k-1,l}$, for some l. Therefore, if $\rho_k \in \psi^L_{k,i}$, then $\rho_{k-1} \in \psi^L_{k-1,l}$. Thus, Ψ is closed under prefixes. It represents a finitely branching tree with finite branches from Ψ, whose vertices are labelled with sets of states.

By definition, ρ_k belongs to $\psi^R_{k,g(k)}$. By (Middle$_\mathrm{R}$), for each k there exists $h(k)$ s.t. $\rho_k \in \psi^L_{k,h(k)}$. Therefore, the tree represented by Ψ is infinite. By König's Lemma, there exists a single infinite branch $\psi^L_{\omega,h}$ containing all finite prefixes of ρ. Therefore, $\rho \in \psi^L_{\omega,h}$. □

Lemma 5.6.7. *For $x \in \{\mathrm{de}, \mathrm{f}\}$, $\sqsubseteq^x_{(1,n)}$, $\sqsubseteq^x_{(n,1)}$ and $\sqsubseteq^x_{(n,n)}$ are transitive.*

Proof. We consider (n,n) simulation; the other cases are similar.

- $x = \mathrm{de}$: Assume that $\mathbf{q}_i \Longrightarrow^\forall_F \mathbf{q}_{i'}$ in ρ^L_ω. We show that there exist l, l' s.t. $l \geq i$ and $\mathbf{s}_l \Longrightarrow^\exists_F \mathbf{s}_{l'}$ in ρ^R_ω.

  Consider any right play $\pi^R_{\omega,g} = \psi^R_{\omega,g} \times \rho^R_\omega$, and any infinite branch $\rho \in \psi^R_{\omega,g}$. We argue that ρ visits an accepting state after round i. By (LimMiddle$_\mathrm{R}$), $\rho \in \psi^L_{\omega,h}$ for some h. Let $\pi^L_{\omega,h} = \rho^L_\omega \times \psi^L_{\omega,h}$ be the corresponding left play. Since $\mathbf{q}_i \Longrightarrow^\forall_F \mathbf{q}_{i'}$ in ρ^L_ω by assumption, and $\pi^L_{\omega,h}$ is conform to the winning strategy σ_L, there exist j, j' s.t. $j \geq i$ and $\mathbf{r}^L_{j,h(j),j} \Longrightarrow^\exists_F \mathbf{r}^L_{j',h(j'),j'}$. Since the game is uniform, by (U4), $\mathbf{r}^L_{j,h(j),j} \Longrightarrow^\forall_F \mathbf{r}^L_{j',h(j'),j'}$. Therefore, by the definition of $\Longrightarrow^\forall_F$, ρ visits an accepting state after round i.

  Therefore, every infinite branch $\rho \in \psi^R_{\omega,g}$ visits an accepting state after round i. There exists a uniform $i'' \geq i$ s.t. this visit happens before round i'' *for any such*

ρ. Thus, $\mathbf{r}^R_{i,g(i),i} \Longrightarrow^{\exists}_F \mathbf{r}^R_{i'',g(i''),i''}$. Since π^R_ω is conform to the winning strategy σ_R, there exist l, l' s.t. $\mathbf{s}_l \Longrightarrow^{\exists}_F \mathbf{s}_{l'}$ in ρ^R_ω.

- $x = \mathrm{f}$: Assume that ρ^L_ω is universally fair, and we show that ρ^R_ω is existentially fair.

  Consider any right play $\pi^R_{\omega,g} = \psi^R_{\omega,g} \times \rho^R_\omega$, and any infinite branch $\rho \in \psi^R_{\omega,g}$. We argue that ρ is fair. By (LimMiddle$_\mathrm{R}$), $\rho \in \psi^L_{\omega,h}$ for some h. Let $\pi^L_{\omega,h} = \rho^L_\omega \times \psi^L_{\omega,h}$ be the corresponding left play. Since ρ^L_ω is universally fair by assumption, and $\pi^L_{\omega,h}$ is conform to the winning strategy σ_L, $\psi^L_{\omega,h}$ is existentially fair. Since the game is uniform, by (U4), $\psi^L_{\omega,h}$ is universally fair. Since $\rho \in \psi^L_{\omega,h}$, by definition ρ is fair.

  Since ρ was arbitrary, $\psi^R_{\omega,g}$ is universally fair by definition. Since π^R_ω is conform to the winning strategy σ_R, ρ^R_ω is existentially fair. □

5.6.4 Comparison with previous work [29]

The notion of multipebble delayed simulation that we study in this thesis differs from the one introduced in [29]. The one from [29] is not adequate as it turns out not to be transitive, as incorrectly claimed therein.

To compare the new notion from this thesis and the old one from [29], we rephrase both in a common formalism. Both notions refer to pebbles being good in some interval, with two important differences.

1. The first difference lies in how those intervals are relatively ordered. In the current definition, $\pi \in W^{\mathrm{de}}$ iff

$$\forall (i \leq i') \cdot \mathbf{q}_i \Longrightarrow^{\forall}_F \mathbf{q}_{i'} \text{ implies } \exists (j \leq j') \cdot i \leq j \wedge \mathbf{s}_j \Longrightarrow^{\exists}_F \mathbf{s}_{j'} \qquad \text{(de)}$$

   while in the notion from [29] we have $\pi \in W^{\mathrm{de}'}$ iff

$$\forall (i \leq i') \cdot \mathbf{q}_i \Longrightarrow^{\forall}_F \mathbf{q}_{i'} \text{ implies } \exists (j \leq j') \cdot i' \leq j' \wedge \mathbf{s}_j \Longrightarrow^{\exists}_F \mathbf{s}_{j'} \qquad \text{(de')}$$

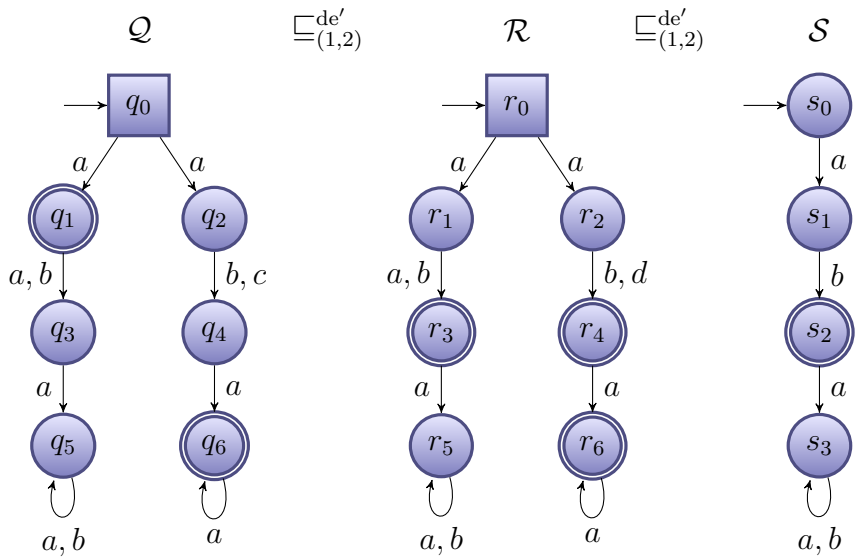
2. The second difference is that i' and j' above are required to be the *minimal* indices satisfying $\mathbf{q}_i \Longrightarrow^{\forall}_F \mathbf{q}_{i'}$ and $\mathbf{s}_j \Longrightarrow^{\exists}_F \mathbf{s}_{j'}$, respectively, while we require no such restriction here.

That is, here we require the past endpoints to be ordered $i <= j$, while in [29] it is the future endpoints that are required to be ordered $i' <= j'$. The two notions differ only if

there is more than one left pebble: For $(1, k)$-simulations, i and j can always be taken equal to i' and j', respectively, and the two notions coincide.

Per se, the different synchronization requirement is not a problem for transitivity. However, the further minimality restriction on endpoints i' and j' does not behave nicely when several games are composed together, since Spoiler might actually force accepting states to be visited *too early*, as we show in the next example.

Example 5.6.2 - $(n, 1)$ and (n, n)-delayed simulation from [29] are not transitive



The notion of delayed simulation from [29] is not transitive in the $(n, 1)$ and (n, n) case. This was incorrectly claimed in Theorem 6 therein for the $(n, 1)$ case. Consider the automata $\mathcal{Q}$, $\mathcal{R}$ and $\mathcal{S}$ above. Clearly, more right pebbles don't help Duplicator. Thus, $\sqsubseteq^{de'}_{(n,1)} = \sqsubseteq^{de'}_{(n,n)}$. With two left pebbles, $\mathcal{R}$ simulates $\mathcal{Q}$ and $\mathcal{S}$ simulates $\mathcal{R}$:

- $\mathcal{Q} \sqsubseteq^{de'}_{(2,1)} \mathcal{R}$: From the initial configuration $\langle q_0, r_0 \rangle$ (Round 0), Spoiler plays action a and transition $r_0 \xrightarrow{a} r_i$, for $i \in \{1, 2\}$. Duplicator splits pebbles maximally on the left, and the game goes to configuration $\langle \{q_1, q_2\}, r_i \rangle$ (Round 1). Then, Spoiler is forced to play action b, and Duplicator's reply depends on i.
  - If $i = 1$, then Duplicator drops the left pebble on q_2, and the game goes to configuration $\langle q_3, r_3 \rangle$ (Round 2). Duplicator has satisfied the acceptance obligation $\{q_1, q_2\} \Longrightarrow^{\forall}_{F} q_3$ since $r_3 \in F$. From the latter position, Dupli-

cator plays a "copycat strategy", and the game goes to $\langle q_5, r_5\rangle$ (Round 3), where it stays forever. Even if $\{q_1, q_2\} \Longrightarrow_F^{\forall} q_5$ holds at Round 3, this does not raise any obligation, since the the additional minimality constraint from [29] fails. Thus, Duplicator wins in this case.

– If $i = 2$, then Duplicator always keeps both left pebbles, and the game goes to $\langle\{q_3, q_4\}, r_4\rangle$ (Round 2). Then, Spoiler plays action a, and the game goes to $\langle\{q_5, q_6\}, r_6\rangle$. Duplicator satisfies the acceptance obligation $\{q_1, q_2\} \Longrightarrow_F^{\forall} \{q_5, q_6\}$ since $r_6 \in F$. Then, the game stays forever in $\langle\{q_5, q_6\}, r_6\rangle$, and no new obligation is raised, thanks to by the minimality condition. Thus, Duplicator wins also in this case.

- $\mathcal{R} \sqsubseteq^{\mathrm{de}'}_{(2,1)} \mathcal{S}$: From the initial configuration $\langle r_0, s_0\rangle$ (Round 0), Spoiler plays action a and Duplicator splits left pebbles maximally. Then, the game goes to configuration $\langle\{r_1, r_2\}, s_1\rangle$ (Round 1), from which Spoiler is forced to play action b. Therefore, the game goes to configuration $\langle\{r_3, r_4\}, s_2\rangle$ (Round 2), at which point Duplicator fulfills with $s_2 \in F$ the obligation $\{r_3, r_4\} \Longrightarrow_F^{\forall} \{r_3, r_4\}$. Then, the game goes to $\langle\{r_5, r_6\}, s_3\rangle$ (Round 3), where it stays forever. No more obligations are raised (by the minimality condition), and Duplicator wins.

However, $\mathcal{Q} \not\sqsubseteq^{\mathrm{de}'}_{(2,1)} \mathcal{S}$. (This suffices to show that $\sqsubseteq^{\mathrm{de}'}_{(n,1)}$ is not transitive, since $\sqsubseteq^{\mathrm{de}'}_{(2,1)} = \sqsubseteq^{\mathrm{de}'}_{(3,1)}$ in this example.) Indeed, from configuration $\langle q_0, s_0\rangle$ (Round 0), let Spoiler play action a and transition $s_0 \xrightarrow{a} s_1$. Duplicator is forced to split pebbles maximally, to prevent Spoiler from playing actions not available in $\mathcal{S}$. Thus, the game goes to configuration $\langle\{q_1, q_2\}, s_1\rangle$ (Round 1), from where Spoiler is forced to play action b, bringing the game to $\langle\{q_3, q_4\}, s_2\rangle$ (Round 2). Then, Spoiler plays action a, and Duplicator has three options for left pebbles, each of those is losing:

- Throw away the pebble on q_4: The game goes to $\langle q_5, s_3\rangle$, from which Spoiler wins by playing action b, which is unavailable form s_3.

- Throw away the pebble on q_3: The game goes to $\langle q_6, s_3\rangle$, and Duplicator loses since $q_6 \in F$ but right pebble can be accepting in the future.

- Keep both pebbles: The game goes to $\langle\{q_5, q_6\}, s_3\rangle$. An obligation is raised since $\{q_1, q_2\} \Longrightarrow_F^{\forall} \{q_5, q_6\}$, but Duplicator cannot match it, and she loses.

Therefore, Spoiler wins in either case, and $\mathcal{Q} \not\sqsubseteq^{\mathrm{de}'}_{(2,1)} \mathcal{S}$.

5.7 Multipebble simulations and subset-like constructions

In this section, we establish connections between multipebble fair simulation and well-known subset constructions for nondeterministic and alternating Büchi automata. Given an automaton $\mathcal{A}$, a *subset-like construction* produces another automaton $\mathcal{T}(\mathcal{A})$ recognizing the same language as $\mathcal{A}$, s.t. $\mathcal{T}(\mathcal{A})$ satisfies some additional desirable property depending on $\mathcal{T}$—for example, one might be interested in removing universal states from the automaton. Construction $\mathcal{T}$ *characterizes* (k_0, k_1)-fair simulation, iff (k_0, k_1)-fair simulation between $\mathcal{A}$ and $\mathcal{B}$ can be decided by computing $(1, 1)$-fair simulation between $\mathcal{T}(\mathcal{A})$ and $\mathcal{T}(\mathcal{B})$:

$$\mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(k_0,k_1)} \mathcal{S} \iff \mathcal{T}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{T}(\mathcal{S})$$

Let n be the maximal number of states in $\mathcal{A}$ and $\mathcal{B}$. In Section 5.7.1, we show that a well-known subset construction for alternation removal in Büchi automata, the so-called *Miyano-Hayashi construction* [92], characterizes $(n, 1)$-fair simulation. In Section 5.7.2, we show that a dual construction can be defined as to characterize $(1, n)$-fair simulation. Finally, in Section 5.7.3 we show that even (n, n)-fair simulation can be reduced to $(1, 1)$-fair simulation, by performing the MH-construction on $\mathcal{A}$ and its dual on $\mathcal{B}$.

5.7.1 Characterizing $(n, 1)$-fair simulation

In this section, we show that a well-known subset construction for alternating Büchi automata, called the *Miyano-Hayashi construction* [92], characterizes $(n, 1)$-fair simulation. For an automaton $\mathcal{A}$, let $\mathcal{MH}(\mathcal{A})$ be the automaton resulting from applying the MH-construction.

Theorem 5.7.1. *Let $\mathcal{Q}$ and $\mathcal{S}$ be two ABAs, and let $n = |\mathcal{Q}|$. Then,*

$$\mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(n,1)} \mathcal{S} \iff \mathcal{MH}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{MH}(\mathcal{S})$$

Remark 5.7.2. The statement below (notice the $(1, 1)$-simulation in the premise)

$$\mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{S} \implies \mathcal{MH}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{MH}(\mathcal{S})$$

had already appeared in [55]. We prove a stronger property by taking $(n, 1)$-simulation in the premise. The converse direction

$$\mathcal{MH}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{MH}(\mathcal{S}) \implies \mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(n,1)} \mathcal{S}$$

has not appeared before. It can only be established in the context of multipebble simulations.

In Section 5.7.1.1 we introduce the construction, in Section 5.7.1.2 we show its correctness, and in Section 5.7.1.3 we prove the theorem. Along the way, we give examples showing that all the stated properties are tight and cannot be improved.

5.7.1.1 The Miyano-Hayashi construction

The Miyano-Hayashi (MH) construction [92] is a subset-like construction which transforms an alternating Büchi automaton $\mathcal{Q}$ into a nondeterministic (non-alternating) Büchi automaton $\mathcal{MH}(\mathcal{Q})$ recognizing the same language. It removes universal nondeterminism from $\mathcal{Q}$. Therefore, we say it performs the *de-universalization* of $\mathcal{Q}$.

The idea is similar to the analogous construction over finite words, with extra bookkeeping needed for recording visits to accepting states, which may not occur at the same time for different runs. A set of obligations is maintained, encoding the requirement that, independently of how universal non-determinism is resolved, an accepting state has to be eventually reached.

Definition 5.7.3 (The Miyano-Hayashi construction [92])**.** *Given an alternating Büchi automaton $\mathcal{Q} = (Q, \Sigma, q_I, \Delta, F, E, U)$, the Miyano-Hayashi construction yields a nondeterministic Büchi automaton*

$$\mathcal{MH}(\mathcal{Q}) := (Q', \Sigma, \mathbf{q}_I, \Delta', F', E', U')$$

where macrostates in $Q' \subseteq 2^Q \times 2^Q$ consist of pairs $\langle \mathbf{q}, \mathbf{o} \rangle$ of subsets of Q s.t. $\mathbf{o} \subseteq \mathbf{q}$, the initial macrostate is $\mathbf{q}_I := (\{q_I\}, \{q_I\} \setminus F)$, there are no universal states $U' := \emptyset$ (hence, $E' = Q'$), the set of accepting macrostates is

$$F' := \{(\mathbf{q}, \mathbf{o}) \mid \mathbf{o} = \emptyset\},$$

and there is a transition $((\mathbf{q}, \mathbf{o}), a, (\mathbf{q}', \mathbf{o}')) \in \Delta'$ iff there exists $\mathbf{q}'' \in \bigotimes \Delta(\mathbf{q}^E, a)$ s.t.

$$\mathbf{q}' = \Delta(\mathbf{q}^U, a) \cup \mathbf{q}''$$

and,

- *if* $\mathbf{o} = \emptyset$*, then* $\mathbf{o}' = \mathbf{q}' \setminus F$*,*
- *otherwise,* $\mathbf{o}' = \big(\Delta(\mathbf{o}^U, a) \cup \mathbf{o}''\big) \setminus F$*, where* $\mathbf{o}'' = \mathbf{q}''{\downarrow}_{\Delta(\mathbf{o}^E,a)}$ *is the restriction of* $\mathbf{q}''$ *to* $\Delta(\mathbf{o}^E, a)$ *(which is well defined, since* $\mathbf{o}^E \subseteq \mathbf{q}^E$ *by construction; cf. Section 2.2).*

There is a tight relationship between obligations in the MH-construction and universally good multipaths.

Lemma 5.7.4. *Let* $\pi = (\mathbf{q}_0, \mathbf{o}_0) \xrightarrow{a_0} (\mathbf{q}_1, \mathbf{o}_1) \xrightarrow{a_1} \cdots$ *be any path in* $\mathcal{MH}(\mathcal{Q})$*. If* $(\mathbf{q}_j, \mathbf{o}_j) \in F'$ *(that is,* $\mathbf{o}_j = \emptyset$*), then there exists* $i \leq j$ *s.t.* $\mathbf{q}_i$ *is universally good at round* j*, i.e.,* $\mathbf{q}_i \Longrightarrow_F^{\forall} \mathbf{q}_j$*.*

The MH-construction produces a different automaton only if there are universal states. I.e., if $U = \emptyset$, then $\mathcal{MH}(\mathcal{Q}) = \mathcal{Q}$. In particular, since $\mathcal{MH}(\mathcal{Q})$ is non-alternating by definition, $\mathcal{MH}(\mathcal{MH}(\mathcal{Q})) = \mathcal{MH}(\mathcal{Q})$, i.e., the Miyano-Hayashi construction is an idempotent operation.

Lemma 5.7.5. *For any ABA* $\mathcal{Q}$*,* $\mathcal{MH}(\mathcal{MH}(\mathcal{Q})) = \mathcal{MH}(\mathcal{Q})$*.*

5.7.1.2 Correctness of the construction

The MH-construction is correct in the sense that it preserves the language of the automaton: $\mathcal{L}(\mathcal{Q}) = \mathcal{L}(\mathcal{MH}(\mathcal{Q}))$ [92]. An even stronger property holds:

- $\mathcal{MH}(\mathcal{Q})$ $(n, 1)$-delayed/fair simulates $\mathcal{Q}$. See Lemma 5.7.6 below.
- $\mathcal{Q}$ $(1, 1)$-fair simulates $\mathcal{MH}(\mathcal{Q})$. See Lemma 5.7.7 below.

Since multipebble simulation implies language inclusion (cf. Theorem 5.4.8), $\mathcal{L}(\mathcal{Q}) = \mathcal{L}(\mathcal{MH}(\mathcal{Q}))$ follows immediately.

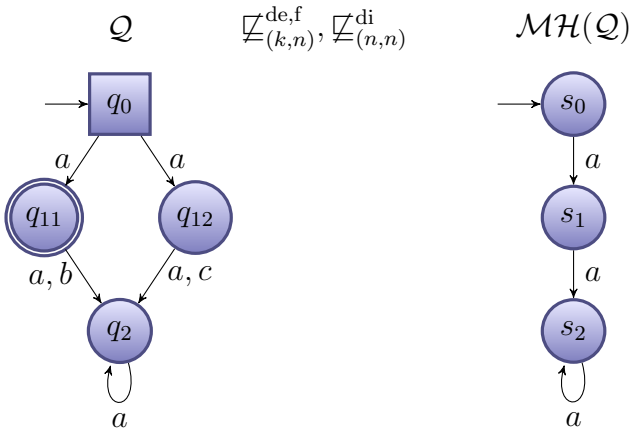
Lemma 5.7.6. *For any ABA* $\mathcal{Q}$ *and* $x \in \{\mathrm{de}, \mathrm{f}\}$*,* $\mathcal{Q} \sqsubseteq^x_{(n,1)} \mathcal{MH}(\mathcal{Q})$*.*

Proof. By Lemma 5.4.6, it suffices to prove $\mathcal{Q} \sqsubseteq^{\mathrm{de}}_{(n,1)} \mathcal{MH}(\mathcal{Q})$. Duplicator uses a strategy σ which mimics exactly the MH-construction: Duplicator maximally hedges her bets on left universal pebbles in $\mathcal{Q}$, and the right pebble in $\mathcal{MH}(\mathcal{Q})$ is moved by copying Spoiler's moves from left existential pebbles in $\mathcal{Q}$. Formally, there exists a strategy σ which maintains the following invariant: If at round k the current configuration is $\langle \mathbf{q}_k, (\mathbf{q}'_k, \mathbf{o}_k) \rangle$, then $\mathbf{q}_k = \mathbf{q}'_k$.

We now argue that σ is winning for Duplicator. Let $\pi = \langle \mathbf{q}_0, (\mathbf{q}_0, \mathbf{o}_0) \rangle \overset{a_0}{\Longrightarrow} \langle \mathbf{q}_1, (\mathbf{q}_1, \mathbf{o}_1) \rangle \overset{a_1}{\Longrightarrow} \cdots$ be any σ-conform play. Define the sequence of indices $\{i_j\}_{j \geq 0}$ as $i_0 = 0$ and, inductively, i_{j+1} is the least index $> i_j$ s.t. $\mathbf{q}_{i_j} \Longrightarrow_F^{\forall} \mathbf{q}_{i_{j+1}}$ If no such index exists, take $i_{j+1} = i_{j+2} = \cdots = \omega$. By the definition of the MH-construction, $\mathbf{o}_{i_j} = \emptyset$ for any $i_j < \omega$.

By the definition of delayed simulation, assume k and h are rounds s.t. $\mathbf{q}_h \Longrightarrow_F^{\forall} \mathbf{q}_k$. Let i_j be the largest index $\leq h$ s.t. $\mathbf{q}_{i_j} \Longrightarrow_F^{*} \mathbf{q}_k$. It exists by Lemma 5.4.1. By the maximality of i_j and by the minimality in the definition of i_{j+1}, $i_j \leq h \leq i_{j+1} \leq k < \omega$. By the MH-construction, $\mathbf{o}_{i_{j+1}} = \emptyset$. Therefore, there exists a finite $i_{j+1} \geq h$ s.t. $(\mathbf{q}_{i_{j+1}}, \mathbf{o}_{i_{j+1}}) \in F'$. □

Example 5.7.1 - Lemma 5.7.6 is tight



Consider the automaton $\mathcal{Q}$ above and its de-universalization $\mathcal{MH}(\mathcal{Q})$. The states in $\mathcal{MH}(\mathcal{Q})$ are $s_0 = (\{q_0\}, \{q_0\})$, $s_1 = (\{q_{11}, q_{12}\}, \{q_{12}\})$ and $s_2 = (\{q_2\}, \{q_2\})$. We have

1) $\mathcal{Q} \not\sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{MH}(\mathcal{Q})$, therefore n pebbles on the left are necessary in Lemma 5.7.6. Indeed, Duplicator needs at least two pebbles on the left to win, in order to prevent Spoiler from playing either action b or c in the second round.

2) $\mathcal{Q} \not\sqsubseteq^{\mathrm{di}}_{(n,n)} \mathcal{MH}(\mathcal{Q})$, thus delayed or fair simulation are necessary in Lemma 5.7.6. Indeed, in the first round Spoiler plays action a and Duplicator splits pebbles on the left. The next configuration is $\langle \mathbf{q}_1, s_1 \rangle$, with $\mathbf{q}_1 := \{q_{11}, q_{12}\}$. But $\mathbf{q}_1 \cap F \neq \emptyset$ and $s_1 \notin F$. Thus, Duplicator loses the direct simulation game.

Lemma 5.7.7. $\mathcal{MH}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{Q}$.

Proof. Duplicator has a winning strategy σ maintaining the following invariant: At round k, let the current partial play be $\pi_k = \rho_k \times \rho'_k$, where

$$\rho_k = (\mathbf{q}_0, \mathbf{o}_0) \xrightarrow{a_0} (\mathbf{q}_1, \mathbf{o}_1) \xrightarrow{a_1} \cdots \xrightarrow{a_{k-1}} (\mathbf{q}_k, \mathbf{o}_k)$$
$$\rho'_k = q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{k-1}} q_k$$

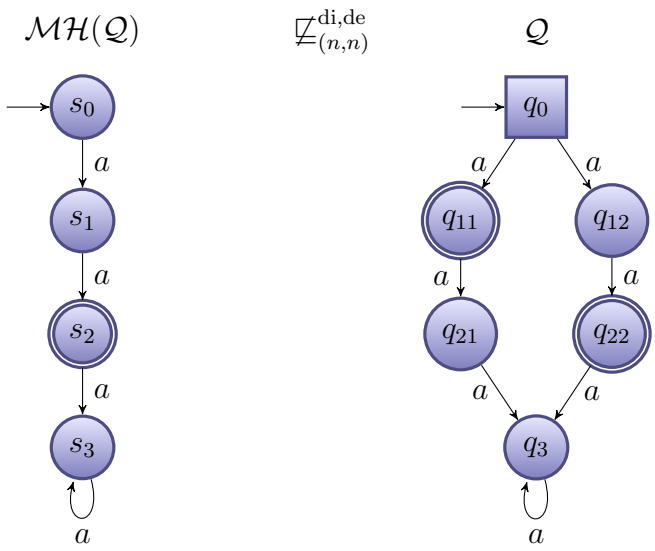
If π_k is σ-conform, then $\rho'_k \in \rho_k$ (with abuse of notation, we actually mean $\rho'_k \in \mathbf{q}_0 \xrightarrow{a_0} \mathbf{q}_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{k-1}} \mathbf{q}_k$). In particular, $q_k \in \mathbf{q}_k$, i.e., Duplicator can always force the right pebble to be somewhere in $\mathbf{q}_k$. Clearly, the invariant holds for the initial configuration: For $k = 0$, $(\mathbf{s}_0, \mathbf{o}_0) = (\{q\}, \{q\} \setminus F)$ and $q_0 = q$. Inductively, assume that the invariant $\rho'_k \in \rho_k$ holds for $k \geq 0$. Duplicator ensures the invariant holds in the next round $k+1$. Assume that Spoiler chooses action $(\mathbf{q}_k, \mathbf{o}_k) \xrightarrow{a_k} (\mathbf{q}_{k+1}, \mathbf{o}_{k+1})$. We consider two cases:

- $q_k \in E$: By the MH-construction, there exists $q' \in \mathbf{q}_{k+1}$ s.t. $q_k \xrightarrow{a_k} q'$. Duplicator moves the right pebble from q_k to $q_{k+1} := q'$,
- $q_k \in U$: By the MH-construction, $\Delta(q_k, a_k) \subseteq \mathbf{q}_{k+1}$. Hence, no matter which transition $q_k \xrightarrow{a_k} q_{k+1}$ Spoiler chooses on the right, $q_{k+1} \in \mathbf{q}_{k+1}$.

In both cases, assume transition $q_k \xrightarrow{a_k} q_{k+1}$ is taken on the right. Let $\rho_{k+1} = \rho_k \xrightarrow{a_k} (\mathbf{q}_{k+1}, \mathbf{o}_{k+1})$ and $\rho'_{k+1} = \rho'_k \xrightarrow{a_k} q_{k+1}$. The invariant $\rho'_{k+1} \in \rho_{k+1}$ is preserved.

We argue that this invariant-preserving strategy σ is winning for Duplicator. Let $\pi_\omega = \rho_\omega \times \rho'_\omega$ be any σ-conform play, and let $\pi = \mathbf{q}_0 \overset{a_0}{\Longrightarrow} \mathbf{q}_1 \overset{a_1}{\Longrightarrow} \cdots$ be the projection of ρ_ω to the first component. Assume ρ_ω is fair. I.e., $(\mathbf{q}_j, \mathbf{o}_j)$ is accepting for infinitely many j's. By Lemma 5.7.4, $\mathbf{q}_i \Longrightarrow^{\forall}_F \mathbf{q}_j$ holds for infinitely many i's and j's with $i \leq j$. Therefore, π is universally fair. By the invariant, $\rho'_\omega \in \pi$. Since π is universally fair, by Lemma 5.4.4, ρ'_ω is universally fair. In particular, it is fair. $\square$

Example 5.7.2 - Lemma 5.7.7 is tight



Consider the automaton $\mathcal{Q}$ above and its de-universalization $\mathcal{MH}(\mathcal{Q})$ on the left. We show $\mathcal{MH}(\mathcal{Q}) \not\sqsubseteq^{x}_{(n,n)} \mathcal{Q}$, for $x \in \{\mathrm{di}, \mathrm{de}\}$. Therefore, fair simulation is necessary in Lemma 5.7.7.

Automaton $\mathcal{MH}(\mathcal{Q})$ has states $s_0 = (\{q_0\}, \{q_0\})$, $s_1 = (\{q_{11}, q_{12}\}, \{q_{12}\})$, $s_2 = (\{q_{21}, q_{22}\}, \emptyset)$, and $s_3 = (\{q_3\}, \{q_3\})$. Duplicator does never take any decision during the game. Indeed, automaton $\mathcal{MH}(\mathcal{Q})$ does not have universal states by construction, and automaton $\mathcal{Q}$ has only deterministic states, except for the initial state q_0, which is universal. In particular, she can never split pebbles. Hence, $\sqsubseteq^{x}_{(n,n)} = \sqsubseteq^{x}_{(1,1)}$.

Suppose Spoiler plays as follows

$$\pi = \langle s_0, q_0 \rangle \xrightarrow{a} \langle s_1, q_{11} \rangle \xrightarrow{a} \langle s_2, q_{21} \rangle (\xrightarrow{a} \langle s_3, q_3 \rangle)^{\omega}$$

At round $k = 2$, $s_2 \in F'$, but $q_{21} \notin F$. Therefore, Spoiler wins w.r.t. the direct winning condition. Moreover, after round k the right pebble remains forever in the non-accepting state $q_3 \notin F$. Thus, Spoiler wins also w.r.t. the delayed winning condition.

5.7.1.3 Multipebble simulations and the Miyano-Hayashi-construction

Since the de-universalization of an automaton $\mathcal{Q}$ does not have universal states, Duplicator never needs extra pebbles on the left.

Lemma 5.7.8. *For ABAs $\mathcal{Q}$ and $\mathcal{S}$, and for any $k_0, k_1 > 0$, if $\mathcal{MH}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(k_0,k_1)} \mathcal{S}$, then*

$\mathcal{MH}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(1,k_1)} \mathcal{S}$,

We now prove that the MH-construction characterizes $(n, 1)$-fair simulation.

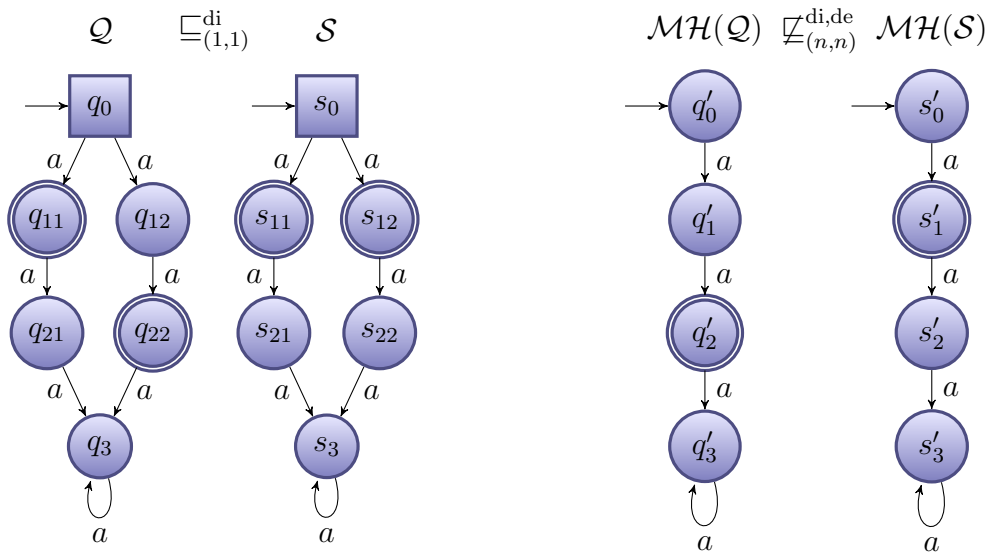
Proof. Proof (of Theorem 5.7.1) We use transitivity, cf. Theorem 5.6.1.

$$\begin{aligned}
&\mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(n,1)} \mathcal{S} \implies \\
&\quad\implies \mathcal{MH}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(n,1)} \mathcal{S} \sqsubseteq^{\mathrm{f}}_{(n,1)} \mathcal{MH}(\mathcal{S}) && \text{by Lemma 5.7.7 and 5.7.6} \\
&\quad\implies \mathcal{MH}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(n,1)} \mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(n,1)} \mathcal{S} \sqsubseteq^{\mathrm{f}}_{(n,1)} \mathcal{MH}(\mathcal{S}) && \text{by Theorem 5.4.7} \\
&\quad\implies \mathcal{MH}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(n,1)} \mathcal{MH}(\mathcal{S}) && \text{by Theorem 5.6.1} \\
&\quad\implies \mathcal{MH}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{MH}(\mathcal{S}) && \text{by Lemma 5.7.8}
\end{aligned}$$

$$\begin{aligned}
&\mathcal{MH}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{MH}(\mathcal{S}) \implies \\
&\quad\implies \mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(n,1)} \mathcal{MH}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{MH}(\mathcal{S}) \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{S} && \text{by Lemma 5.7.6 and 5.7.7} \\
&\quad\implies \mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(n,1)} \mathcal{MH}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(n,1)} \mathcal{MH}(\mathcal{S}) \sqsubseteq^{\mathrm{f}}_{(n,1)} \mathcal{S} && \text{by Theorem 5.4.7} \\
&\quad\implies \mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(n,1)} \mathcal{S} && \text{by Theorem 5.6.1} \qquad \square
\end{aligned}$$

Theorem 5.7.1 does not hold for either direct or delayed simulation [4].

Example 5.7.3 - The MH-construction preserves neither direct, nor delayed simulation



[4] Unlike as incorrectly claimed in [55].

Consider the two automata $\mathcal{Q}$ and $\mathcal{S}$ on the left above, and their de-universalizations $\mathcal{MH}(\mathcal{Q})$ and $\mathcal{MH}(\mathcal{S})$ on the right. For $x \in \{\mathrm{di}, \mathrm{de}\}$, $\mathcal{Q} \sqsubseteq^{x}_{(1,1)} \mathcal{S}$, but $\mathcal{MH}(\mathcal{Q}) \not\sqsubseteq^{x}_{(n,n)} \mathcal{MH}(\mathcal{S})$. Therefore, the MH-construction does not preserve either direct, or delayed simulation.

- $\mathcal{Q} \sqsubseteq^{\mathrm{di}}_{(1,1)} \mathcal{S}$: Suppose Spoiler on the right takes transition $s_0 \xrightarrow{a} s_{1i}$, for $i \in \{1, 2\}$. Then, Duplicator on the left takes transition $q_0 \xrightarrow{a} q_{11}$. The winning condition is satisfied because s_{1i} is accepting for any $i \in \{1, 2\}$. From configuration $\langle q_{11}, s_{1i}\rangle$, Duplicator can ensure that no other accepting state is visited in $\mathcal{Q}$ on the left.

- $\mathcal{MH}(\mathcal{Q}) \not\sqsubseteq^{x}_{(n,n)} \mathcal{MH}(\mathcal{S})$: The macrostates in $\mathcal{MH}(\mathcal{Q})$ are defined as $q'_0 = (\{q_0\}, \{q_0\})$, $q'_1 = (\{q_{11}, q_{12}\}, \{q_{12}\})$, $q'_2 = (\{q_{21}, q_{22}\}, \emptyset)$ and $q'_3 = (\{q_3\}, \{q_3\})$. Similarly, the macrostates in $\mathcal{MH}(\mathcal{S})$ are defined as $s'_0 = (\{s_0\}, \{s_0\})$, $s'_1 = (\{s_{11}, s_{12}\}, \emptyset)$, $s'_2 = (\{s_{21}, s_{22}\}, \{s_{21}, s_{22}\})$ and $s'_3 = (\{s_3\}, \{s_3\})$. Since both automata are deterministic, multipebble simulation collapses to 1-pebble simulation. Spoiler wins by playing the infinite word a^ω: In the two induced left path $\pi = q'_0 \xrightarrow{a} q'_1 \xrightarrow{a} q'_2(\xrightarrow{a} q'_3)^\omega$ and right path $\pi' = s'_0 \xrightarrow{a} s'_1 \xrightarrow{a} s'_2(\xrightarrow{a} s'_3)^\omega$, $q'_2 \in F'$, but neither s'_2 nor s'_3 is accepting.

5.7.2 Characterizing $(1, n)$-fair simulation

Theorem 5.7.1 provides a characterization of left pebbles in terms of the Miyano-Hayashi subset-like construction. In this section, we show that a construction dual to MH characterizes pebbles on the right (i.e., $(1, n)$-fair simulation) in a similar way. We call it the *fair subset construction*; for an automaton, $\mathcal{A}$, let $\mathcal{F}(\mathcal{A})$ be the automaton obtained by applying the construction.

Theorem 5.7.9. *Let $\mathcal{Q}$ and $\mathcal{S}$ be two uniform ABAs, and let $n = |\mathcal{S}|$. Then,*

$$\mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(1,n)} \mathcal{S} \iff \mathcal{F}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{F}(\mathcal{S})$$

The restriction to uniform automata (cf. Section 5.5) is not essential, but, for simplicity, we only define the fair subset construction in this case. In Section 5.7.2.1 we introduce the construction, in Section 5.7.2.2 we prove its correctness, and in Section 5.7.2.3 we prove the theorem. We show with examples the tightness of the stated properties.

5.7.2.1 The fair subset construction

The construction is defined for uniform automata (cf. Section 5.5). A general definition for arbitrary automata can be given, but it would require a more complex model of alternation not considered in this thesis.

Definition 5.7.10 (The fair subset construction)**.** *Given a uniform ABA $\mathcal{Q}$,*

$$\mathcal{Q} = (Q, \Sigma, q_I, \Delta, F, E, U)$$

the fair subset construction *yields a uniform ABA $\mathcal{F}(\mathcal{Q})$,*

$$\mathcal{F}(\mathcal{Q}) = (Q', \Sigma, \mathbf{q}_I, \Delta, F', E', U')$$

where macrostates in $Q' \subseteq 2^Q \times 2^Q$ consist of pairs of subsets of Q, the initial macrostate is $\mathbf{q}_I := (\{q_I\}, \{q_I\} \cap F)$, the set of accepting macrostates is

$$F' = \{(\mathbf{q}, \mathbf{g}) \mid \mathbf{g} = \mathbf{q} \neq \emptyset\}$$

and $((\mathbf{q}, \mathbf{g}), a, (\mathbf{q}', \mathbf{g}')) \in \Delta'$ iff either

- $\mathbf{q} \subseteq E$ *and*

$$\mathbf{q}' \subseteq \bigcup \Delta(\mathbf{q}, a)$$
$$\mathbf{g}' = \begin{cases} (\bigcup \Delta(\mathbf{g}, a) \cup F) \cap \mathbf{q}' & \text{if } \mathbf{g} \neq \mathbf{q} \\ \mathbf{q}' \cap F & \text{otherwise.} \end{cases}$$

  Intuitively, $\mathbf{q}'$ is obtained by selecting a possibly empty set of a-successors from every state $q \in \mathbf{q}$; $\mathbf{g}'$ is updated to keep track of which states have visited an accepting state.

- $\mathbf{q} \subseteq U$ *and*

$$\mathbf{q}' \in \bigotimes \Delta(\mathbf{q}, a)$$
$$\mathbf{g}' = \begin{cases} \mathbf{q}' \downarrow_{\Delta(\mathbf{g}, a)} \cup (\mathbf{q}' \cap F) & \text{if } \mathbf{g} \neq \mathbf{q} \\ \mathbf{q}' \cap F & \text{otherwise.} \end{cases}$$

  Intuitively, $\mathbf{q}'$ is obtained by selecting exactly one a-successor from every state $q \in \mathbf{q}$; like above, $\mathbf{g}'$ is updated to keep track of which states have visited an accepting state.

(See Section 2.2 for the notation $\downarrow$).) Since $\mathcal{Q}$ is uniform, the two cases above are exhaustive. Accordingly, a macrostate $(\mathbf{q}, \mathbf{g})$ is existential $(\mathbf{q}, \mathbf{g}) \in E'$ iff $\mathbf{q} \subseteq E$, and universal $(\mathbf{q}, \mathbf{g}) \in U'$ iff $\mathbf{q} \subseteq U$.

Remark 5.7.11. Unlike the Miyano-Hayashi procedure (see Section 5.7.1.1), the fair subset construction does not actually perform alternation removal. That is, if $\mathcal{Q}$ is an alternating automaton, then $\mathcal{F}(\mathcal{Q})$ is still alternating. The sole purpose of the fair subset construction is to describe multipebble simulations in terms of 1-pebble simulations.

Notice that, if $\mathcal{Q}$ is uniform, so it is $\mathcal{F}(\mathcal{Q})$. Also, it always holds that, if $(\mathbf{q}, \mathbf{g})$ is a reachable configuration, then $\mathbf{g} \subseteq \mathbf{q}$ and $\mathbf{q} \cap F \subseteq \mathbf{g}$. Notice that $\mathbf{g}'$ depends uniquely on $\mathbf{q}$, $\mathbf{g}$, $\mathbf{q}'$ and input symbol a. In pictures, we do not represent the sink state $(\emptyset, \emptyset)$.

The following lemma is immediate from the definition of the fair subset construction.

Lemma 5.7.12. *Let $\pi = (\mathbf{q}_0, \mathbf{g}_0) \xrightarrow{a_0} (\mathbf{q}_1, \mathbf{g}_1) \xrightarrow{a_1} \cdots$ be any path in $\mathcal{F}(\mathcal{Q})$.*

a) If $(\mathbf{q}_i, \mathbf{g}_i), (\mathbf{q}_j, \mathbf{g}_j) \in F'$ with $i \leq j$, then $\mathbf{q}_i \Longrightarrow_F^{\exists} \mathbf{q}_j$.

b) If $\mathbf{q}_i \Longrightarrow_F^{\exists} \mathbf{q}_j \Longrightarrow_F^{\exists} \mathbf{q}_k$, then there exists k' s.t. $i \leq k' \leq k$ and $(\mathbf{q}_{k'}, \mathbf{g}_{k'}) \in F'$.

5.7.2.2 Correctness of the construction

The fair subset construction preserves the language of the automaton: $\mathcal{L}(\mathcal{Q}) = \mathcal{L}(\mathcal{F}(\mathcal{Q}))$. Similarly to the MH-construction, we show the following stronger properties:

- $\mathcal{F}(\mathcal{Q})$ $(1,1)$-direct simulates $\mathcal{Q}$. See Lemma 5.7.13 below.
- $\mathcal{Q}$ $(1,n)$-fair simulates $\mathcal{F}(\mathcal{Q})$. See Lemma 5.7.14 below.

Lemma 5.7.13. *For a uniform $\mathcal{Q}$, $\mathcal{Q} \sqsubseteq^{\mathrm{di}}_{(1,1)} \mathcal{F}(\mathcal{Q})$.*

Proof. The winning strategy for Duplicator is to simply mimic Spoiler's behavior. Formally, if $\langle q_k, (\mathbf{q}_k, \mathbf{g}_k)\rangle$ is the current configuration at round k, then Duplicator maintains the following invariant: $\mathbf{q}_k = \{q_k\}$. This is clearly winning, since if $q_k \in F$, then $\{q_k\} \cap F \subseteq \mathbf{g}_k \subseteq \{q_k\}$, therefore $\mathbf{g}_k = \mathbf{q}_k$ and $(\mathbf{q}_k, \mathbf{g}_k) \in F'$.

Since $\mathcal{F}(\mathcal{Q})$ is uniform, we need to consider only purely existential and purely universal rounds. In an existential round, $\mathbf{q}_k \subseteq E$. If Spoiler moves as $q_k \xrightarrow{a} q_{k+1}$, Duplicator replies as $(\mathbf{q}_k, \mathbf{g}_k) = (\{q_k\}, \mathbf{g}_k) \xrightarrow{a} (\mathbf{q}_{k+1}, \mathbf{g}_{k+1}) := (\{q_{k+1}\}, \mathbf{g}_{k+1})$, where $\mathbf{g}_{k+1} := \{q_{k+1}\} \cap F$.

In a universal round, $\mathbf{q}_k \subseteq U$. If Spoiler moves as $(\mathbf{q}_k, \mathbf{g}_k) = (\{q_k\}, \mathbf{g}_k) \xrightarrow{a} (\mathbf{q}_{k+1}, \mathbf{g}_{k+1})$, then $\mathbf{q}_{k+1} \in \bigotimes \Delta(\{q_k\}, a)$, implying $\mathbf{q}_{k+1} = \{q'\}$, for some $q' \in \Delta(q_k, a)$. Thus, Duplicator replies as $q_k \xrightarrow{a} q_{k+1}$, with $q_{k+1} := q'$. $\square$

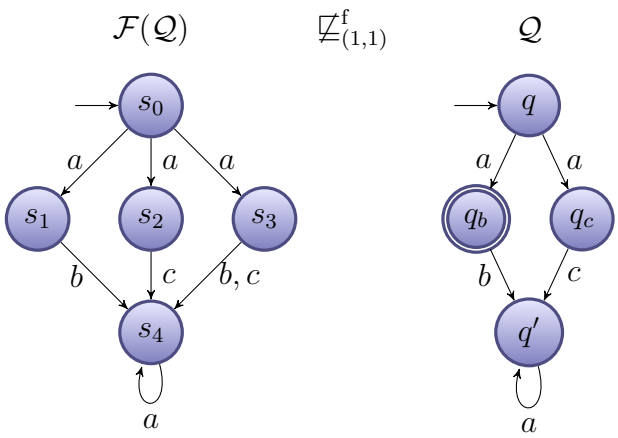
Lemma 5.7.14. *For a uniform $\mathcal{Q}$, $\mathcal{F}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(1,n)} \mathcal{Q}$, where $n = |\mathcal{Q}|$.*

Proof. The winning strategy for Duplicator is to simply mimic Spoiler's action, possibly using multiple pebbles. Let $\langle(\mathbf{q}_k, \mathbf{g}_k), \mathbf{q}'_k\rangle$ be the current configuration at round k. Duplicator maintains the following invariant: $\mathbf{q}'_k = \mathbf{q}_k$. This is winning, since, if Spoiler builds a fair path, then $(\mathbf{q}_k, \mathbf{g}_k) \in F'$ infinitely often, and, by Lemma 5.7.12(a), $\mathbf{q}_i \Longrightarrow^{\exists}_F \mathbf{q}_j$ for arbitrarily large i; that is, right pebbles are existentially good infinitely often and Duplicator builds an existentially fair multipath.

Since $\mathcal{F}(\mathcal{Q})$ is uniform, we only need to consider purely existential and purely universal rounds. In an existential round, $\mathbf{q}_k \subseteq E$. If Spoiler moves as $(\mathbf{q}_k, \mathbf{g}_k) \xrightarrow{a} (\mathbf{q}_{k+1}, \mathbf{g}_{k+1})$, then, by the definition of fair subset construction, $\mathbf{q}_{k+1} \subseteq \bigcup \Delta(\mathbf{q}_k, a)$. Duplicator replies on the right as $\mathbf{q}_k \overset{a}{\Longrightarrow} \mathbf{q}_{k+1}$, thus preserving the invariant. (Without loss of generality, we can assume that $\mathbf{q}_{k+1} \neq \emptyset$, otherwise Spoiler would trivially lose as she could not be accepting anymore in $\mathcal{F}(\mathcal{Q})$.)

In a universal round, $\mathbf{q}_k \subseteq U$. If Spoiler performs action a and moves the right pebbles from $\mathbf{q}_k$ to $\mathbf{q}_{k+1}$, then $\mathbf{q}_{k+1} \in \bigotimes \Delta(\mathbf{q}_k, a)$. Duplicator replies on the left as $(\mathbf{q}_k, \mathbf{g}_k) \xrightarrow{a} (\mathbf{q}_{k+1}, \mathbf{g}_{k+1})$, for some $\mathbf{g}_{k+1}$, which is uniquely determined by the fair subset construction. □

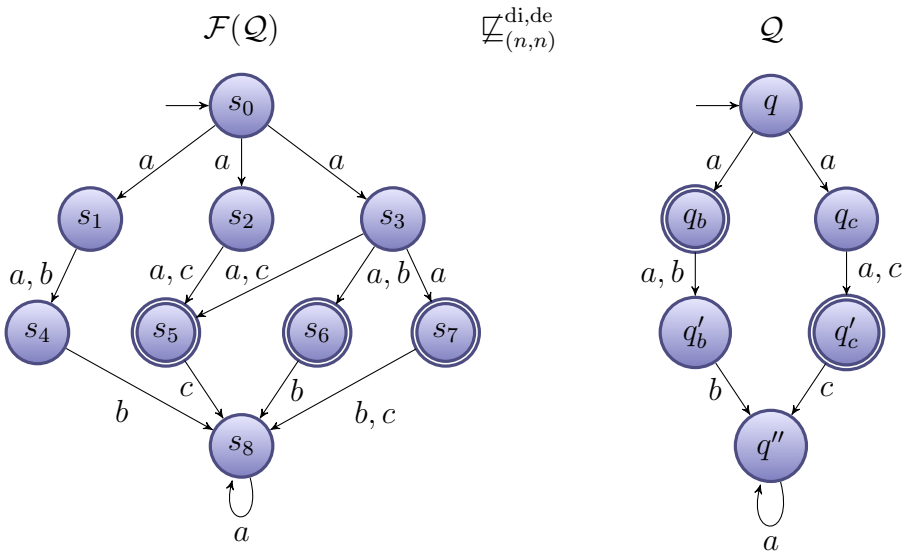
Example 5.7.4 - n pebbles are needed in Lemma 5.7.14



Consider the automata above. We have $\mathcal{F}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(1,2)} \mathcal{Q}$, but $\mathcal{F}(\mathcal{Q}) \not\sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{Q}$. On the left $\mathcal{F}(\mathcal{Q})$ is s.t. $s_0 = (\{q\}, \{q\})$, $s_1 = (\{q_b\}, \{q_b\})$, $s_2 = (\{q_c\}, \{q_c\})$, $s_3 = (\{q_b, q_c\}, \{q_b, q_c\})$, $s_4 = (\{q'\}, \{q'\})$. Spoiler wins by taking transition $s_0 \xrightarrow{a} s_3$.

Duplicator has to choose either transition $q \xrightarrow{a} q_b$ or $q \xrightarrow{a} q_c$. In either case, Spoiler can play either c or b, respectively, and Duplicator loses.

Example 5.7.5 - Fair simulation is needed in Lemma 5.7.14



We have $\mathcal{F}(\mathcal{Q}) \not\sqsubseteq^{\mathrm{di,de}}_{(n,n)} \mathcal{Q}$. On the left $\mathcal{F}(\mathcal{Q})$ is s.t. $s_0 = (\{q\}, \emptyset)$, $s_1 = (\{q_b\}, \{q_b\})$, $s_2 = (\{q_c\}, \emptyset)$, $s_3 = (\{q_b, q_c\}, \{q_b\})$, $s_4 = (\{q'_b\}, \emptyset)$, $s_5 = (\{q'_c\}, \{q'_c\})$, $s_6 = (\{q'_b\}, \{q'_b\})$, $s_7 = (\{q'_b, q'_c\}, \{q'_b, q'_c\})$, $s_8 = (\{q''\}, \emptyset)$. Spoiler wins as follows: At round 0, she takes transition $s_0 \xrightarrow{a} s_3$. Duplicator is forced to split pebbles on the right by taking transition $\{q\} \overset{a}{\Longrightarrow} \{q_b, q_c\}$. From configuration $\langle s_3, \{q_b, q_c\}\rangle$, Spoiler takes transition $s_3 \xrightarrow{a} s_7$. Duplicator is still forced to use both pebbles, and she takes transition $\{q_b, q_c\} \overset{a}{\Longrightarrow} \{q'_b, q'_c\}$. From configuration $\langle s_7, \{q'_b, q'_c\}\rangle$, Spoiler plays transition $s_7 \xrightarrow{b} s_8$. Duplicator has to drop the pebble from q'_c, and to take transition $\{q'_b, q'_c\} \overset{b}{\Longrightarrow} \{q''\}$. The following rounds are obvious as Spoiler is forced to play a^ω and configuration $\langle s_8, \{q''\}\rangle$ repeats indefinitely. Spoiler wins because $s_7 \in F'$ at round 2, but it is not the case that $\{q'_b, q'_c\} \Longrightarrow^{\exists}_{F} \{q''\}$ for any later round.

5.7.2.3 Multipebble simulations and the fair subset construction

Intuitively, Duplicator never needs to split pebbles on a fair subset automaton $\mathcal{F}(\mathcal{S})$, since, by the definition of the fair subset construction, she can always find a successor "large enough" where to place her single pebble. In the lemma below, we formalize

this intuition. (The corresponding statement for the MH-construction holds for trivial reasons, cf. Lemma 5.7.8.)

Lemma 5.7.15. *For an ABA $\mathcal{Q}$ and a uniform ABA $\mathcal{S}$, let $k_0, k_1 > 0$, if $\mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(k_0,k_1)} \mathcal{F}(\mathcal{S})$, then $\mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(k_0,1)} \mathcal{F}(\mathcal{S})$.*

Proof. Let $n = |\mathcal{F}(\mathcal{S})|$. It suffices to prove the statement for $k_1 = n$. Let $\mathcal{G} = \mathbb{G}^{\mathrm{f}}_{(k_0,n)}(\mathcal{Q}, \mathcal{F}(\mathcal{S}))$ and $\mathcal{G}' = \mathbb{G}^{\mathrm{f}}_{(k_0,1)}(\mathcal{Q}, \mathcal{S})$, and assume that Duplicator wins $\mathcal{G}$. We show that she wins $\mathcal{G}'$ as well.

Let the current configuration of $\mathcal{G}$ and of $\mathcal{G}'$ be $\langle \mathbf{q}_k, \mathbf{x}_k \rangle$ and $\langle \mathbf{q}'_k, (\mathbf{s}_k, \mathbf{g}_k) \rangle$, respectively, where $\mathbf{x}_k = \{(\mathbf{s}_{j,k}, \mathbf{g}_{j,k})\}_{j \leq n}$. Duplicator wins by maintaining the following invariant: $\mathbf{q}_k = \mathbf{q}'_k$ and $\mathbf{s}_k = \bigcup \{\mathbf{s}_{j,k}\}_{j \leq n}$. This is achieved as follows. The left pebbles in $\mathcal{G}'$ are moved isomorphically to left pebbles in $\mathcal{G}$. We discuss how to synchronize right pebbles. When Duplicator in $\mathcal{G}$ splits right pebbles to several successors, Duplicator in $\mathcal{G}'$ moves the unique right pebble to the union of those, which exists by the definition of fair subset construction. Similarly, when Spoiler in $\mathcal{G}'$ moves the unique right pebble to some successor, this fixes a successor for each right pebble in $\mathcal{G}$, and this will be the move of Spoiler in the latter game.

Assumer Spoiler plays action a_k in $\mathcal{G}$. Then, action a_k is also played by Spoiler in $\mathcal{G}'$. Since $\mathcal{S}$ is uniform, $\mathcal{F}(\mathcal{S})$ is uniform as well, and right pebbles in $\mathcal{G}$ are either all existential or all universal.

- $\{\mathbf{s}_{j,k}\}_{j \leq n} \subseteq 2^E$. Assume Duplicator moves right pebbles in $\mathcal{G}$ as

  $$\{(\mathbf{s}_{j,k}, \mathbf{g}_{j,k})\}_{j \leq n} \stackrel{a_k}{\Longrightarrow} \{(\mathbf{s}_{j,k+1}, \mathbf{g}_{j,k+1})\}_{j \leq n}$$

  By definition, for each $\mathbf{s}_{j,k+1}$ there exists $\mathbf{s}_{j',k}$ s.t. $\mathbf{s}_{j',k} \stackrel{a_k}{\Longrightarrow} \mathbf{s}_{j,k+1}$ (in $\mathcal{Q}$). By the invariant, $\mathbf{s}_k = \bigcup \{\mathbf{s}_{j,k}\}_{j \leq n}$. Therefore, by letting $\mathbf{s}_{k+1} := \bigcup \{\mathbf{s}_{j,k+1}\}_{j \leq n}$, we have $\mathbf{s}_k \stackrel{a_k}{\Longrightarrow} \mathbf{s}_{k+1}$. Thus, the following is a valid transition in $\mathcal{F}(\mathcal{Q})$

  $$(\mathbf{s}_k, \mathbf{g}_k) \xrightarrow{a_k} (\mathbf{s}_{k+1}, \mathbf{g}_{k+1})$$

  for some $\mathbf{g}_{k+1}$ uniquely fixed by the fair subset construction. Thus, Duplicator moves the right pebble in $\mathcal{G}'$ as above, from $(\mathbf{s}_k, \mathbf{g}_k)$ to $(\mathbf{s}_{k+1}, \mathbf{g}_{k+1})$.

- $\{\mathbf{s}_{j,k}\}_{j \leq n} \subseteq 2^U$. Assume Spoiler moves the unique right pebble in $\mathcal{G}'$ as

  $$(\mathbf{s}_k, \mathbf{g}_k) \xrightarrow{a_k} (\mathbf{s}_{k+1}, \mathbf{g}_{k+1})$$

  By the definition of the fair subset construction, $\mathbf{s}_{k+1} \in \bigotimes \Delta(\mathbf{s}_k, a_k)$. For each macrostate $\mathbf{s}_{j,k} \subseteq \mathbf{s}_k$, the move above induces a successor macrostate

$\mathbf{s}_{j,k+1} \in \bigotimes \Delta(\mathbf{s}_{j,k}, a_k)$ by letting $\mathbf{s}_{j,k+1} = \mathbf{s}_{k+1} \downarrow_{\Delta(\mathbf{s}_{j,k}, a_k)}$ (see Section 2.2 for the notation $\downarrow$). Therefore, Spoiler in $\mathcal{G}$ moves pebbles from $\{(\mathbf{s}_{j,k}, \mathbf{g}_{j,k})\}_{j \leq n}$ to $\{(\mathbf{s}_{j,k+1}, \mathbf{g}_{j,k+1})\}_{j \leq n}$ as defined above, for sets $\{\mathbf{g}_{j,k+1}\}_{j \leq n}$ uniquely fixed by the fair subset construction. Also in this case, $\bigcup\{(\mathbf{s}_{j,k+1})\}_{j \leq n} = \mathbf{s}_{k+1}$.

We now argue that Duplicator wins $\mathcal{G}'$. Let $w = a_0 a_1 \cdots$, and let $\pi = \pi_0 \times \pi_1$ and $\pi' = \pi_0 \times \pi_1'$ be the two resulting plays in $\mathcal{G}$ and in $\mathcal{G}'$, respectively, where

$$\begin{aligned}
\pi_0 &= \mathbf{q}_0 \overset{a_0}{\Longrightarrow} \mathbf{q}_1 \overset{a_1}{\Longrightarrow} \cdots \\
\pi_1 &= \mathbf{x}_0 \overset{a_0}{\Longrightarrow} \mathbf{x}_1 \overset{a_1}{\Longrightarrow} \cdots, \text{ with } \mathbf{x}_k = \{(\mathbf{s}_{j,k}, \mathbf{g}_{j,k})\}_{j \leq n} \\
\pi_1' &= (\mathbf{s}_0, \mathbf{g}_0) \xrightarrow{a_0} (\mathbf{s}_1, \mathbf{g}_1) \xrightarrow{a_1} \cdots
\end{aligned}$$

By the invariant, $\mathbf{s}_k = \bigcup\{\mathbf{s}_{j,k} \mid (\mathbf{s}_{j,k}, \mathbf{g}_{j,k}) \in \mathbf{x}_k\}$. Assume π_0 is universally fair. We show that π_1' is fair.

Since Duplicator wins $\mathcal{G}$ and π_0 is universally fair, π_1 is existentially fair. Therefore, in $\mathcal{G}$, right pebbles are good infinitely often. Let $i_0 \leq i_1 \leq i_2$ be s.t.

$$\mathbf{x}_{i_0} \Longrightarrow_F^{\exists} \mathbf{x}_{i_1} \Longrightarrow_F^{\exists} \mathbf{x}_{i_2}$$

Arbitrarily large such numbers exist since π_1 is existentially fair. By definition, for any right pebble $x'' = (\mathbf{s}'', \mathbf{g}'') \in \mathbf{x}_{i_2}$, there exists a right pebble $x' = (\mathbf{s}', \mathbf{g}') \in \mathbf{x}_{i_1}$ s.t. $x' \longrightarrow_F^* x''$. In turn, this means that there exist k' and an accepting pebble $y' = (\mathbf{t}', \mathbf{f}') \in \mathbf{x}_{k'}$ s.t. $i_1 \leq k' \leq i_2$ and $x' \xrightarrow{w_0} y' \xrightarrow{w_1} x''$, where $w_0 = w[i_1, k'-1]$ and $w_1 = w[k', i_2 - 1]$. Similarly, for any such $x' = (\mathbf{s}', \mathbf{g}') \in \mathbf{x}_{i_1}$, there exists $x = (\mathbf{s}, \mathbf{g}) \in \mathbf{x}_{i_0}$ s.t. $x \longrightarrow_F^* x'$. As above, there exists $k \leq k'$ and an accepting pebble $y = (\mathbf{t}, \mathbf{f}) \in \mathbf{x}_k$ s.t. $i_0 \leq k \leq i_1$ and $x \longrightarrow y \longrightarrow x' \longrightarrow y' \longrightarrow x''$. Since y and y' are accepting, by Lemma 5.7.12(a), $\mathbf{t} \Longrightarrow_F^{\exists} \mathbf{t}'$. By Lemma 5.4.1, $\mathbf{t} \Longrightarrow_F^{\exists} \mathbf{s}'$. By the invariant, $\mathbf{s}_{i_2}$ is the union of all these $\mathbf{s}'$. Therefore, by taking unions, $\mathbf{s}_{i_0} \Longrightarrow_F^{\exists} \mathbf{s}_{i_2}$ by Lemma 5.4.5.

By applying the same argument to further indices i_3 and i_4 s.t. $i_2 \leq i_3 \leq i_4$ and

$$\mathbf{x}_{i_2} \Longrightarrow_F^{\exists} \mathbf{x}_{i_3} \Longrightarrow_F^{\exists} \mathbf{x}_{i_4}$$

we get $\mathbf{s}_{i_2} \Longrightarrow_F^{\exists} \mathbf{s}_{i_4}$. By Lemma 5.7.12(b), there exists j s.t. $i_0 \leq j \leq i_4$ s.t. $(\mathbf{s}_j, \mathbf{g}_j) \in F'$. Since i_0 can be taken arbitrarily large, π_1' is fair. $\square$

While the Miyano-Hayashi construction is an idempotent operation, this is not true for the fair subset construction. Yet, $\mathcal{F}^2(\mathcal{Q})$ is simulation-equivalent to $\mathcal{F}(\mathcal{Q})$.

Lemma 5.7.16. $\mathcal{F}^2(\mathcal{Q}) \approx^{\mathrm{f}}_{(1,1)} \mathcal{F}(\mathcal{Q})$.

Proof. $\mathcal{F}(\mathcal{F}(\mathcal{Q})) \sqsubseteq^{\mathrm{f}}_{(1,n')} \mathcal{F}(\mathcal{Q})$ holds by Lemma 5.7.14, with $n' = |\mathcal{F}(\mathcal{Q})|$. By Lemma 5.7.15, $\mathcal{F}(\mathcal{F}(\mathcal{Q})) \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{F}(\mathcal{Q})$. Conversely, $\mathcal{F}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{F}(\mathcal{F}(\mathcal{Q}))$ by Lemma 5.7.13. □

We prove that the fair subset construction characterizes $(1, n)$-fair simulation.

Proof (of Theorem 5.7.9). We use transitivity, cf. Theorem 5.6.1.

$$\begin{aligned}
\mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(1,n)} \mathcal{S} &\implies \mathcal{F}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(1,n)} \mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(1,n)} \mathcal{S} \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{F}(\mathcal{S}) && \text{by Lemma 5.7.14 and 5.7.13}\\
&\implies \mathcal{F}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(1,n)} \mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(1,n)} \mathcal{S} \sqsubseteq^{\mathrm{f}}_{(1,n)} \mathcal{F}(\mathcal{S}) && \text{by Theorem 5.4.7}\\
&\implies \mathcal{F}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(1,n)} \mathcal{F}(\mathcal{S}) && \text{by Theorem 5.6.1}\\
&\implies \mathcal{F}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{F}(\mathcal{S}) && \text{by Lemma 5.7.15}
\end{aligned}$$

$$\begin{aligned}
&\mathcal{F}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{F}(\mathcal{S}) \implies\\
&\quad\implies \mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{F}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{F}(\mathcal{S}) \sqsubseteq^{\mathrm{f}}_{(1,n)} \mathcal{S} && \text{by Lemma 5.7.13 and 5.7.14}\\
&\quad\implies \mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(1,n)} \mathcal{F}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(1,n)} \mathcal{F}(\mathcal{S}) \sqsubseteq^{\mathrm{f}}_{(1,n)} \mathcal{S} && \text{by Theorem 5.4.7}\\
&\quad\implies \mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(1,n)} \mathcal{S} && \text{by Theorem 5.6.1}
\end{aligned}$$ □

5.7.3 Characterizing (n, n)-fair simulation

As we have seen with Theorem 5.7.1 in Section 5.7.1.1, the Miyano-Hayashi construction characterizes $(n, 1)$-fair simulation. Similarly, by Theorem 5.7.9 in Section 5.7.2.1, the fair subset construction characterizes $(1, n)$-fair simulation. In this section, we use both results to characterize (n, n)-fair simulation.

Theorem 5.7.17. *Let $\mathcal{Q}$ be an ABA, and let $\mathcal{S}$ be a uniform ABA. Moreover, let $n = \max\{|\mathcal{Q}|, |\mathcal{S}|\}$. Then,*

$$\mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(n,n)} \mathcal{S} \iff \mathcal{MH}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{F}(\mathcal{S})$$

Proof. We use transitivity (cf. Theorem 5.6.1).

$$\begin{aligned}
\mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(n,n)} \mathcal{S} &\implies \mathcal{MH}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(n,n)} \mathcal{S} \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{F}(\mathcal{S}) && \text{by Lem. 5.7.7 and 5.7.13}\\
&\implies \mathcal{MH}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(n,n)} \mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(n,n)} \mathcal{S} \sqsubseteq^{\mathrm{f}}_{(n,n)} \mathcal{F}(\mathcal{S}) && \text{by Theorem 5.4.7}\\
&\implies \mathcal{MH}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(n,n)} \mathcal{F}(\mathcal{S}) && \text{by Theorem 5.6.1}\\
&\implies \mathcal{MH}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(1,n)} \mathcal{F}(\mathcal{S}) && \text{by Lemma 5.7.8}\\
&\implies \mathcal{MH}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{F}(\mathcal{S}) && \text{by Lemma 5.7.15}
\end{aligned}$$

$$\begin{aligned}
&\mathcal{MH}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{F}(\mathcal{S}) \implies \\
&\qquad \implies \mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(n,1)} \mathcal{MH}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{F}(\mathcal{S}) \sqsubseteq^{\mathrm{f}}_{(1,n)} \mathcal{S} && \text{by Lemma 5.7.6 and 5.7.14} \\
&\qquad \implies \mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(n,n)} \mathcal{MH}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(n,n)} \mathcal{F}(\mathcal{S}) \sqsubseteq^{\mathrm{f}}_{(n,n)} \mathcal{S} && \text{by Theorem 5.4.7} \\
&\qquad \implies \mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(n,n)} \mathcal{S} && \text{by Theorem 5.6.1}
\end{aligned}$$

□

Moreover, we also show that the two constructions commute with each other, up to fair-simulation equivalence. The proof of the following theorem is at the end of the section.

Theorem 5.7.18. *For a uniform $\mathcal{Q}$, $\mathcal{MH}(\mathcal{F}(\mathcal{Q})) \approx^{\mathrm{f}}_{(1,1)} \mathcal{F}(\mathcal{MH}(\mathcal{Q}))$.*

Before proving Theorem 5.7.18, we need some preliminary result.

In Theorem 5.7.17, the MH-construction is applied on the left and the fair construction on the right. In the next lemma, we study what happens in the symmetric case.

Lemma 5.7.19. *Let $\mathcal{Q}$ be an uniform ABA, and let $\mathcal{S}$ be an ABA. Let $n = \max\{|\mathcal{Q}|, |\mathcal{S}|\}$. Then,*

$$\mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(n,n)} \mathcal{S} \iff \mathcal{F}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(n,n)} \mathcal{MH}(\mathcal{S})$$

Proof.

$$\begin{aligned}
\mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(n,n)} \mathcal{S} &\implies \mathcal{F}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(1,n)} \mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(n,n)} \mathcal{S} \sqsubseteq^{\mathrm{f}}_{(n,1)} \mathcal{MH}(\mathcal{S}) && \text{by Lem. 5.7.14 and 5.7.6} \\
&\implies \mathcal{F}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(n,n)} \mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(n,n)} \mathcal{S} \sqsubseteq^{\mathrm{f}}_{(n,n)} \mathcal{MH}(\mathcal{S}) && \text{by Theorem 5.4.7} \\
&\implies \mathcal{F}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(n,n)} \mathcal{MH}(\mathcal{S}) && \text{by Theorem 5.6.1}
\end{aligned}$$

$$\begin{aligned}
&\mathcal{F}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(n,n)} \mathcal{MH}(\mathcal{S}) \\
&\qquad \implies \mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{F}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(n,n)} \mathcal{MH}(\mathcal{S}) \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{S} && \text{by Lemma 5.7.13 and 5.7.7} \\
&\qquad \implies \mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(n,n)} \mathcal{F}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(n,n)} \mathcal{MH}(\mathcal{S}) \sqsubseteq^{\mathrm{f}}_{(n,n)} \mathcal{S} && \text{by Theorem 5.4.7} \\
&\qquad \implies \mathcal{Q} \sqsubseteq^{\mathrm{f}}_{(n,n)} \mathcal{S} && \text{by Theorem 5.6.1}
\end{aligned}$$

□

The following is an immediate corollary of Theorem 5.7.17 and Lemma 5.7.19 when $\mathcal{Q} = \mathcal{S}$.

Corollary 5.7.20. *For any uniform ABA $\mathcal{Q}$, let $n = |\mathcal{Q}|$. Then,*

$$\mathcal{MH}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{F}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(n,n)} \mathcal{MH}(\mathcal{Q})$$

We can finally prove that the Miyano-Hayashi construction and its dual commute with each other up to $(1,1)$-fair simulation equivalence.

Proof (of Theorem 5.7.18). Let $n = \max\{|\mathcal{Q}|, |\mathcal{F}(\mathcal{Q})|, |\mathcal{MH}(\mathcal{Q})|\}$.

$$\begin{aligned}
&\mathcal{F}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(n,n)} \mathcal{MH}(\mathcal{Q}) && \text{by Corollary 5.7.20}\\
&\implies \mathcal{MH}(\mathcal{F}(\mathcal{Q})) \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{F}(\mathcal{MH}(\mathcal{Q})) && \text{by Theorem 5.7.17}
\end{aligned}$$

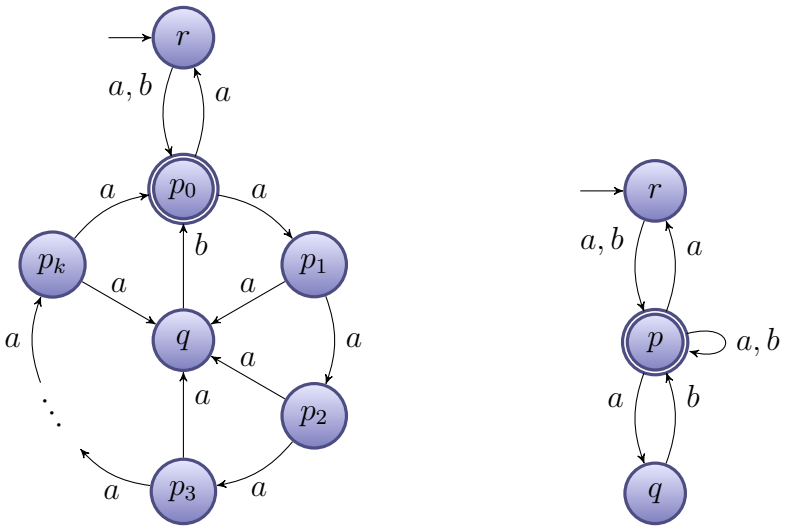
$$\begin{aligned}
&\mathcal{F}(\mathcal{MH}(\mathcal{Q})) \sqsubseteq^{\mathrm{f}}_{(1,n)} \mathcal{MH}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{F}(\mathcal{Q}) && \text{by Lem. 5.7.14 and Cor. 5.7.20}\\
&\implies \mathcal{F}(\mathcal{MH}(\mathcal{Q})) \sqsubseteq^{\mathrm{f}}_{(1,n)} \mathcal{F}(\mathcal{Q}) && \text{by Theorem 5.4.7 and 5.6.1}\\
&\implies \mathcal{F}(\mathcal{MH}(\mathcal{Q})) \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{F}(\mathcal{Q}) && \text{by Lemma 5.7.15}\\
&\implies \mathcal{F}(\mathcal{MH}(\mathcal{Q})) \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{F}(\mathcal{Q}) \sqsubseteq^{\mathrm{f}}_{(n,1)} \mathcal{MH}(\mathcal{F}(\mathcal{Q})) && \text{by Lemma 5.7.6}\\
&\implies \mathcal{F}(\mathcal{MH}(\mathcal{Q})) \sqsubseteq^{\mathrm{f}}_{(n,1)} \mathcal{MH}(\mathcal{F}(\mathcal{Q})) && \text{by Theorem 5.4.7 and 5.6.1}\\
&\implies \mathcal{F}(\mathcal{MH}(\mathcal{Q})) \sqsubseteq^{\mathrm{f}}_{(1,1)} \mathcal{MH}(\mathcal{F}(\mathcal{Q}))
\end{aligned}$$

where the last step follows from the simple fact that $\mathcal{F}(\mathcal{MH}(\mathcal{Q}))$ has only existential states and Duplicator cannot split pebbles on the left in this case. □

5.8 Quotienting

In this section, we show how and when multipebble simulation preorders can be used for quotienting alternating Büchi automata, i.e., when they are *good for quotienting* (GFQ) (cf. Section 2.4.4). The motivation for considering quotienting w.r.t. multipebble simulations is that the reduction in size can be substantial; this holds already for nondeterministic automata, as shown in the following example.

Example 5.8.1 - Multipebble delayed simulation quotients can achieve arbitrary large compression ratios



a) Family of automata $\mathcal{Q}_k$ b) The quotients $\mathcal{Q}_k/\approx^{2\text{-de}}$

Consider the family of automata $\mathcal{Q}_k$ above, over the alphabet $\Sigma = \{a, b\}$. Each $\mathcal{Q}_k$ has $k + 3$ states. We analyze quotienting w.r.t. usual forward and backward simulation, and then w.r.t. multipebble simulation.

Forward 1-pebble simulation No two p_i, p_j (with $i \neq j$) are forward simulation equivalent. First, note that p_1 does not simulate p_0: Indeed, p_0 can go to r, from which both actions a and b are enabled, while from p_1 one can either go to q or to p_2, but neither of them can do both a and b. The same reasoning shows that p_0 is not simulated by any p_j, with $j \neq 0$. Finally, p_i is not simulated by p_j, with $i \neq j$, since from configuration $\langle p_i, p_j \rangle$ Spoiler can force configuration $\langle p_0, p_k \rangle$ (for some $k \neq 0$), and then win as above. Therefore, for $i \neq j$, p_i and p_j are forward simulation incomparable.

Backward simulation No two p_i, p_j (with $i \neq j$) are backward simulation equivalent: Indeed, r is the unique initial state, and p_i can backward go to r via a unique path in $i + 1$ steps; therefore for $i \neq j$, p_i and p_j are backward simulation incomparable.

Multipebble simulation However, all p_j's are multipebble delayed simulation equivalent (and $k = 2$ pebbles suffice):

$$p_0 \approx^{\text{2-de}} p_1 \approx^{\text{2-de}} \cdots \approx^{\text{2-de}} p_k$$

Indeed, p_1 2-simulates p_0 as follows:

- If Spoiler plays $p_0 \xrightarrow{a} p_1$, then Duplicator just plays $p_1 \xrightarrow{a} p_2$, getting one step close to the accepting state p_0.
- If Spoiler plays $p_0 \xrightarrow{a} r$, then Duplicator splits pebbles to $\{q, p_2\}$. Then, if Spoiler plays $r \xrightarrow{a} p_0$, then Duplicator drops the pebble on q and goes from p_2 to p_3 (she gets two steps closer to the accepting state p_0). Otherwise, if Spoiler plays $r \xrightarrow{b} p_0$, then Duplicator drops the pebble on p_2 and goes from q to p_0, thus visiting an accepting state.

A similar argument shows that any p_j simulates p_0. It is not difficult to see that this implies that any p_j in fact simulates any p_i. Therefore, all p_j's are 2-delayed simulation equivalent. After quotienting, we obtain the 3-states quotient automaton $\mathcal{Q}_k/\approx^{\text{2-de}}$ on the right. Thus, the compression ratio achieved by multipebble simulation w.r.t. 1-pebble forward and backward simulations can be arbitrarily large.

The results of this section are summarized in Table 5.2 (the $(1, 1)$ case is from [55]). We have seen in Section 2.5 that fair simulation is not GFQ, and this holds already for nondeterministic Büchi automata. Therefore, we only consider direct and delayed simulation. In Section 5.8.2, we present *minimax quotients* [55] for multipebble direct simulation, and we prove that $\sqsubseteq^{\text{di}}_{(k_0,k_1)}$, for any $k_0, k_1 > 0$, is GFQ when considering (an extension of) minimax quotients.

In Section 5.8.3, we explain why minimax quotients do not work for delayed simulation. This leads to Section 5.8.4, where we present *semielective quotients* [55] for delayed simulation. We we prove that multipebble $(1, n)$-delayed simulation is GFQ for semielective quotients, where n is the size of the automaton. Perhaps surprisingly, we show that $\sqsubseteq^{\text{de}}_{(1,k)}$ for $1 < k < n$, and $\sqsubseteq^{\text{de}}_{(k,1)}$ for $k > 1$, are not GFQ in general.

5.8.1 Quotienting difficulties

When quotienting in alternating models where each state can be either existential or universal, problems arise if one has both existential and universal states in the same

Simulation \Pebbles	$(1,1)$	(k_0,k_1)	$(1,n)$	$(n,1)/(n,n)$
Direct simulation (minimax quotients) Section 5.8.2	✓	✓	✓	✓
Delayed simulation (semielective quotients) Section 5.8.4	✓	×	✓	×

Table 5.2: Quotienting summary (✓= GFQ, ×= not GFQ)

equivalence class. In this case, one has what is called a *mixed equivalence class*. The issue is that it is not clear whether a mixed class should be declared existential or universal in the quotient automaton. Basically, the proper treatment of mixed classes is the main issue when quotienting alternating automata.

Of course, one could sidestep the problem by splitting each mixed class into two separate classes, one existential and the other universal. However, this might double the number of states in the quotient, and it also makes the problem uninteresting. Thus, we insist on having mixed classes in the quotient automaton.

In [55], the problem of mixed classes is solved by unilaterally declaring them *existential*. Consequently, since universal states in mixed classes are now interpreted "existentially", the transition relation needs to be designed carefully as to avoid introducing spurious computations in the automaton. Two kind of quotients are defined, namely *minimax* and *semielective*, to be used, respectively, with direct and delayed simulation.

Remark 5.8.1. Another difference between quotienting alternating and non-alternating models is that, in alternating models, the monotonicity property of Lemma 2.4.7 fails; that is, coarser relations does not necessarily give rise to quotient automata recognizing coarser languages. In fact, when two existential states are merged together, the language grows, while when two universal states are merged together, the language shrinks. Since usually both existential and universal states are involved, the overall result is unpredictable in general.

5.8.2 Minimax quotients

Minimax quotients [55] are a generalization of max quotients for alternating models. Recall that, in max quotients, one considers only transitions induced by maximal successors. In minimax quotients, the same thing happens for existential states, and, for

universal states, one consider only transitions to *minimal successors*. In the following, fix an ABA $\mathcal{Q} = (Q, \Sigma, q_I, \Delta, E, U, F)$, together with a preorder $\preceq \subseteq Q \times Q$.

Definition 5.8.2 (Minimal successors)**.** *For a symbol $a \in \Sigma$ and states q and q' in $\mathcal{Q}$, q' is a $\preceq$-minimal a-successor of q iff $q \xrightarrow{a} q'$ and, for any other $q'' \in Q$ s.t. $q \xrightarrow{a} q''$, if $q'' \preceq q'$, then $q' \preceq q''$. Let $\min_a^{\preceq}(q)$ be the set of $\preceq$-minimal a-successors of q.*

The *minimax* quotient automaton $\mathcal{Q}^{\mathrm{m}}_{\approx} = (Q_{\approx}, \Sigma, q^I_{\approx}, \Delta^{\mathrm{m}}_{\approx}, E^{\mathrm{m}}_{\approx}, U^{\mathrm{m}}_{\approx}, F_{\approx})$ is defined as follows. States in the quotient are equivalence classes of states, $Q_{\approx} = [Q]$, the initial state is $q^I_{\approx} = [q^I]$ and final states are those in $F_{\approx} = [F]$. Existential and mixed classes are declared existential in the quotient, $E^{\mathrm{m}}_{\approx} = [E]$, and universal classes are declared universal in the quotient: $U^{\mathrm{m}}_{\approx} = [Q] \setminus E^{\mathrm{m}}_{\approx}$. The transition relation is defined as follows: $([q], a, [q']) \in \Delta^{\mathrm{m}}_{\approx}$ iff either

- $q \in E$ and $\exists(\widehat{q} \in [q], \widehat{q}' \in [q'])$ s.t. $(\widehat{q}, a, \widehat{q}') \in \Delta$ and $\widehat{q}' \in \max_a^{\preceq}(\widehat{q})$, or
- $q \in U$ and $\exists(\widehat{q} \in [q], \widehat{q}' \in [q'])$ s.t. $(\widehat{q}, a, \widehat{q}') \in \Delta$ and $\widehat{q}' \in \min_a^{\preceq}(\widehat{q})$.

Remark 5.8.3. The restriction to maximal/minimal successors in pure classes is not needed for correctness, although it is beneficial in reducing the size of the transition relation. In mixed classes, since they are declared existential, considering only minimal successors of universal states is required for correctness. (Dually, if they were declared universal, then the restriction to maximal successors of existential states would be required.)

Theorem 5.8.4 (Cf. Theorem 2 of [55])**.** *Direct simulation is GFQ for minimax quotients.*

The following is a simple, but crucial lemma for the further study of the properties of quotients. It follows from the definition of simulation.

Lemma 5.8.5 (cf. Corollary 3 of [55])**.** *Let q and s be two equivalent states, $q \approx^x s$. Then,*

1. *If $q, s \in E$, then $\forall(q' \in \max_a^{\sqsubseteq^x}(q)) \cdot \exists(s' \in \max_a^{\sqsubseteq^x}(s)) \cdot q' \approx^x s'$.*
2. *If $q, s \in U$, then $\forall(q' \in \min_a^{\sqsubseteq^x}(q)) \cdot \exists(s' \in \min_a^{\sqsubseteq^x}(s)) \cdot q' \approx^x s'$.*
3. *If $q \in E$ and $s \in U$, then $\forall(q' \in \max_a^{\sqsubseteq^x}(q), s' \in \min_a^{\sqsubseteq^x}(s)) \cdot q' \approx^x s'$.*

Remark 5.8.6. As a consequence of case 3. above, equivalent states in minimax mixed classes have only transitions to equivalent states. In other words, mixed classes are deterministic states in the minimax quotient, thus it does not really matter whether they are declared existential or universal.[5]

5.8.2.1 Improved minimax quotients

Lemma 5.8.5 is saying even more: If we restrict ourselves to maximal/minimal successors, then, for two equivalent states p and r (existential or universal), every transition from p is subsumed by some transition from r *going to an equivalent state*. Thus, for each quotient class $[q] \in [Q]$, we can fix a *representative* $\lceil q \rceil \in [q]$ among the class, s.t. any transition from any other $\widehat{q} \in [q]$ can be simulated by $\lceil q \rceil$ alone.

Therefore, it suffices to consider just the transitions induced by representatives. In general, this reduces the number of transitions. For 1-pebble simulations, by Lemma 5.8.5, we obtain exactly the same transitions. However, in the context of multipebble simulations, fixing representatives can reduce the size of the transition relation.

Formally, fix a representative-selection function $\lceil \cdot \rceil : Q \mapsto Q$ s.t., for every $q \in Q$, 1) it picks an element from the class, i.e., $\lceil q \rceil \in [q]$, and 2) it respects the class, i.e., for any $q' \in Q$ with $q' \in [q]$, $\lceil q \rceil = \lceil q' \rceil$. The *minimax+* quotient automaton $\mathcal{Q}^{\text{m+}}_{\approx} = (Q_{\approx}, \Sigma, q^I_{\approx}, \Delta^{\text{m+}}_{\approx}, E^{\text{m+}}_{\approx}, U^{\text{m+}}_{\approx}, F_{\approx})$ is defined as the minimax one, with the following differences. Quotient states inherit the type of representatives: That is, if $\lceil q \rceil \in E$, then $[q] \in E^{\text{m+}}_{\approx}$, and if $\lceil q \rceil \in U$, then $[q] \in U^{\text{m+}}_{\approx}$. Transitions are defined as follows: $([q], a, [q']) \in \Delta^{\text{m+}}_{\approx}$ iff either

- $\lceil q \rceil \in E$ and $\exists(\widehat{q}' \in [q'])$ s.t. $(\lceil q \rceil, a, \widehat{q}') \in \Delta$ and $\widehat{q}' \in \max^{\preceq}_{a}(\lceil q \rceil)$, or
- $\lceil q \rceil \in U$ and $\exists(\widehat{q}' \in [q'])$ s.t. $(\lceil q \rceil, a, \widehat{q}') \in \Delta$ and $\widehat{q}' \in \min^{\preceq}_{a}(\lceil q \rceil)$.

Remark 5.8.7. Fixing representatives it is not required for correctness, but in the multipebble case it can reduce the size of the transition relation.

We prove that direct multipebble simulation is GFQ for minimax+ quotients.

Theorem 5.8.8. *For $k_0, k_1 > 0$, (k_0, k_1)-direct simulation is GFQ for minimax+ quotients.*

[5] In fact, states $q \in E$ and $s \in U$ with $q \approx^x s$ are already deterministic in the original automaton, after non-maximal transitions from q and non-minimal transitions from s have been removed. Thus, it is possible to make such q's universal (or the s's existential), and mixed classes won't even arise.

We prove the theorem above in Section 5.8.2.3. Below, we compare minimax and minimax+ quotients (in the $(1,1)$-case they coincide), and in Section 5.8.2.2 we propose some further improvement to the latter.

Transition structure Minimax+ quotients are actually a family of quotients, where each concrete quotient depends on the choice of a representative for each equivalence class. In general, different choices of representatives induce different transition structures since, for two equivalent existential states q_0 and q_1, any maximal transition from q_0 might be subsumed only by a *set* of maximal transitions from q_1.

Therefore, the fact that direct simulation is GFQ w.r.t. minimax quotients for any choice of representatives is a non-trivial correctness property. Moreover, it allows one to choose representatives as to reduce the number of outgoing transitions from every quotient state.

However, from Lemma 5.8.5, it follows that, in the case of $(1,1)$-quotients, *any* choice of representatives will induce exactly the same transitions. Indeed, for $(1,1)$-simulations, if two existential states are equivalent, then any maximal transition of one is simulated by *a single* maximal transition of the other; by maximality, these transition in fact go to equivalent states, i.e., to the same equivalence class. Therefore, for $(1,1)$-simulation the transition structure does not depend on representatives, and minimax and minimax+ quotients coincide.

Mixed classes The choice of representatives for mixed classes determines whether the quotient state is declared to be existential or universal. Since $(1,1)$-minimax mixed classes are deterministic (cf. Remark 5.8.6), it does not make any difference whether the class is declared to be existential or universal in $(1,1)$-quotients.

However, this does not need to be the case for general (k_0,k_1)-quotients. Indeed, (k_0,k_1)-minimax mixed classes are non-deterministic in general, and the choice of representative does really change the semantics of the quotient automaton. Anyway, Theorem 5.8.8 shows that quotienting is correct for any such choice. In particular, one can in fact declare minimax mixed classes to be either existential or universal, by choosing a suitable existential or a universal representative, respectively.

5.8.2.2 Further improvements

We consider two ways of further reducing the number of transitions in the quotient automaton:

1. We allow a transition to be subsumed by a *set* of transitions in the definition of maximality/minimality of a-successors.

2. We allow representatives to depend on the input symbol $a \in \Sigma$, i.e., there might be different representatives for different input symbols $a_0, a_1, \ldots$.

Maximal sets of successors In general (k_1, k_2)-quotients it is not necessary to consider *every* maximal/minimal successor of a given representative, but it is safe to discard those successors which are (k_1, k_2)-simulated by a *set* of other maximal/minimal successors. For instance, for a given existential representative $\lceil q \rceil \in E$ and symbol $a \in \Sigma$, we say that $\mathbf{q}'$ is a set of *maximal a-successors of* $\lceil q \rceil$ iff

$$\mathbf{q}' \subseteq \max_a^{\preceq}(\lceil q \rceil) \wedge \left(\forall q'' \in \max_a^{\preceq}(\lceil q \rceil) \setminus \mathbf{q}' \cdot q'' \preceq \mathbf{q}'\right)$$

where $\preceq = \sqsubseteq^{\mathrm{di}}_{(k_1,k_2)}$. A similar definition can be given for sets of minimal successors of universal states. Notice that the above definition is non-deterministic, in the sense that there might be different sets of maximal/minimal successors: In this case, one can just take any such $\subseteq$-maximal/minimal set of successors.

Formally, one has a transition $([q], a, [q']) \in \Delta^{\mathrm{m+}}_{\approx}$ iff either

- $\lceil q \rceil \in E$ and $\exists\, \widehat{q}' \in [q'] \cap \mathbf{q}'$ s.t. $(\lceil q \rceil, a, \widehat{q}') \in \Delta$ and $\mathbf{q}'$ is a set of maximal a-successors of $\lceil q \rceil$, or

- $\lceil q \rceil \in U$ and $\exists\, \widehat{q}' \in [q'] \cap \mathbf{q}'$ s.t. $(\lceil q \rceil, a, \widehat{q}') \in \Delta$ and $\mathbf{q}'$ is a set of minimal a-successors of $\lceil q \rceil$.

Non-uniform representatives An other way of reducing the number of transitions in the quotient automaton is to relax the way representatives are chosen. Instead of fixing a representative once and for all for every equivalence class, one can let the representative depend on the input symbol. That is, for every $a \in \Sigma$, we select a (possibly different) representative $\lceil q \rceil_a$, and a-successors in the quotient are those maximal/minimal successors induced by $\lceil q \rceil_a$. Since quotient states have to be declared either existential or universal, non-uniform representatives need to be either all existential or all universal for a given class. That is, either $\bigcup_{a \in \Sigma}\{\lceil q \rceil_a\} \subseteq E$ or $\bigcup_{a \in \Sigma}\{\lceil q \rceil_a\} \subseteq U$ holds.

In this way, for each input symbol a, one can select the representative $\lceil q \rceil_a$ with the least number of maximal/minimal successors. The only tradeoff is whether mixed classes should have existential or universal representatives; in this case, one can compare

the best existential representative against the best universal one, and then selects the best of these two.

This induces an improved notion of minimax quotient where transitions are defined as follows: $([q], a, [q']) \in \Delta^{\text{m+}}_{\approx}$ iff either

- $\lceil q \rceil_a \in E$ and $\exists\, \widehat{q}' \in [q']$ s.t. $(\lceil q \rceil_a, a, \widehat{q}') \in \Delta$ and $\widehat{q}' \in \max^{\preceq}_a(\lceil q \rceil_a)$, or
- $\lceil q \rceil_a \in U$ and $\exists\, \widehat{q}' \in [q']$ s.t. $(\lceil q \rceil_a, a, \widehat{q}') \in \Delta$ and $\widehat{q}' \in \min^{\preceq}_a(\lceil q \rceil_a)$.

Since the choice of representative does not affect the transitions for $(1, 1)$-quotients, this notion coincides with the usual one for $(1, 1)$-simulations. Non-uniform representatives can be used in addition to the previous optimization regarding maximal sets of successors.

5.8.2.3 Correctness of minimax+ quotients

The rest of the section is devoted to the proof of Theorem 5.8.8. In the following, we write just $[q]$ for $[q]_{\text{m+}}$, where we quotient w.r.t. the equivalence induced by the transitive closure of $\sqsubseteq^{\text{di}}_{(k_1,k_2)}$, for any fixed $k_1, k_2 > 0$. For simplicity, we just say that "the quotient is taken w.r.t. $\sqsubseteq^{\text{di}}_{(k_1,k_2)}$". We actually prove the following stronger statement, where we show that quotienting does not only preserves the language, but even maximal multipebble direct simulation itself.

Theorem 5.8.9. *Let $\mathcal{Q}$ be an ABA with n states, and let $\mathcal{Q}'$ be its quotient w.r.t. (the equivalence induced by the transitive closure of) $\sqsubseteq^{\text{di}}_{(k_1,k_2)}$. Then, for any state q in $\mathcal{Q}$ and its corresponding quotient class $[q]$ in $\mathcal{Q}'$, q is (n, n)-direct simulation equivalent to $[q]$, i.e., $q \approx^{\text{di}}_{(n,n)} [q]$.*

Theorem 5.8.9 immediately follows from Lemmas 5.8.10 and 5.8.12 below, since $q \approx^{\text{di}}_{(n,n)} q$.

Lemma 5.8.10. *If* $\mathrm{q} \sqsubseteq^{\text{di}}_{(n,n)} \mathrm{s}$, *then* $[\mathrm{q}] \sqsubseteq^{\text{di}}_{(n,n)} \mathrm{s}$, *where the quotient is taken w.r.t.* $\sqsubseteq^{\text{di}}_{(k_1,k_2)}$.

Remark 5.8.11. Lemma 5.8.10 holds even if we do not require maximality of transitions induced from existential states in the quotient. Indeed, the lemma holds for any possible way of selecting transitions from existential states—even if none is selected: The reason is that any other way of selecting transition other than the maximal ones (i.e., adding non-maximal transitions or removing strictly maximal ones) would only reduce the set of behaviors of the quotient automaton, and this would make Spoiler weaker, thus preserving the direction of the lemma.

Proof (of Lemma 5.8.10). Let $\mathbb{G} = \mathbb{G}^{\mathrm{di}}_{(n,n)}([\mathbf{q}], \mathbf{s})$ be the outer simulation game, and let $\mathbb{G}' = \mathbb{G}^{\mathrm{di}}_{(n,n)}(\mathbf{q}, \mathbf{s})$ be the inner simulation game. While playing the outer game $\mathbb{G}$, we update the inner game $\mathbb{G}'$ in order to maintain the following invariant: If at round i the outer game is in configuration $\langle [\mathbf{q}_i], \mathbf{s}_i \rangle$, then $\mathbf{q}_i \sqsubseteq^{\mathrm{di}}_{(n,n)} \mathbf{s}_i$. It is easily seen that the invariant implies the lemma: Indeed, if $[\mathbf{q}_i] \cap F' \neq \emptyset$, then, by the definition of quotient, there exists $\hat{q} \in \mathbf{q}_i$ s.t. $[\hat{q}] \in F'$. By the definition of di-simulation, all states in the quotient are accepting, i.e., $[\hat{q}] \subseteq F$, and, in particular, $\hat{q} \in F$. Thus, $\mathbf{q}_i \cap F \neq \emptyset$. By the invariant and by the definition of di-simulation, $\mathbf{s}_i \subseteq F$.

Assume the current configuration in $\mathbb{G}$ is $\langle [\mathbf{q}], \mathbf{s} \rangle$, and $\mathbf{q} \sqsubseteq^{\mathrm{di}}_{(n,n)} \mathbf{s}$. We show that, no matter what Spoiler does, Duplicator can ensure to go in a configuration of the form $\langle [\mathbf{q}'], \mathbf{s}' \rangle$ s.t. $\mathbf{q}' \sqsubseteq^{\mathrm{di}}_{(n,n)} \mathbf{s}'$. W.l.o.g. we assume that $\mathbf{q}$ is a set of representatives; that is, $\mathbf{q} = \{q_0, \ldots, q_k\}$ with $q_i = \lceil q_i \rceil$. If not, just replace any element $\hat{q} \in \mathbf{q}$ with an equivalent representative $\lceil \hat{q} \rceil$. This gives a new set $\lceil \mathbf{q} \rceil$. This operation is correct since direct simulation $\sqsubseteq^{\mathrm{di}}_{(k_1,k_2)}$ preserves final states (so final states can only be replaced by final states), and $\sqsubseteq^{\mathrm{di}}_{(k_1,k_2)} \subseteq \sqsubseteq^{\mathrm{di}}_{(n,n)}$, where $\sqsubseteq^{\mathrm{di}}_{(n,n)}$ is transitive. Therefore, $\lceil \mathbf{q} \rceil \approx \mathbf{q} \sqsubseteq^{\mathrm{di}}_{(n,n)} \mathbf{s}$, which implies $\lceil \mathbf{q} \rceil \sqsubseteq^{\mathrm{di}}_{(n,n)} \mathbf{s}$.

We partition the two sets $[\mathbf{q}]$ and $\mathbf{s}$ into existential and universal states, as follows:

$$[\mathbf{q}] = [\mathbf{q}^E] \cup [\mathbf{q}^U]$$
$$\mathbf{s} = \mathbf{s}^E \cup \mathbf{s}^U$$

Here, with $\mathbf{q}^E$ we mean a set of states in E, and similarly with the other sets above. Therefore, $[\mathbf{q}^E]$ is the set of classes with existential representatives in $\mathbf{q}^E$, and $[\mathbf{q}^U]$ is the set of classes with universal representatives in $\mathbf{q}^U$. By the assumption above, and by the definition of minimax quotients, classes in $[\mathbf{q}^E]$ are existential, and classes in $[\mathbf{q}^U]$ are universal. Let Spoiler choose input symbol a and transition

$$([\mathbf{q}], \mathbf{s}, a, [\bar{\mathbf{q}}], \bar{\mathbf{s}}) \in \Gamma^{\mathrm{Sp}}_{\mathbb{G}}$$

that is, existential-left pebbles in $[\mathbf{q}^E]$ are moved to $[\bar{\mathbf{q}}]$, and universal-right pebbles in $\mathbf{s}^U$ are moved to $\bar{\mathbf{s}}$. By the definition of minimax quotient, transitions from existential classes in $[\mathbf{q}^E]$ are those induced by their existential representatives in $\mathbf{q}^E$. Also, we take the elements in $\bar{\mathbf{q}}$ to be those who receive the transitions coming from $\mathbf{q}^E$. As noted above, we do not actually require maximality of transitions from existential states. Spoiler's move in $\mathbb{G}$ directly induces the move by Spoiler in $\mathbb{G}'$ below

$$(\mathbf{q}, \mathbf{s}, a, \bar{\mathbf{q}}, \bar{\mathbf{s}}) \in \Gamma^{\mathrm{Sp}}_{\mathbb{G}'}$$

Then, we apply Duplicator's winning strategy in $\mathbb{G}'$, to obtain

$$(\mathbf{q}, \mathbf{s}, a, \bar{\mathbf{q}}, \bar{\mathbf{s}}, \mathbf{q}', \mathbf{s}') \in \Gamma^{\mathrm{Dup}}_{\mathbb{G}'}$$

where, by the definition of Duplicator's move, $\mathbf{q}'$ is a subset of $\bar{\mathbf{q}} \cup \Delta(\mathbf{q}^U, a)$ and $\mathbf{s}'$ is a subset of $\bar{\mathbf{s}} \cup \Delta(\mathbf{s}^E, a)$. To ensure that transitions from $\mathbf{q}^U$ to $\mathbf{q}'$ do induce corresponding transitions between the classes $[\mathbf{q}^U]$ and $[\mathbf{q}']$, we assume with no loss of generality that Duplicator only selects $\sqsubseteq^{\mathrm{di}}_{(k_1,k_2)}$-*minimal* a-successors of states in $\mathbf{q}^U$. Indeed, for each non-minimal a-successor there exists a smaller, minimal one which is at least as good for Duplicator.

This implies that Duplicator's move in $\mathbb{G}'$ directly induces a corresponding move in $\mathbb{G}$. Therefore, we can define Duplicator's response in $\mathbb{G}$ as

$$([\mathbf{q}], \mathbf{s}, a, [\bar{\mathbf{q}}], \bar{\mathbf{s}}, [\mathbf{q}'], \mathbf{s}') \in \Gamma^{\mathrm{Dup}}_{\mathbb{G}}$$

$\mathbf{q}' \sqsubseteq^{\mathrm{di}}_{(n,n)} \mathbf{s}'$ since Duplicator is playing according to a winning strategy in $\mathbb{G}'$, and the invariant is preserved. □

Lemma 5.8.12. *If* $\mathbf{q} \sqsubseteq^{\mathrm{di}}_{(n,n)} \mathbf{s}$*, then* $\mathbf{q} \sqsubseteq^{\mathrm{di}}_{(n,n)} [\mathbf{s}]$*, where the quotient is taken w.r.t.* $\sqsubseteq^{\mathrm{di}}_{(k_1,k_2)}$.

Remark 5.8.13. The lemma holds even if we do not require minimality of transitions from universal states, since selecting transitions in the quotient other that minimal ones would only make Spoiler weaker. This is dual to Lemma 5.8.10.

Proof (of Lemma 5.8.12). The structure of the proof is similar to that of Lemma 5.8.10. Let $\mathbb{G} = \mathbb{G}^{\mathrm{di}}_{(n,n)}(\mathbf{q}, [\mathbf{s}])$ be the outer and let $\mathbb{G}' = \mathbb{G}^{\mathrm{di}}_{(n,n)}(\mathbf{q}, \mathbf{s})$ be the inner simulation game. We maintain the following invariant: If at round i the outer game is in configuration $\langle \mathbf{q}_i, [\mathbf{s}_i] \rangle$, then $\mathbf{q}_i \sqsubseteq^{\mathrm{di}}_{(n,n)} \mathbf{s}_i$. Clearly, the invariant implies the lemma: If $\mathbf{q}_i \cap F \neq \emptyset$, then, by the invariant, $\mathbf{s}_i \subseteq F$, and, by the definition of quotient, $[\mathbf{s}_i] \subseteq F'$.

Assume that the current configuration in $\mathbb{G}$ is $\langle \mathbf{q}, [\mathbf{s}] \rangle$, with $\mathbf{q} \sqsubseteq^{\mathrm{di}}_{(n,n)} \mathbf{s}$. We show that, no matter what Spoiler does, Duplicator can force a configuration of the form $\langle \mathbf{q}', [\mathbf{s}'] \rangle$ s.t. $\mathbf{q}' \sqsubseteq^{\mathrm{di}}_{(n,n)} \mathbf{s}'$. As in Lemma 5.8.10, we can safely assume that $\mathbf{s}$ is a set of representatives, that is, $\mathbf{s} = \lceil \mathbf{s} \rceil$.

We partition the two sets $\mathbf{q}$ and $[\mathbf{s}]$ into existential and universal states, as follows:

$$\mathbf{q} = \mathbf{q}^E \cup \mathbf{q}^U$$
$$[\mathbf{s}] = [\mathbf{s}^E] \cup [\mathbf{s}^U]$$

Here, with $\mathbf{q}^E$ we mean a set of states in E, and similarly with the other sets above. Therefore, $[\mathbf{s}^E]$ is the set of classes with existential representatives in $\mathbf{s}^E$, and $[\mathbf{s}^U]$ is the set of classes with universal representatives in $\mathbf{s}^U$. By the definition of quotient, classes in $[\mathbf{s}^E]$ are existential and classes in $[\mathbf{s}^U]$ are universal. Let Spoiler choose input symbol a and transition

$$(\mathbf{q}, [\mathbf{s}], a, \bar{\mathbf{q}}, [\bar{\mathbf{s}}]) \in \Gamma^{\mathrm{Sp}}_{\mathbb{G}}$$

where existential-left pebbles in $\mathbf{q}^E$ are moved to $\bar{\mathbf{q}}$, and universal-right pebbles in $[\mathbf{s}^U]$ are moved to $[\bar{\mathbf{s}}]$. By the definition of minimax quotient, transitions from universal classes in $[\mathbf{s}^U]$ are those induced by their universal members. As in Lemma 5.8.10, we can assume that these transitions go from representatives in $\mathbf{s}^U$ to elements in $\bar{\mathbf{s}}$. Notice that we *do not* require minimality of transitions from universal states here: as in Lemma 5.8.10, allowing less transitions would only make Spoiler weaker. Spoiler's move in $\mathbb{G}$ directly induces the move by Spoiler in $\mathbb{G}'$ below

$$(\mathbf{q}, \mathbf{s}, a, \bar{\mathbf{q}}, \bar{\mathbf{s}}) \in \Gamma^{\mathrm{Sp}}_{\mathbb{G}'}$$

Then, we apply Duplicator's winning strategy in $\mathbb{G}'$, to obtain

$$(\mathbf{q}, \mathbf{s}, a, \bar{\mathbf{q}}, \bar{\mathbf{s}}, \mathbf{q}', \mathbf{s}') \in \Gamma^{\mathrm{Dup}}_{\mathbb{G}'}$$

where, by the definition of Duplicator's move, $\mathbf{q}'$ is a subset of $\bar{\mathbf{q}} \cup \Delta(\mathbf{q}^U, a)$ and $\mathbf{s}'$ is a subset of $\bar{\mathbf{s}} \cup \Delta(\mathbf{s}^E, a)$. To ensure that transitions from $\mathbf{s}^E$ to $\mathbf{s}'$ do induce corresponding transitions between the classes $[\mathbf{s}^E]$ and $[\mathbf{s}']$, we assume with no loss of generality that Duplicator only selects $\sqsubseteq^{\mathrm{di}}_{(k_1,k_2)}$-*maximal* a-successors of states in $\mathbf{s}^E$. Indeed, for each non-maximal successor there exists a larger, maximal one which is at least as good for Duplicator. This implies that Duplicator's move in $\mathbb{G}'$ directly induces a corresponding move in $\mathbb{G}$. Therefore, we define Duplicator's response in $\mathbb{G}$ as

$$(\mathbf{q}, [\mathbf{s}], a, \bar{\mathbf{q}}, [\bar{\mathbf{s}}], \mathbf{q}', [\mathbf{s}']) \in \Gamma^{\mathrm{Dup}}_{\mathbb{G}}$$

Since Duplicator is playing according to a winning strategy in $\mathbb{G}'$, $\mathbf{q}' \sqsubseteq^{\mathrm{di}}_{(n,n)} \mathbf{s}'$, and the invariant is preserved. □

5.8.3 Requirements for delayed simulation

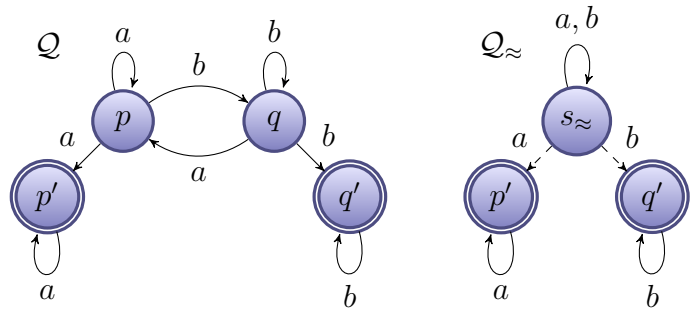
Quotienting w.r.t. delayed simulation is more difficult. We rule out several optimizations that are correct in the direct simulation case by considering several requirements peculiar to delayed simulation.

Non-maximal successors We have seen in Section 2.6.3 that, already for non-deterministic automata, transitions to non-maximal successors (of existential states) cannot be discarded with delayed simulation. Indeed, only considering maximal successors is incorrect since a visit to an accepting state might only occur by performing a non-maximal transition. See also Example 2.6.2. (This is not the case with direct simulation, since, if a simulation-smaller state is accepting, then every larger state is accepting as well.)

Requirement 1. *Consider all transitions from existential states.*

No fixed representatives for existential states One might wonder whether representatives can be fixed in advance in delayed simulation quotients, like in minimax quotients. This turns out to be incorrect for existential states. The reason is the same as above: Fixing a representative might prevent accepting states from being visited.

Example 5.8.2 - No fixed representatives for existential states



Consider the automaton $\mathcal{Q}$ above, which is a NBA. Clearly, $p \approx^{\mathrm{de}}_{(1,1)} q$. For example, if p does a and goes to the accepting state, then q can do a and go to p, from which the accepting state can be reached in the next round. Both p and q accept a^ω and b^ω.

The automaton on the right is the naïve quotient of $\mathcal{Q}$ w.r.t. $\approx^{\mathrm{de}}_{(1,1)}$, obtained by considering all transitions. If transitions are fixed to be those induced by p, then the dashed b-transition $s_{\approx} \xrightarrow{b} q'$ would disappear, and the quotient automaton would not accept b^ω anymore. Similarly, if q is taken as a representative, then the dashed a-transition $s_{\approx} \xrightarrow{a} p'$ would disappear and b^ω could not be recognized.

Therefore, fixed representatives cannot be chosen, and transitions from every existential state in a quotient class have to be considered.

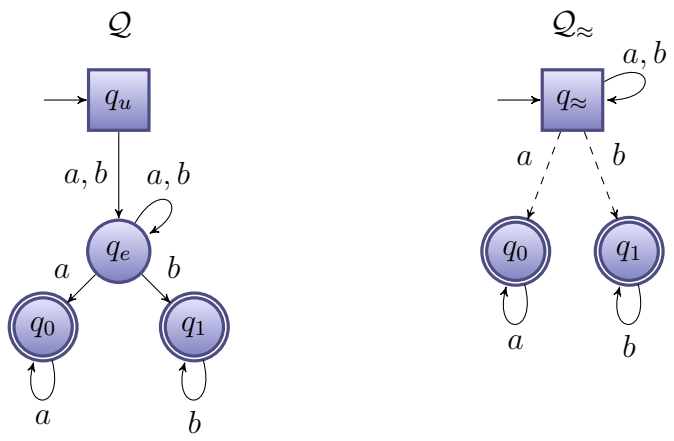
Requirement 2. *Consider transitions from all existential states.*

Combining this with the previous requirement, we have to consider *all* transitions from *all* existential states.

Existential mixed classes In minimax quotients, as a by-product of the ability of choosing representatives, mixed classes can be declared either universal or existential (by choosing an suitable representative). From the previous requirement, we know that, for delayed simulation, existential representatives cannot be fixed. However, it might still be the case that a mixed class could be declared either existential (and all transitions from equivalent existential states need to be considered), or universal (by picking a universal representative).

We show that this is not the case, and that mixed classes have to be declared necessarily *existential*. Intuitively, if a mixed class were to be declared universal, then, for correctness, non-maximal transitions from existential states should be discarded—otherwise, any non-maximal transition would overly restrict the behavior of the mixed class. On the other side, we know from Requiement 1 that all transitions from existential states need to be considered, thus, non-maximal transitions cannot be discarded. Therefore, mixed classes cannot be declared universal.

Example 5.8.3 - Mixed classes need to be declared existential



Consider the automaton $\mathcal{Q}$ above. Clearly,

$$q_u \approx^{\mathrm{de}}_{(1,1)} q_e, \quad q_0 \sqsubseteq^{\mathrm{de}}_{(1,1)} q_e \quad \text{and} \quad q_1 \sqsubseteq^{\mathrm{de}}_{(1,1)} q_e$$

Therefore, state q_0 is a non-maximal a-successor of q_e, and state q_1 is non-maximal b-successors of q_e, since, in both cases, q_e is the only maximal successor.

Now consider the quotient automaton $\mathcal{Q}_\approx$, obtained by identifying q_u and q_e in the quotient state $q_\approx = \{q_u, q_e\}$. State $q_\approx$ is a mixed class. Assume that it is declared universal. We need to decide whether the dashed transitions, which go to non-maximal successors of q_e, need to be included in the quotient or not. If we include them, they over-constrain the behavior of the mixed class (since it is universal), and $\mathcal{Q}_\approx$ would recognize the empty language. Yet, removing any of them would disconnect accepting states, and $\mathcal{Q}_\approx$ would still recognize the empty language. Therefore, if $q_\approx$ is declared universal, then no modification to the transition structure preserves the language.

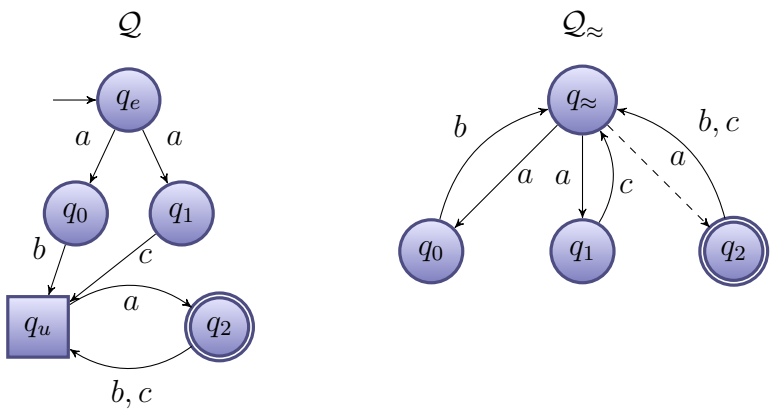
Requirement 3. *Declare mixed classes existential.*

Transitions from universal states in mixed classes This is the only requirement which is specific to multipebble simulations.

As we have seen, from Requiement 3, mixed classes need to be declared existential, and, from Requirements 1 and 2, *all* transitions from *all* existential states in a mixed class should be considered (i.e., no representatives and no pruning of successors).

It remains to decide what is the contribution of universal states in mixed classes. For the $(1, 1)$-simulations on ABA in [55], it is actually possible to ignore universal states altogether. In the multipebble context this is incorrect. The reason is similar as for Requiement 1, since ignoring such transitions might prevent visiting accepting states.

Example 5.8.4 - Transitions from universal states in mixed classes are needed



Consider the automaton $\mathcal{Q}$ above. States q_e and q_u are $(1,2)$-delayed simulation equivalent. Under quotienting, we get the mixed class $q_\approx = \{q_e, q_u\}$ in $\mathcal{Q}_\approx$.

The dashed transition $q_\approx \xrightarrow{a} q_2$ is due to the universal state q_u. If we remove it, $\mathcal{Q}_\approx$ would recognize the empty language, while $\mathcal{Q}$ has clearly non-empty language. Therefore, we cannot remove transitions induced by universal state in mixed classes.

Requirement 4. *Consider (minimal) transitions from universal states in mixed classes.*

5.8.4 Semielective quotients

By considering all requirements in the previous section, we obtain the following notion of semielective quotient [55].

1. Existential states induce transitions both in existential and in mixed classes. This is as in naïve quotients, and, by the examples in the previous section, it is unavoidable.

2. Universal states induce minimal transitions, both in universal and in mixed classes.

We can still fix representatives, but just for universal states. Formally, fix a function $\lceil\cdot\rceil : U \mapsto U$ s.t., for any universal state $q \in U$, its representative $\lceil q \rceil \in U$ is itself a universal state; moreover, for any other equivalent universal state $q' \in [q] \cap U$, $\lceil q \rceil = \lceil q' \rceil$. We define the *semielective* quotient automaton $\mathcal{Q}^{\text{se}}_\approx = (Q_\approx, \Sigma, q^I_\approx, \Delta^{\text{se}}_\approx, E^{\text{se}}_\approx, U^{\text{se}}_\approx, F_\approx)$ as follows. States in the quotient are equivalence classes of states, $Q_\approx = [Q]$, the initial state is $q^I_\approx = [q^I]$ and final states are those in $F_\approx = [F]$. Existential states are both existential and mixed classes, i.e., $E^{\text{se}}_\approx = [E]$, and universal states purely universal classes, i.e., $U^{\text{se}}_\approx = Q_\approx \setminus E^{\text{se}}_\approx$. $\Delta^{\text{se}}_\approx$ is defined as follows: $([q], a, [q']) \in \Delta^{\text{se}}_\approx$ iff either

- $\exists (q^E \in [q] \cap E, \widehat{q}' \in [q'])$ s.t. $(q^E, a, \widehat{q}') \in \Delta$, or

- $\exists (q^U \in [q] \cap U, \widehat{q}' \in [q'])$ s.t $(\lceil q^U \rceil, a, \widehat{q}') \in \Delta$ and $\widehat{q}' \in \min^{\preceq}_a(\lceil q^U \rceil)$.

All successors of existential states are considered, both in mixed and existential classes. For universal states, only their representatives induce transitions, and only to minimal successors.

Remark 5.8.14. The notion of semielective quotients from [55] does not use representatives. For the same reason as in Section 5.8.2.1, for $(1,1)$-simulations they coincide.

Remark 5.8.15. While the restriction to minimal successors of universal states in mixed classes is needed for correctness, this is not the case for universal classes. Nonetheless, it helps reducing the number of transitions in semielective quotients.

The reason why semielective quotients preserve the language is more subtle than for minimax quotients. In the latter case, we have seen in Section 5.8.2.1 that we can fix representatives, and, since equivalence classes inherit the type from the representative, there is no universal/existential mismatch for transitions in mixed classes.

However, in semielective quotients we cannot fix representatives, and mixed classes have to be declared existential. For $(1,1)$-simulations, by Lemma 5.8.5, transitions from universal states are still redundant. Therefore, one can ignore them, and there is no type mismatch.

Theorem 5.8.16 (Cf. Theorem 5 of [55])**.** *Delayed simulation is GFQ for semielective quotients.*

However, we have seen in Example 5.8.4 that, for multipebble simulation, transitions from universal states in mixed classes are necessary for correctness. This creates a *type mismatch* in mixed classes. Certainly, the restriction to minimal successors is necessary for correctness. Why it suffices follows from the fact that there is an essentially unique minimal transition (up to equivalence). In fact, $(1,1)$ and $(1,n)$ simulations admit *least successors*.

Definition 5.8.17 (Least successors)**.** *For a state s, we say that $s' \in \Delta(s,a)$ is an* $\preceq$-least a-successor of s*, if for every other a-successor $s'' \in \Delta(s,a)$, $s' \preceq s''$. Let* $\mathrm{least}_a^{\preceq}(s)$ *be the set of (necessarily $\preceq$-equivalent) a-least successors of s.*

Either there is no a-least successor, i.e., $\mathrm{least}_a^{\preceq}(s) = \emptyset$, or $\mathrm{least}_a^{\preceq}(s) = \min_a^{\preceq}(s)$ and a-minimal/least successors are all equivalent to each other. While a-least successors do not exist in general (k_0,k_1)-simulations, they do exist in the limit $(1,1)$ and $(1,n)$ cases for universal states which are simulation-equivalent to some existential state. (The $(1,1)$ case follows from Lemma 5.8.5.) Essentially, this says that there is a unique way of selecting a transition from a universal state in a mixed class.

Lemma 5.8.18. *Let $s \in U$ and $q \in E$. For $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$, if $q \approx^x_{(1,n)} s$, then there exists $s' \in \mathrm{least}_a^{\sqsubseteq^{\mathrm{x}}_{(1,\mathrm{n})}}(s)$. Consequently,* $\min_a^{\sqsubseteq^x_{(1,n)}}(s) = \mathrm{least}_a^{\sqsubseteq^{\mathrm{x}}_{(1,\mathrm{n})}}(s) \neq \emptyset$.

The lemma fails for both the $(1,k)$, with $1 < k < n$ and the $(k,1)$ case, with $k > 1$, as we show in Examples 5.8.5 and 5.8.6, respectively.

Proof. Let $s \in U$, $q \in E$, and $q \approx^x_{(1,n)} s$. From $s \sqsubseteq^x_{(1,n)} q$ and by the definition of simulation, there exists $\mathbf{q}' \subseteq \Delta(q, a)$ and $s' \in \Delta(s, a)$ s.t. $s' \sqsubseteq^x_{(1,n)} \mathbf{q}'$.

We show that s' is an a-least successor of s. Let s'' be any other a-successor of s. From $q \sqsubseteq^x_{(1,n)} s$ and since $s \in U$, by the definition of simulation it follows that, for any $q' \in \mathbf{q}'$, we have $q' \sqsubseteq^x_{(1,n)} s''$. Since $s' \sqsubseteq^x_{(1,n)} \mathbf{q}'$, and any element in $\mathbf{q}'$ is simulated by s'', by transitivity (cf. Theorem 5.6.1), $s' \sqsubseteq^x_{(1,n)} s''$. □

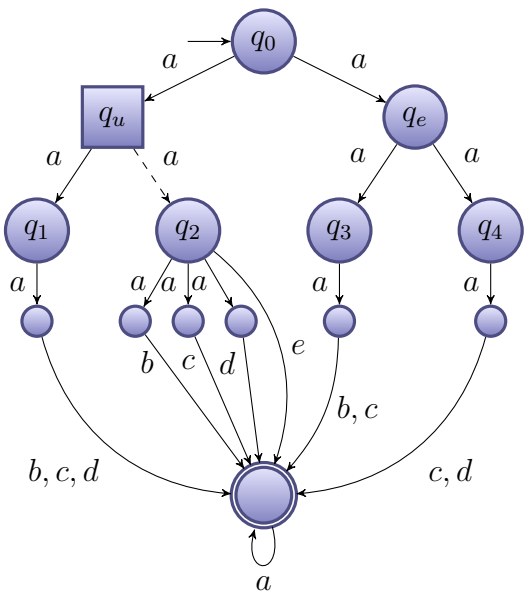
Therefore, for $(1, n)$-simulations there exists a unique equivalence class of minimal successors of universal states in mixed classes. This allows us to show that maximal $(1, n)$-delayed simulation is GFQ for semielective quotients.

Theorem 5.8.19. *$(1, n)$-delayed simulation is GFQ for semielective quotients.*

Remark 5.8.20. Alternatively, correctness follows from the fact that universal states in mixed classes are essentially deterministic in the original automaton, in the sense that they become so (up to equivalence) by removing non-minimal transitions.

Notice that, in Theorem 5.8.19, we state that the *maximal* $(1, n)$-delayed simulation is GFQ, where n is the number of states in the automaton. Perhaps surprisingly, this is tight, in the sense that neither $(1, k)$-delayed simulation for $1 < k < n$, nor $(k, 1)$-delayed simulation for $k > 1$ are GFQ in general. In both cases, Lem. 5.8.18 fails.

Example 5.8.5 - $(1, k)$-delayed simulation is not GFQ for $1 < k < n$



Unlike for NBA [42], quotienting ABA w.r.t. $(1,k)$-de simulation does not preserve the language of the automaton in general. The problem lies again in mixed classes, where it is absolutely necessary to select $(1,n)$-minimal transitions of universal states, and this can only be done by looking at the maximal $(1,n)$-simulation, and not just at any $(1,k)$-fragment thereof.

Consider the automaton above, and let $k = 2$. We have the following simulations:

- $q_u \sqsubseteq^{\mathrm{de}}_{(1,k)} q_e$: From configuration $\langle q_u, q_e\rangle$, Spoiler has to select input symbol a, and Duplicator selects transitions $q_u \xrightarrow{a} q_1$ and $q_e \xRightarrow{a} \{q_3, q_4\}$. From configuration $\langle q_1, \{q_3, q_4\}\rangle$, Duplicator clearly wins.

- $q_e \sqsubseteq^{\mathrm{de}}_{(1,k)} q_u$: On the right, if Spoiler selects transition $q_u \xrightarrow{a} q_1$, then Duplicator wins since q_2 can do (after one a step) b, c, d. Therefore, assume Spoiler selects transition $q_u \xrightarrow{a} q_2$.

  On the left, assume Spoiler selects transition $q_e \xrightarrow{a} q_3$ (the other case $q_e \xrightarrow{a} q_4$ is analogous). From configuration $\langle q_3, q_2\rangle$, Duplicator splits 2 pebbles and wins.

- $q_2 \not\sqsubseteq^{\mathrm{de}}_{(1,n)} q_1$: Spoiler can play action e.

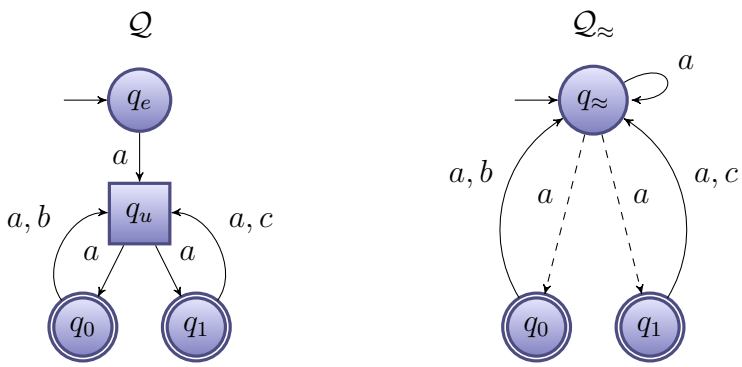
- $q_1 \sqsubseteq^{\mathrm{de}}_{(1,n)} q_2$: Duplicator wins by splitting pebbles to the three a-successors of q_2.

- $q_1 \not\sqsubseteq^{\mathrm{de}}_{(1,k)} q_2$: With only $k = 2$ pebbles, Duplicator loses.

Therefore, the two equivalent states q_u and q_e form a mixed class in the quotient. The only difference between the $(1,n)$- and $(1,k)$-semielective quotient is the dashed transition above: If $(1,n)$-simulation is considered, then state q_2 *is not* a minimal a-successor of q_u (since state q_1 is a strictly smaller successor) and the dashed transition is correctly discarded in the quotient. However, if $(1,k)$-simulation is considered, then q_2 becomes a minimal successor of q_u, (for $k = 2$, q_1 and q_2 become incomparable), and the dashed transition is now included. But this is incorrect, as the quotient automaton would accept the word $w = aaea^\omega \notin \mathcal{L}(q_0) = aaa\{b + c + d\}a^\omega$.

Technically, Lemma 5.8.18 fails for $(1,k)$-simulations: While q_1 is an a-least successor of q_u w.r.t. $(1,n)$-de simulation, there exist no such a-least successor w.r.t $(1,k)$-de simulation.

Example 5.8.6 - $(k, 1)$-delayed simulation is not GFQ for $k > 1$



Even $(k, 1)$-delayed simulation is not GFQ for semielective quotients, for $k > 1$. We actually show that in such a setting, *mixed classes cannot be declared existential*, and this holds already for $k = 2$. From Requiement 3, we know that mixed classes cannot be declared universal either, and this holds already for $(1, 1)$-simulation (hence, for $(k, 1)$-simulation). Therefore, there is no way of defining a suitable quotient structure as to make $(k, 1)$-delayed simulation GFQ.

Consider the automaton $\mathcal{Q}$ above. It recognizes the language $\mathcal{L}(\mathcal{Q}) = a^\omega$.

- $q_u \sqsubseteq^{\text{de}}_{(2,1)} q_i$, for $i = 0, 1$: Every time the left pebble is on q_u, Duplicator splits pebbles maximally to $\{q_0, q_1\}$. In the next round, the game is thus in configuration $\langle\{q_0, q_1\}, q_u\rangle$. Spoiler is forced to play action a, and she takes transition $q_u \xrightarrow{a} q_i$, with $i = 0, 1$. Thus, the game goes back to the initial configuration $\langle q_u, q_i\rangle$. Duplicator wins since she visits accepting states infinitely often.

- $q_u \sqsubseteq^{\text{de}}_{(2,1)} q_e$: Duplicator wins as above.

- $q_e \sqsubseteq^{\text{de}}_{(2,1)} q_u$: Spoiler is forced to play action a, and the game goes to a configuration of the form $\langle q_u, q_i\rangle$, from which Duplicator wins as above.

Therefore, if $\approx$ is the equivalence induced by $\sqsubseteq^{\text{de}}_{(2,1)}$, then $q_u \approx q_e$ and, in the quotient automaton $\mathcal{Q}_\approx$, $q_\approx = \{q_u, q_e\}$ is a mixed class. By Requiement 3, $q_\approx$ has to be declared existential in $\mathcal{Q}_\approx$. However, if $q_\approx$ is existential, then there is no way of choosing any subset of the two dashed transitions as to make $\mathcal{Q}_\approx$ recognize precisely a^ω.

Technically, Lemma 5.8.18 fails for $(k, 1)$-simulations: With $k > 1$ left pebbles, $q_u \in U$ has no a-least successor (even if $q_u \approx q_e \in E$), since no single transition

$q_u \xrightarrow{a} q_0$ or $q_u \xrightarrow{a} q_1$ alone subsumes transition $q_e \xrightarrow{a} q_u$. Indeed, only the multi-transition $q_u \overset{a}{\Longrightarrow} \{q_0, q_1\}$ subsumes the latter.

5.8.4.1 Correctness

In the rest of this section we prove Theorem 5.8.19. Actually, we prove the stronger statement that quotienting w.r.t $(1, n)$-delayed simulation not only preserves the language, but also $(1, n)$-*fair* simulation.

Theorem 5.8.21. *Let $\mathcal{Q}$ be an ABA with n states, and let $\mathcal{Q}'$ be its quotient w.r.t. $\sqsubseteq^{\mathrm{de}}_{(1,n)}$. Then, for any state q in $\mathcal{Q}$ and its corresponding quotient class $[q]_{\mathrm{se}}$ in $\mathcal{Q}'$, q is $(1, n)$-fair simulation equivalent to $[q]_{\mathrm{se}}$, i.e., $q \approx^{\mathrm{f}}_{(1,n)} [q]_{\mathrm{se}}$.*

Remark 5.8.22. It is possible to show that quotienting even preserves $(1, n)$-*delayed* simulation itself. However, for simplicity, we show it only for fair simulation.

The two directions of the equivalence in Theorem 5.8.21 are stated as Lemmas 5.8.25 and 5.8.28, respectively.

5.8.4.2 $\mathcal{Q}$ simulates $\mathcal{Q}_{\approx}$ (Lemma 5.8.25)

The general proof strategy here is as with Theorem 3 in [55] (cf. also Section 4.5): We show the existence of simulation-preserving strategies (which are not necessarily winning), and we obtain winning strategies via a modified composition of a simulation-preserving strategy and a winning strategy.

Respecting strategies For sets of states $\mathbf{q}$ and $\mathbf{s}$, we write $\mathbf{q} \preceq \mathbf{s}$ iff, for all $q \in \mathbf{q}$, $q \sqsubseteq^{\mathrm{de}}_{(1,n)} \mathbf{s}$. $\preceq$-respecting winning strategies (cf. Section 2.3) exist[6] in the basic simulation game between $[q]$ and any $\widehat{q} \in [q]$.

Lemma 5.8.23 (cf. Corollary 6 of [55])**.** *If $[q] \preceq \mathbf{s}$, then there exists a $\preceq$-preserving strategy σ for Duplicator in the basic simulation game between $[q]$ and $\mathbf{s}$. In particular, this holds when $\mathbf{s} = \{\widehat{q}\}$, for any $\widehat{q} \in [q]$.*

Proof. We write $q \sqsubseteq_\sigma \mathbf{s}$ when Duplicator wins the delayed simulation game $\mathbb{G}^{\mathrm{de}}(q, \mathbf{s})$ by following winning strategy σ. Let $\mathbb{G} = \mathbb{G}([q], \mathbf{s})$ be the basic simulation game between $[q]$ and $\mathbf{s}$, and, at round k, let the current configuration in $\mathbb{G}$ be $\langle [q_k], \mathbf{s}_k \rangle$. We prove $[q_k] \preceq \mathbf{s}_k$, and show a (memoryless) strategy ensuring this property.

[6] This is analogous to Corollary 4.5.6 in the context of jumping simulations.

By proceed by induction. At the initial round $[q_0] \preceq \mathrm{s}_0$ holds by assumption, where $q_0 = q$ and $\mathrm{s}_0 = \mathrm{s}$.

For the inductive step, assume $[q_k] \preceq \mathrm{s}_k$. Let Spoiler choose the next input symbol a_k. We consider two cases, depending on whether $[q_k]$ is existential or universal.

First case: $[q_k] \in E'$. Let Spoiler move as

$$([q_k], \mathrm{s}_k, a_k, [q_{k+1}], \mathrm{s}') \in \Gamma^{\mathrm{Sp}}_{\mathbb{G}}$$

By the definition of semielective quotient, there exist $\widehat{q} \in [q_k]$ and $q' \in [q_{k+1}]$ s.t. $q' \in \Delta(\widehat{q}, a)$. By induction hypothesis, $\widehat{q} \sqsubseteq^{\mathrm{de}}_{(1,n)} \mathrm{s}_k$. Let $\widehat{\sigma}$ be a winning strategy for Duplicator in $\mathbb{G}(\widehat{q}, \mathrm{s}_k)$. We distinguish two sub-cases, depending on whether $\widehat{q}$ is existential or universal.

- *First sub-case:* $\widehat{q} \in E$. Let Spoiler choose transition $(\widehat{q}, \mathrm{s}_k, a_k, q', \mathrm{s}') \in \Gamma^{\mathrm{Sp}}_{\mathbb{G}(\widehat{q}, \mathrm{s}_k)}$, and let s_{k+1} be the result of Duplicator playing according to strategy $\widehat{\sigma}$, i.e., $\widehat{\sigma}(\widehat{q}, \mathrm{s}_k)(a_k, q', \mathrm{s}') = (q', \mathrm{s}_{k+1})$. Clearly, $q' \sqsubseteq^{\mathrm{de}}_{(1,n)} \mathrm{s}_{k+1}$ since winning strategies are $\sqsubseteq^{\mathrm{de}}_{(1,n)}$-preserving. Therefore, by transitivity, $[q_{k+1}] \preceq \mathrm{s}_{k+1}$.

- *Second sub-case:* $\widehat{q} \in U$. Notice that Spoiler has already fixed a successor q' of $\widehat{q}$, but in the game $\mathbb{G}(\widehat{q}, \mathrm{s}_k)$, it is *Duplicator* that has to select a successor of $\widehat{q}$, and this is done according to the winning strategy $\widehat{\sigma}$. This is a mismatch between the two games.

  The issue is solved by noticing that, by the definition of semielective quotient, q' is not an arbitrary a_k-successor of $\widehat{q}$, but a *minimal* one, i.e., $q' \in \min^{\sqsubseteq^{\mathrm{de}}_{(1,n)}}_{a_k}(\widehat{q})$. Since $[q_k]$ is an existential state in the quotient containing a universal state $\widehat{q}$, it is a mixed class. Therefore, there exists an existential state $q^E \in [q_k] \cap E$ s.t. $\widehat{q} \approx q^E$. By Lemma 5.8.18, $\min^{\sqsubseteq^{\mathrm{de}}_{(1,n)}}_{a_k}(\widehat{q}) = \mathrm{least}^{\sqsubseteq^{\mathrm{de}}_{(1,n)}}_{a_k}(\widehat{q})$, therefore q' is a a_k-least successor of $\widehat{q}$. Now, let Spoiler select transition $(\widehat{q}, \mathrm{s}_k, a_k, \emptyset, \mathrm{s}') \in \Gamma^{\mathrm{Sp}}_{\mathbb{G}(\widehat{q}, \mathrm{s}_k)}$, and assume Duplicator's winning strategy $\widehat{\sigma}$ selects some state $\widehat{q}'$ as an a_k-successor of $\widehat{q}$: $\widehat{\sigma}(\widehat{q}, \mathrm{s}_k)(a_k, \emptyset, \mathrm{s}') = (\widehat{q}', \mathrm{s}'')$. But q' is an a_k-least successor, therefore $q' \sqsubseteq^{\mathrm{de}}_{(1,n)} \widehat{q}' \sqsubseteq^{\mathrm{de}}_{(1,n)} \mathrm{s}''$. By transitivity, we have $q' \sqsubseteq^{\mathrm{de}}_{(1,n)} \mathrm{s}''$. Take $\mathrm{s}_{k+1} := \mathrm{s}''$. Clearly, $[q_{k+1}] \preceq \mathrm{s}_{k+1}$.

In either case, $[q_{k+1}] \preceq \mathrm{s}_{k+1}$. Therefore, we define σ as

$$\sigma(\pi_k)(a_k, [q_{k+1}], \mathrm{s}') = ([q_{k+1}], \mathrm{s}_{k+1})$$

Second case: $[q_k] \in U'$. Let Spoiler choose transition

$$([q_k], \mathbf{s}_k, a_k, \emptyset, \mathbf{s}') \in \Gamma^{\mathrm{Sp}}_{\mathbb{G}}$$

By the definition of semielective quotient, $q_k \in U$. Let let $\lceil q_k \rceil \in U$ be the universal representative of q_k. By induction hypothesis, $\lceil q_k \rceil \sqsubseteq^{\mathrm{de}}_{(1,n)} \mathbf{s}_k$, and let $\widehat{\sigma}$ be a corresponding winning strategy for Duplicator. Let Spoiler choose transition $(\lceil q_k \rceil, \mathbf{s}_k, a_k, \emptyset, \mathbf{s}') \in \Gamma^{\mathrm{Sp}}_{\mathbb{G}(\lceil q_k \rceil, \mathbf{s}_k)}$. Duplicator replies according to $\widehat{\sigma}$: $\widehat{\sigma}(\lceil q_k \rceil, \mathbf{s}_k)(a_k, \emptyset, \mathbf{s}') = (q', \mathbf{s}'')$. Clearly, $q' \sqsubseteq^{\mathrm{de}}_{(1,n)} \mathbf{s}''$. Let q'' be any a-minimal successor of $\lceil q_k \rceil$ s.t. $q'' \sqsubseteq^{\mathrm{de}}_{(1,n)} q'$. By transitivity, $q'' \sqsubseteq^{\mathrm{de}}_{(1,n)} \mathbf{s}''$. Take $\mathbf{s}_{k+1} := \mathbf{s}''$ and $q_{k+1} := q''$. By minimality of q_{k+1} and the definition of semielective+ quotient, there exists an a_k-transition from $[q_k]$ to $[q_{k+1}]$. Thus, Duplicator's response can be defined as

$$\sigma(\pi_k)(a_k, \emptyset, \mathbf{s}') = ([q_{k+1}], \mathbf{s}_{k+1})$$

Clearly, $[q_{k+1}] \preceq \mathbf{s}_{k+1}$. This concludes the description of the second case. □

The following lemma states that composition preserves $\preceq$-respecting strategies, and, moreover, if there is an initial pending obligation, then that obligation will be satisfied.[7]

Lemma 5.8.24. *Let $[q] \preceq r \sqsubseteq^{\mathrm{de}}_{(1,n)} \mathbf{s}$, and let σ_0 be a $\preceq$-respecting strategy in $\mathbb{G}([q], r)$ and σ_1 is a winning strategy in $\mathbb{G}_{\mathrm{de}}(r, \mathbf{s})$. Define $\sigma := \sigma_0 \bowtie \sigma_1$. Then,*

1. *σ is $\preceq$-respecting in $\mathbb{G}([q], \mathbf{s})$.*
2. *If $r \in F$, then for any σ-conform play $\langle [q_0], \mathbf{s}_0 \rangle \overset{a_0}{\Longrightarrow} \langle [q_1], \mathbf{s}_1 \rangle \overset{a_1}{\Longrightarrow} \cdots$ starting at $\langle [q_0], \mathbf{s}_0 \rangle = \langle [q], \mathbf{s} \rangle$, there exists $j \geq 0$ s.t. $\mathbf{s}_0 \Longrightarrow^{\exists}_F \mathbf{s}_j$.*

Proof. For Point 1), consider the definition of the logbook as in the proof of transitivity (cf. Section 5.6.3), and the shape it takes in the $(1, n)$-simulation case. At round k, if the outer game is in configuration $\langle [q_k], \mathbf{s}_k \rangle$, then the unique left game is in configuration $\langle [q_k], \mathbf{r}_k \rangle$ for some $\mathbf{r}_k$. Moreover, for any $r_{k,i} \in \mathbf{r}_k$, $\langle r_{k,i}, \mathbf{s}_k \rangle$ is the current configuration of some right game, where the pebbles in $\mathbf{s}_k$ are the same in all right games. This is with no loss of generality, as we can reason as in assumption (FA2). Since σ_0 is $\preceq$-respecting, $[q_k] \preceq \mathbf{r}_k$. Since σ_1 is winning, and winning strategies are $\sqsubseteq^{\mathrm{de}}_{(1,n)}$-respecting, $r_{k,i} \sqsubseteq^{\mathrm{de}}_{(1,n)} \mathbf{s}_k$. Therefore, $\mathbf{r}_k \preceq \mathbf{s}_k$, which implies $[q_k] \preceq \mathbf{s}_k$.

For Point 2), further assume $r \in F$. Consider any play $\pi^R_{\omega,g} = \psi^R_{\omega,g} \times \rho^R_{\omega}$ in the limit right logbook B^R_{ω}, where $\psi^R_{\omega,g} = r_0 \xrightarrow{a_0} r_1 \xrightarrow{a_1} \cdots$ with $r_0 = r$, and $\rho^R_{\omega} = \mathbf{s}_0 \overset{a_0}{\Longrightarrow} \mathbf{s}_1 \overset{a_1}{\Longrightarrow} \cdots$. Since $\psi^R_{\omega,g}$ is rooted at $r \in F$ and $\pi^R_{\omega,g}$ is conform to the winning strategy σ_1, there exists $j \geq 0$ s.t. $\mathbf{s}_0 \Longrightarrow^{\exists}_F \mathbf{s}_j$. □

[7] This is analogous to Lemmas 4.5.7 and 4.5.8 in the context of jumping simulations.

We are now ready to prove the first direction of Theorem 5.8.21.

Lemma 5.8.25. *If* $q \sqsubseteq^{\mathrm{de}}_{(1,n)} \mathbf{s}$*, then* $[q]_{\mathrm{se}} \sqsubseteq^{\mathrm{f}}_{(1,n)} \mathbf{s}$*, where the quotient is taken w.r.t.* $\sqsubseteq^{\mathrm{de}}_{(1,n)}$.

Proof. At any round k,

- Fix a representative $q_k^F \in [q_k]$ s.t. $q_k^F \in F$ is accepting if $[q_k] \in F'$.
- For any $q \in [q_k]$, fix a $\preceq$-preserving strategy $\sigma^0_{[q_k],q}$ in the basic simulation game $\mathbb{G}([q_k], q)$ (which exists by Lemma 5.8.23).
- For every $q \in [q_k]$, fix a winning strategy for Duplicator $\sigma^{\mathrm{de}}_{q,\mathbf{s}_k}$ in the delayed simulation game $\mathbb{G}^{\mathrm{de}}_{(1,n)}(q, \mathbf{s}_k)$.

The idea is to use a modified join of a $\preceq$-preserving strategy and a de-winning strategy [55]. First, we keep track of the oldest pending obligation:

$$i(k) := \min\big(\{i \leq k \mid [q_i] \in F' \wedge \forall (i \leq j \leq k) \ \cdot \ \mathbf{s}_j \not\Longrightarrow^{\exists}_F \mathbf{s}_k\} \cup \{k\}\big)$$

That is, at round k, $i(k)$ is the oldest obligation which is still not satisfied, and $i(k) = k$ if no obligation is currently pending. Intuitively, Duplicator tries to satisfy the oldest pending obligation. Meanwhile, if more obligations do arise, they are simply discarded, and index $i(k)$ does not change. Therefore, Duplicator is not winning w.r.t. the delayed acceptance condition, as she might "skip" some obligation (but only finitely many). However, if infinitely many obligations do arise, then Duplicator satisfies infinitely many of them (in fact, all of them by Lemma 5.4.1).

σ is defined via the following modified join:

$$\sigma(\pi_k) := (\sigma^0_{[q_{i(k)}],q^F_{i(k)}} \bowtie \sigma^{\mathrm{de}}_{q^F_{i(k)},\mathbf{s}_{i(k)}})(\pi_k[i(k), k])$$
$$\pi_k[i(k), k] := \langle [q_{i(k)}], \mathbf{s}_{i(k)} \rangle \overset{a_{i(k)}}{\Longrightarrow} \langle [q_{i(k)+1}], \mathbf{s}_{i(k)+1} \rangle \overset{a_{i(k)+1}}{\Longrightarrow} \cdots \langle [q_k], \mathbf{s}_k \rangle$$

where $\pi_k[i(k), k]$ is the subsequence of π_k starting at $i(k)$. This achieves two goals:

(Goal-0) To satisfy the oldest obligation (achieved by $\sigma^{\mathrm{de}}_{q^F_{i(k)},\mathbf{s}_{i(k)}}$), and

(Goal-1) To preserve the capability of satisfying future obligations (achieved by $\sigma^0_{[q_{i(k)}],q^F_{i(k)}}$).

By Lemma 5.8.24, Point 1), σ is $\preceq$-respecting. This means that $[q_i] \preceq \mathbf{s}_i$ at any round i; in particular, $[q_{i(k)}] \preceq \mathbf{s}_{i(k)}$, which implies $q^F_{i(k)} \sqsubseteq^{\mathrm{de}}_{(1,n)} \mathbf{s}_{i(k)}$. Therefore, the strategy $\sigma^{\mathrm{de}}_{q^F_{i(k)},\mathbf{s}_{i(k)}}$ exists and σ is well-defined. Thus, simulation is preserved at any round, and (Goal-1) is achieved.

(Goal-0) is achieved as well. Let $\pi = \langle [q], \mathrm{s} \rangle \xRightarrow{a_0} \langle [q_1], \mathrm{s}_1 \rangle \xRightarrow{a_1} \cdots$ be any σ-conform play, and, at round k, let $i(k) \leq k$ be the oldest pending obligation. As long as this obligation is not satisfied, i.e., $\mathrm{s}_{i(k)} \not\Longrightarrow_F^{\exists} \mathrm{s}_k$, π is conform to strategy $\sigma^0_{[q_h], q_h^F} \bowtie \sigma^{\mathrm{de}}_{q_h^F, \mathrm{s}_h}$. By Lemma 5.8.24, Point 2), there exists $j \geq i(k)$ s.t. $\mathrm{s}_{i(k)} \Longrightarrow_F^{\exists} \mathrm{s}_j$. □

Remark 5.8.26. Let us remark again that the proof above fails to show the stronger property $[q]_{\mathrm{se}} \sqsubseteq^{\mathrm{de}}_{(1,n)} \mathrm{s}$, since the strategy σ as defined above may miss intermediate obligations. In fact, σ only preserves infinitely many obligations.

5.8.4.3 $\mathcal{Q}_{\approx}$ simulates $\mathcal{Q}$ (Lemma 5.8.28)

$(1, n)$-equivalent universal states can mimic each other's minimal transitions, and go to $(1, n)$-equivalent states. It is an immediate consequence of the fact that right pebbles cannot be split on universal states.

Lemma 5.8.27 (cf. Corollary 3, Point 2 of [55])**.** *Let $q, s \in U$. For $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$, if $q \approx^x_{(1,n)} s$, then, for any $q' \in \min_a^{\sqsubseteq^x_{(1,n)}}(q)$, there exists $s' \in \min_a^{\sqsubseteq^x_{(1,n)}}(s)$ s.t. $q' \approx^x_{(1,n)} s'$.*

Proof. For states q and s in U, let $q \approx^x_{(1,n)} s$, and, for another state q', let $q' \in \min_a^{\sqsubseteq^x_{(1,n)}}(q)$. By the definition of simulation, there exists $s' \in \Delta(s, a)$ s.t. $s' \sqsubseteq^x_{(1,n)} q'$.

We show that any such s' is in fact an x-minimal a-successor of s. Let $s'' \in \Delta(s, a)$ be any other a-successor of s s.t. $s'' \sqsubseteq^x_{(1,n)} s'$, and we have to show $s' \sqsubseteq^x_{(1,n)} s''$ as well. Since $q \sqsubseteq^x_{(1,n)} s$, from the definition of simulation, there exists $q'' \in \Delta(q, a)$ s.t. $q'' \sqsubseteq^x_{(1,n)} s''$. Hence, we have the following chain of inclusions:

$$q'' \sqsubseteq^x_{(1,n)} s'' \sqsubseteq^x_{(1,n)} s' \sqsubseteq^x_{(1,n)} q'$$

By transitivity (cf. Theorem 5.6.1), $q'' \sqsubseteq^x_{(1,n)} q'$, and, by the minimality of q', $q' \sqsubseteq^x_{(1,n)} q''$. By transitivity again, all states in $\{q'', s'', s', q'\}$ are equivalent. Therefore, $s' \sqsubseteq^x_{(1,n)} s''$ and $q' \approx^x_{(1,n)} s'$. □

We now prove the other direction of Theorem 5.8.21. While it suffices to show that $[q]_{\mathrm{se}}$ $(1, n)$-fair simulates q, we actually prove that $[q]_{\mathrm{se}}$ $(1, 1)$-*direct* simulates q.

Lemma 5.8.28. *For any $q \in Q$, $q \sqsubseteq^{\mathrm{di}}_{(1,1)} [q]_{\mathrm{se}}$.*

We can prove a stronger statement since more transitions are available in semielective quotients than in minimax quotients—in particular, we exploit minimal transitions from universal states in mixed classes. Moreover, the statement holds when quotienting w.r.t. even fair, or ordinary simulation.

Proof. We maintain the following invariant: If $(s_k, [q_k])$ is the current configuration in $\mathbb{G}^{\mathrm{di}}_{(1,1)}(q, [q])$, then $s_k \in [q_k]$. Clearly, the invariant implies that the winning condition for direct simulation is satisfied: If $s_k \in F$, then $[q_k] \in F'$.

The initial configuration is $(s_0, [q_0])$ with $s_0 = q$, and $[q_0] = [q]$, and the invariant clearly holds since $q \in [q]$. Inductively, assume the current configuration is $(s_k, [q_k])$ and the invariant $s_k \in [q_k]$ holds. We distinguish three different cases.

Case 1: $s_k \in E$. By the definition of semielective quotient, $[q_k] \in E'$. Assume Spoiler chooses transition

$$(s_k, [q_k], a_k, s_{k+1}, \emptyset) \in \Gamma^{\mathrm{Sp}}$$

From $(s_k, a_k, s_{k+1}) \in \Delta$, the invariant $s_k \in [q_k]$ and by the definition of semielective quotient, there exists a transition $([q_k], a_k, [s_{k+1}]) \in \Delta^{\mathrm{se}}_{\approx}$. Thus, Duplicator can select transition

$$(s_k, [q_k], a_k, s_{k+1}, \emptyset, s_{k+1}, [s_{k+1}]) \in \Gamma^{\mathrm{Dup}}$$

Clearly $s_{k+1} \in [s_{k+1}]$, and the invariant is preserved. We have not used any property of simulation here, just the definition of semielective quotients.

Case 2: $s_k \in U \wedge [q_k] \in E'$. Spoiler only chooses a_k,

$$(s_k, [q_k], a_k, \emptyset, \emptyset) \in \Gamma^{\mathrm{Sp}}$$

Let $\lceil s_k \rceil \in U$ be the representative of s_k, and let $s' \in \min^{\sqsubseteq^{\mathrm{de}}_{(1,n)}}_{a_k}(\lceil s_k \rceil)$ be any de-minimal a_k-successor of $\lceil s_k \rceil$. By the definition of semielective quotient and by the minimality of s', there exists a transition $([q_k], a_k, [s'])$ $\in \Delta^{\mathrm{se}}_{\approx}$. Moreover, since $\lceil s_k \rceil \approx^{\mathrm{de}}_{(1,n)} s_k$, by Lemma 5.8.27, there exists $s_{k+1} \in \min^{\sqsubseteq^{\mathrm{de}}_{(1,n)}}_{a_k}(s_k)$ s.t. $s_{k+1} \approx^{\mathrm{de}}_{(1,n)} s'$. Therefore, $[s'] = [s_{k+1}]$ and Duplicator can choose transition

$$(s_k, [q_k], a_k, \emptyset, \emptyset, s_{k+1}, [s']) \in \Gamma^{\mathrm{Dup}}$$

Obviously, $s_{k+1} \in [s'] = [s_{k+1}]$, and the invariant is preserved.

Case 3: $s_k \in U \wedge [q_k] \in U'$. We use the minimality of successors of universal states in universal classes. Assume Spoiler chooses transition

$$(s_k, [q_k], a_k, \emptyset, [q_{k+1}]) \in \Gamma^{\mathrm{Sp}}$$

From the definition of quotient, there exists a transition $(\lceil q_k \rceil, a_k, q_{k+1}) \in \Delta$ s.t. $q_{k+1} \in \min^{\sqsubseteq^{\mathrm{de}}_{(1,n)}}_{a_k}(\lceil q_k \rceil)$. From the invariant $s_k \in [q_k]$, we have $s_k \approx^{\mathrm{de}}_{(1,n)} \lceil q_k \rceil$. By

Lemma 5.8.27, there exists $s_{k+1} \in \min_{a_k}^{\sqsubseteq^{\mathrm{de}}_{(1,n)}}(s_k)$ s.t. $s_{k+1} \approx^{\mathrm{de}}_{(1,n)} q_{k+1}$. Therefore, Duplicator can select transition

$$(s_k, [q_k], a_k, \emptyset, [q_{k+1}], s_{k+1}, [q_{k+1}]) \in \Gamma^{\mathrm{Dup}}$$

s.t. $s_{k+1} \in [q_{k+1}]$, thus preserving the invariant. □

5.9 Computing multipebble simulations

In this section, we show how to solve multipebble simulation games. We encode each simulation game into a finite 2-player arena with a ω-regular winning condition. The winning region of Duplicator is then characterized by a (fixed) fixpoint formula φ.

In each case, for a fixed number of pebbles k, we get an arena of size polynomial in n, over which φ can be evaluated in polynomial time. Therefore, k-pebble simulation is in PTIME for fixed $k > 0$.

We show that these constructions imply that finite-memory strategies suffice for Duplicator. Better bounds might be given (i.e., memoryless strategies), but our finite-memory characterization suffices in the rest of the chapter.

The approach is similar for direct, delayed and fair simulation. Let $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$. We define a finite x-arena $\mathcal{G}^x = (P^x, P_0^x, P_1^x, p_I^x, \Gamma^x, \Gamma_0^x, \Gamma_1^x)$, where

- $P^x = P_0^x \cup P_1^x$ is the set of configurations,
- P_0^x is the set of Spoiler's configurations,
- P_1^x is the set of Duplicator's configurations,
- $p_I^x \in P_0^x$ is the initial configuration,
- $\Gamma^x = \Gamma_0^x \cup \Gamma_1^x$ is the set of transitions,
- $\Gamma_0^x \subseteq P_0^x \times P_1^x$ is the set of transitions of Spoiler, and
- $\Gamma_1^x \subseteq P_1^x \times P_0^x$ is the set of transitions of Duplicator.

We take the view of Duplicator. In the following, let $\mathrm{cpre}^x(\cdot) : 2^{P_0^x} \mapsto 2^{P_0^x}$ be a monotone controlled predecessor operator s.t., for any $X \subseteq P_0^x$, $\mathsf{CPre}^x(X)$ is the set of configurations from where Duplicator can force the game into X in one round:

$$\mathsf{CPre}^x(X) := \{p_0 \in P_0^x \mid \forall((p_0, p_1) \in \Gamma_0^x) \cdot \exists((p_1, p_0') \in P_1^x) \cdot p_0' \in X\}$$

5.9.1 Direct simulation

Fix an ABA $\mathcal{Q}$. The arena $\mathcal{G}^{\mathrm{di}}$ for direct simulation is the same as in the definition of the associated ordinary simulation game (cf. Section 5.3). That is, $P_0^{\mathrm{di}} = P_{\mathsf{Sp}}$, $P_1^{\mathrm{di}} = P_{\mathsf{Dup}}$, $\Gamma_0^{\mathrm{di}} = \Gamma^{\mathsf{Sp}}$ and $\Gamma_1^{\mathrm{di}} = \Gamma^{\mathsf{Dup}}$. The winning condition for direct simulation is a *safety condition*. Let T^{di} be the set of safe configurations,

$$T^{\mathrm{di}} = \{\langle \mathbf{q}, \mathbf{s}\rangle \mid \mathbf{q} \cap F \neq \emptyset \text{ implies } \mathbf{s} \subseteq F\}$$

Then, $\mathbf{q} \sqsubseteq^{\mathrm{di}}_{(k_1,k_2)} \mathbf{s}$ iff Duplicator can ensure never leaving T^{di} when starting from the initial configuration $p_I^{\mathrm{di}} := \langle \mathbf{q}, \mathbf{s}\rangle$. This can be verified by checking whether $p_I^{\mathrm{di}} \in W^{\mathrm{di}}$, where

$$W^{\mathrm{di}} = \nu X \ \cdot \ T^{\mathrm{di}} \cap \mathsf{CPre}^x(X)$$

5.9.2 Fair simulation

The arena for fair simulation $\mathcal{G}^{\mathrm{f}}$ is similar to the one for direct simulation, with the difference that we need extra bookkeeping for recording whether pebbles have visited accepting states or not.

- Spoiler's configurations are of the form $\langle \mathbf{q}, \mathbf{b}, \mathbf{s}, \mathbf{g}\rangle \in P_0^{\mathrm{f}}$, with $\mathbf{q}, \mathbf{b}, \mathbf{s}, \mathbf{g} \subseteq Q$.
- Duplicator's configurations are of the form $\langle \mathbf{q}, \mathbf{b}, \mathbf{s}, \mathbf{g}, a, \mathbf{q}', \mathbf{s}'\rangle \in P_1^{\mathrm{f}}$, with $\mathbf{q}, \mathbf{b}, \mathbf{s}, \mathbf{g}, \mathbf{q}', \mathbf{s}' \subseteq Q$ and $a \in \Sigma$.

The additional sets $\mathbf{b}$ and $\mathbf{g}$ are always a subset of $\mathbf{q}$ and $\mathbf{s}$, respectively, and they keep track of which pebbles in $\mathbf{q}$ and $\mathbf{s}$ are, respectively, "bad" or "good". When $\mathbf{b}_j = \emptyset$ holds at round j, there are no more bad left pebbles, and we say that *the red light flashes*. This event witnesses the existence of $i \leq j$ s.t. $\mathbf{q}_i \Longrightarrow^{\forall}_F \mathbf{q}_j$. Similarly, when $\mathbf{g}_j = \mathbf{s}_j$, all right pebbles are good, we say that *the green light flashes*, implying that there exists $i \leq j$ s.t. $\mathbf{s}_i \Longrightarrow^{\exists}_F \mathbf{s}_j$. This terminology is borrowed from [104]. After a light flashes, the corresponding set $\mathbf{b}$/$\mathbf{g}$ is reset to an initial condition in order to start tracking new bad/god pebbles. Then, the winning condition for fair simulation requires that if the red light flashes infinitely often, then the green light flashes infinitely often.

Formally, we have the following transitions. For any transition in the simulation game $(\mathbf{q}, \mathbf{s}, a, \mathbf{q}', \mathbf{s}') \in \Gamma^{\mathsf{Sp}}$, there is a transition in the arena

$$(\langle \mathbf{q}, \mathbf{b}, \mathbf{s}, \mathbf{g}\rangle, \langle \mathbf{q}, \mathbf{b}, \mathbf{s}, \mathbf{g}, a, \mathbf{q}', \mathbf{s}'\rangle) \in \Gamma_0^{\mathrm{f}}$$

Similarly, for any $(\mathbf{q}, \mathbf{s}, a, \mathbf{q}', \mathbf{s}', \mathbf{q}'', \mathbf{s}'') \in \Gamma^{\mathrm{Dup}}$, we have

$$(\langle \mathbf{q}, \mathbf{b}, \mathbf{s}, \mathbf{g}, a, \mathbf{q}', \mathbf{s}' \rangle, \langle \mathbf{q}'', \mathbf{b}', \mathbf{s}'', \mathbf{g}' \rangle) \in \Gamma_1^{\mathrm{f}}$$

where the new sets $\mathbf{b}'$ and $\mathbf{g}'$ are defined as follows:

$$\mathbf{g}' = \begin{cases} \mathbf{s}'' \cap F & \text{if } \mathbf{g} = \mathbf{s} \\ \{s \in \mathbf{s}'' \mid s \in F \text{ or } s \in \Delta(\mathbf{g}, a)\} & \text{otherwise} \end{cases}$$

$$\mathbf{b}' = \begin{cases} \mathbf{q}'' \setminus F & \text{if } \mathbf{b} = \emptyset \\ \{q \in \mathbf{q}'' \mid q \notin F \text{ and } q \in \Delta(\mathbf{b}, a)\} & \text{otherwise} \end{cases}$$

Intuitively, bad states in $\mathbf{b}'$ are those states in $\mathbf{q}'$ which are not accepting and with some bad predecessor in $\mathbf{b}$. Notice the similarity of the this update rule with the MH-construction from Section 5.7.1.1. Similarly, good states in $\mathbf{g}'$ are those states in $\mathbf{s}'$ which are either accepting, or with some good predecessor in $\mathbf{g}$. This rule comes from the fair subset construction of Section 5.7.2.1.

Let $T_0^{\mathrm{f}} = \{\langle \mathbf{q}, \mathbf{b}, \mathbf{s}, \mathbf{g} \rangle \mid \mathbf{b} = \emptyset\}$ be the set of configurations where the red light flashes, and let $T_1^{\mathrm{f}} = \{\langle \mathbf{q}, \mathbf{b}, \mathbf{s}, \mathbf{g} \rangle \mid \mathbf{g} = \mathbf{s}\}$ be the set of configurations where the green light flashes. The winning criterion for fair simulation is translated in a 1-pair Street condition (also known as a *reactivity condition* [87]): If T_0^{f} is visited infinitely often, then T_1^{f} is visited infinitely often. Therefore, $\mathbf{q} \sqsubseteq^{\mathrm{f}}_{(k_1,k_2)} \mathbf{s}$ iff $p_I^{\mathrm{f}} \in W^{\mathrm{f}}$, where $p_I^{\mathrm{f}} = \langle \mathbf{q}, \mathbf{q} \setminus F, \mathbf{s}, \mathbf{s} \cap F \rangle$, and

$$W^{\mathrm{f}} = \nu X \cdot \mu Y \cdot \nu Z \cdot (T_1^{\mathrm{f}} \cap \mathsf{CPre}^{\mathrm{f}}(X)) \cup (T_0^{\mathrm{f}} \cap \mathsf{CPre}^{\mathrm{f}}(Y)) \cup ((P_0^{\mathrm{f}} \setminus T_0^{\mathrm{f}}) \cap \mathsf{CPre}^{\mathrm{f}}(Z))$$

The correctness of the construction is established with the following observation:

Lemma 5.9.1. *Let $\pi = p_0 \overset{a_0}{\Longrightarrow} p_1 \overset{a_1}{\Longrightarrow} \cdots$ be an infinite sequence of configurations in a play of $\mathcal{G}^{\mathrm{f}}$, with $p_i = \langle \mathbf{q}_i, \mathbf{b}_i, \mathbf{s}_i, \mathbf{g}_i \rangle$. Let $\pi_0 = \mathbf{q}_0 \overset{a_0}{\Longrightarrow} \mathbf{q}_1 \overset{a_1}{\Longrightarrow} \cdots$ and $\pi_1 = \mathbf{s}_0 \overset{a_0}{\Longrightarrow} \mathbf{s}_1 \overset{a_1}{\Longrightarrow} \cdots$ be the projections of π to the first and third component, respectively. Then, for any $j \geq 0$,*

$$\text{if } p_j \in T_0^{\mathrm{f}}, \text{ then } \forall (i < j, p_i \in T_0^{\mathrm{f}}) \cdot q_i \Longrightarrow_F^{\forall} q_j$$
$$\text{if } p_j \in T_1^{\mathrm{f}}, \text{ then } \forall (i < j, p_i \in T_1^{\mathrm{f}}) \cdot q_i \Longrightarrow_F^{\exists} q_j$$

In particular,

- *π_0 is universally fair iff the red light flashes infinitely often, i.e., iff $p_i \in T_0^{\mathrm{f}}$ for infinitely many i's.*
- *π_1 is existentially fair iff the green light flashes infinitely often, i.e., iff $p_i \in T_1^{\mathrm{f}}$ for infinitely many i's.*

Finite memory strategies suffice for Duplicator The set W^{f} is Duplicator's winning region in a parity game with three priorities. Since parity games are memoryless determined [41], Duplicator has memoryless winning strategies in $\mathcal{G}^{\mathrm{f}}$. A memoryless strategy in $\mathcal{G}^{\mathrm{f}}$ corresponds to a finite-memory strategy in the original simulation game $\mathbb{G}^{\mathrm{f}}$. This latter strategy is obtained by keeping track of the extra sets $\mathbf{b}$ and $\mathbf{g}$ encoded in $\mathcal{G}^{\mathrm{f}}$. Since there are at most $2^n \cdot 2^n$ such pairs of sets, the memory needed in $\mathbb{G}^{\mathrm{f}}$ can be uniformly bounded by 2^{2n}.

Lemma 5.9.2. *If Duplicator has a winning strategy in $\mathbb{G}^{\mathrm{f}}$, then she has a winning strategy $\hat{\sigma}$ of memory bounded by 2^{2n}.*

We do not claim that the upper bound above is optimal, but it suffices for establishing the complexity results of Section 5.10.

5.9.3 Delayed simulation

In fair simulation, the red and green lights flash independently of each other. This suffices there, since the winning condition for fair simulation is does not depend on exactly *when* such flashes occur, as it can only observe whether flashes appear infinitely often or not. In delayed simulation, the winning condition is more subtle, and timing matters now: Indeed, every time the red light flashes, then eventually the green light has to flash. Such obligations are not cumulative, in the sense that every red flash has to be satisfied separately by a green flash.

For simplicity, we do not present a general solution for the (k_0, k_1)-case, and we deal just with the $(1, k)$-case. Indeed, for approximating language inclusion, one should use the coarser fair simulation, and, from the results about quotienting of Section 5.8, (k_0, k_1)-delayed simulation is not GFQ for $k_0 > 1$. Thus, one rarely needs to compute arbitrary (k_0, k_1)-delayed simulation

The winning condition for delayed simulation requires that, every time the red light flashes, a separate green light has to flash. In principle, one should record a new obligation for the green light each time the red light flashes. However, by Lemma 5.4.1,

$$\text{If } i_0 \leq i_1 \text{ and } \mathbf{s}_{i_1} \Longrightarrow_F^{\exists} \mathbf{s}_j, \text{ then } \mathbf{s}_{i_0} \Longrightarrow_F^{\exists} \mathbf{s}_j,$$

that is, a green flash a time i_1 subsumes a green flash at a previous time i_0. Therefore, once the green light flashes at some round, we also have a green flash at all previous rounds. This suggests that all previous obligations are immediately satisfied.

The algorithm of [42] Based on this intuition, the algorithm in [42] tries to satisfy just the most recent red flash:

When the red light flashes, reset right good pebbles.

That is, every time a new obligation arrives, one forgets the previous one(s) and starts afresh. This is *sound* in the sense that, if a green light flashes, then all previous obligations are satisfied. However, if the red light flashes too quickly, then right pebbles are continually reset, and they might not have enough time to visit accepting states before the next red flash. Therefore, the update rule of [42] is not *complete*, and it actually computes an *under-approximation* to $(1, k)$-delayed simulation.

The (corrected) algorithm The issue of completeness is solved by bookkeeping two different pending obligations: 1) The oldest pending obligation, called the *principal obligation*, and 2) The most recent pending obligation, called the *fresh obligation*. We say that the green light flashes when the principal obligation is satisfied. The principal and fresh obligations are updated as follows:

When the red light flashes, reset right good pebbles for the fresh obligation. When the green light flashes, the fresh obligation becomes the new principal obligation.

Formally, let $\{q\}, \mathbf{s}, \mathbf{g}, \mathbf{h}, \mathbf{q}', \mathbf{s}' \subseteq Q$ and $a \in \Sigma$. Then,

- Spoiler's configurations in $\mathcal{G}^{\mathrm{de}}$ are of the form $\langle \{q\}, \mathbf{s}, \mathbf{g}, \mathbf{h} \rangle \in P_0^{\mathrm{de}}$.
- Duplicator's configurations in $\mathcal{G}^{\mathrm{de}}$ are of the form $\langle \{q\}, \mathbf{s}, \mathbf{g}, \mathbf{h}, a, \mathbf{q}', \mathbf{s}' \rangle \in P_1^{\mathrm{de}}$.

Good pebbles for the principal and fresh obligation are recorded in sets $\mathbf{g}$ and $\mathbf{h}$, respectively. Transitions are as follows: If $(\{q\}, \mathbf{s}, a, \mathbf{q}', \mathbf{s}') \in \Gamma^{\mathrm{Sp}}$, then

$$(\langle \{q\}, \mathbf{s}, \mathbf{g}, \mathbf{h} \rangle, \langle \{q\}, \mathbf{s}, \mathbf{g}, \mathbf{h}, a, \mathbf{q}', \mathbf{s}' \rangle) \in \Gamma_0^{\mathrm{de}}$$

and, if $(\{q\}, \mathbf{s}, a, \mathbf{q}', \mathbf{s}', \{q''\}, \mathbf{s}'') \in \Gamma^{\mathrm{Dup}}$, then

$$(\langle \{q\}, \mathbf{s}, \mathbf{g}, \mathbf{h}, a, \mathbf{q}', \mathbf{s}' \rangle, \langle \{q''\}, \mathbf{s}'', \mathbf{g}'', \mathbf{h}'' \rangle) \in \Gamma_1^{\mathrm{de}}$$

, where the new sets $\mathbf{g}''$ and $\mathbf{h}''$ are updated as follows:

$$\mathbf{g}' = \{s \in \mathbf{s}'' \mid s \in F \text{ or } s \in \Delta(\mathbf{g}, a)\}$$
$$\mathbf{h}' = \{s \in \mathbf{s}'' \mid s \in F \text{ or } s \in \Delta(\mathbf{h}, a)\}$$
$$\mathbf{g}'' = \begin{cases} \mathbf{h}' & \text{if } \mathbf{g} = \mathbf{s} \\ \mathbf{g}' & \text{otherwise} \end{cases}$$
$$\mathbf{h}'' = \begin{cases} \mathbf{s}'' \cap F & \text{if } q'' \in F \\ \mathbf{h}' & \text{otherwise} \end{cases}$$

Duplicator wins iff the green light flashes infinitely often. Let T^{de} be the set of configurations where the green light flashes, i.e.,

$$T^{\mathrm{de}} = \{\langle \{q\}, \mathbf{s}, \mathbf{g}, \mathbf{h} \rangle \mid \mathbf{g} = \mathbf{s}\}$$

Therefore, $q \sqsubseteq^{\mathrm{de}}_{(k_1,k_2)} \mathbf{s}$ iff $p_0^I \in W^{\mathrm{de}}$, where $p_I^{\mathrm{de}} = \langle \{q\}, \mathbf{s}, \mathbf{g}, \mathbf{g} \rangle$, with $\mathbf{g} = \mathbf{s}$ if $q \notin F$ and $\mathbf{g} = \mathbf{s} \cap F$ otherwise, and

$$W^{\mathrm{de}} = \nu X \cdot \mu Y \cdot (T^{\mathrm{de}} \cap \mathsf{CPre}^{\mathrm{de}}(X)) \cup \mathsf{CPre}^{\mathrm{de}}(Y)$$

Remark 5.9.3. The algorithm for $(1, k)$-delayed simulation of [29] is more complex as it keeps track not only of the oldest and newest pending obligations, but also of all intermediate ones. This results in a queue of good pebbles which can potentially be of exponential length. We simplify this queue to one length two, by just keeping its first and last element ($\mathbf{g}$ and $\mathbf{h}$).

Correctness is established with the following observation.

Lemma 5.9.4. *Let $\pi = p_0 \overset{a_0}{\Longrightarrow} p_1 \overset{a_1}{\Longrightarrow} \cdots$ be an infinite sequence of configurations in a play of $\mathcal{G}^{\mathrm{de}}$, where, for any $i \geq 0$, p_i has the shape $p_i = \langle \mathbf{q}_i, \mathbf{s}_i, \mathbf{g}_i, \mathbf{h}_i \rangle$. Then,*

- *If Spoiler triggers an obligation at round i which is not eventually met by Duplicator, then the green light flashes only finitely many times.*

  Formally, if $q_i \in F$ and, for all $j \geq i$, $\mathbf{s}_i \not\Longrightarrow^{\exists}_F \mathbf{s}_j$, then, there exists a round $j \geq i$ (when a previous obligation is met), s.t., for any round $k \geq j$, $\mathbf{s}_k \neq \mathbf{g}_k$.

- *If Spoiler triggers only finitely many obligations, and these are met by Duplicator, then the green light does not flash only finitely many times.*

  Formally, if there exists i s.t. $q_i \in F$ and $j \geq i$ s.t. $\mathbf{s}_i \Longrightarrow^{\exists}_F \mathbf{s}_j$, and, for all $k \geq i$, $q_k \notin F$, then, for any $k \geq j$, $\mathbf{h}_k = \mathbf{g}_k = \mathbf{s}_k$.

Complexity	Type	Pebbles $(\mathcal{A}, \mathcal{B})$	Lower bound $(\mathcal{A}, \mathcal{B})$	Upper bound $(\mathcal{A}, \mathcal{B})$	Where
PSPACE	$\mathrm{di}, \mathrm{de}, \mathrm{f}$	$(1, n)$	(NBA, NBA)	(NBA, ABA)	Section 5.10.3
		$(n, 1)$	(UBA, UBA)	(ABA, UBA)	(similar)
	de, f	(n, n)	(UBA, NBA)	(ABA, ABA)	(similar)
EXPTIME	de, f	$(1, n)$	(UBA, NBA)		Section 5.10.4
		$(n, 1)$	(UBA, NBA)		(conjecture)
		$(1, \sqrt{n})$	(NBA, NBA)		Section 5.10.5
	di	$(1, n)$	(UBA, NBA)		(conjecture)
		$(n, 1)$	(UBA, NBA)		(conjecture)
		(n, n)	(UBA, NBA)		(conjecture)

Table 5.3: Complexity summary

Example: the first line means that computing $\sqsubseteq^{\mathrm{di}}_{(1,n)}$ is PSPACE-hard already for NBAs $\mathcal{A}$ and $\mathcal{B}$, and it is in PSPACE for a NBA $\mathcal{A}$ and an ABA $\mathcal{B}$.

- *If Spoiler triggers infinitely many obligations, and these are met by Duplicator, then the green light flashes infinitely often.*

Therefore, Duplicator meets all obligations iff the green light flashes infinitely often.

Finite memory strategies suffice for Duplicator Like for fair simulation, we can interpret the additional sets $\mathbf{g}$ and $\mathbf{h}$ in $\mathcal{G}^{\mathrm{de}}$ as the memory for Duplicator's strategies in the original game $\mathbb{G}^{\mathrm{de}}$. Since there are at most $2^n \cdot 2^n$ such pairs of sets, the memory needed by Duplicator in $\mathbb{G}^{\mathrm{de}}$ can be uniformly bounded by 2^{2n}.

Lemma 5.9.5. *If Duplicator has a winning strategy in the $(1, k)$-delayed simulation game $\mathbb{G}^{\mathrm{de}}$, then she has a winning strategy $\hat{\sigma}$ of memory bounded by 2^{2n}.*

5.10 Complexity of multipebble simulations

In this section, we study the theoretical complexity of checking multipebble simulations for NBAs and ABAs. We consider direct, delayed and fair simulation, as well as different pebble configurations. No general complexity result for checking multipebble simulations was previously known, even on NBAs.

In Section 5.9, we have presented algorithms for solving multipebble simulation games. In particular, we have shown that it suffices to solve a finite game of exponential size with a simple ω-regular winning condition, from which trivial EXPTIME upper bound follows. Here, we study when such a bound is tight, or a better PSPACE-completeness characterization can be given.

In general, a PSPACE lower bound holds when Duplicator is allowed to use sufficiently many pebbles, either on $\mathcal{A}$, on $\mathcal{B}$, or on both. (When the number of pebbles is fixed, the problem is in PTIME, cf. Section 5.9.) This bound is tight, i.e., the problem is PSPACE-complete, when Duplicator can play without trade-offs during the game. In this case, alternation is removed and we are left with a 1-player game of exponential size, which can be solved in PSPACE.

Otherwise, if Duplicator uses many pebbles *and* she still has to make inherent choices during the game, then we have to solve a 2-player game of exponential size. This suffices to encode EXPTIME computations, and we get an EXPTIME lower bound.

In the following, we give more details about the reductions. Our results are summarized in Table 5.3. The cases left open can be presumably solved with reductions similar to the ones given here.

PSPACE-completeness For lower bounds, we use reductions from certain kinds of tiling problems. Domino-tiling games provide a natural computational model encapsulating the essential content of several complexity classes. In this section, we use models characterizing PSPACE and EXPTIME. See Section 5.10.2 for an introduction to domino-tiling games.

A PSPACE-lower bound applies to all types of maximal multipebble simulations we consider. This happens since, whenever n^c pebbles are used on either side, for some $c > 0$, the combinatorial structure of the problem allows us to encode 1-player tiling problems. We explain this technique for the special case of $(1, n)$-simulations in Section 5.10.3. The other cases can be dealt with in a similar way.

The PSPACE lower bound is tight when we can show that Duplicator can play without any tradeoff during the simulation game. With this we mean that there exists a uniform strategy σ^* s.t., if Duplicator can win the game, then σ^* itself is winning. Therefore, in a simulation game without tradeoffs, Duplicator will always play according to a fixed strategy σ^*, and σ^* is a winning strategy exactly when there exists a winning strategy. Since Duplicator plays according to a strategy σ^* which has been fixed in advance, we are in fact eliminating Duplicator from the game, and we are left with a

1-player game. Additionally, if σ^* can be assumed to be of exponential memory, then the size of the game remains exponential even after Duplicator has been eliminated.

Games admitting uniform strategies for Duplicator are in general all $(1, n)$ and $(n, 1)$-simulations where Duplicator never controls the single pebble on $\mathcal{A}$, or on $\mathcal{B}$ respectively. This happens precisely when $\mathcal{A}$ is an NBA, and $\mathcal{B}$ is a UBA, respectively. For the special case of delayed and fair simulation, even (n, n)-simulation admits uniform strategies, since Duplicator uses left-blind strategies (cf. Definition 5.4.14). This implies PSPACE-completeness in all the cases mentioned. See the first half of Table 5.3. In Section 5.10.1 we illustrate the details for $(1, n)$-simulations.

EXPTIME-completeness In those cases where Duplicator has many pebbles, but she also incurs in tradeoffs during the simulation game, an EXPTIME lower bound applies. In general, this holds in the following cases:

1. We are in a $(1, n)$ or in a $(n, 1)$ simulation game, but Duplicator can sometimes control the single pebble on $\mathcal{A}$ or on $\mathcal{B}$, respectively. This happens precisely when $\mathcal{A}$ is a UBA, or $\mathcal{B}$ is an NBA, respectively. We deal with the $(1, n)$ case in Section 5.10.4.

   We conjecture that an EXPTIME lower bound for the $(n, 1)$ case can be shown with similar techniques.

2. We are in a $(1, \sqrt{n})$ simulation game where Duplicator controls a high, but non-maximal number of pebbles. Therefore, there is a combinatorial blow-up *and* there are tradeoffs. See Section 5.10.5.

3. (Conjecture) We are in a (n, n)-direct simulation game. Here, due to the acceptance condition of direct simulation, Duplicator has tradeoffs due to the non-monotonicity of the acceptance condition w.r.t. splitting pebbles.

The first two cases are dealt with for delayed and fair simulation. We leave the direct simulation case, and the last case, as a conjecture. See the second half of Table 5.3.

5.10.1 PSPACE-membership

In this section, we give PSPACE algorithms for the following simulations:

- $(1, n)$-direct, delayed and fair simulation, where $\mathcal{A}$ is an NBA.
- $(n, 1)$-direct, delayed and fair simulation, where $\mathcal{B}$ is a UBA.

- (n, n)-delayed and fair simulation, without any restriction on $\mathcal{A}$ or $\mathcal{B}$.

The idea is to show that Duplicator has uniform winning strategies of finite exponential memory. We call such strategies *oblivious*.

Definition 5.10.1 (Oblivious strategies)**.** *A strategy σ^* for Duplicator is* oblivious w.r.t. a game $\mathcal{G}$ *iff it has at most exponential memory and,*

If there exists a winning strategy σ in $\mathcal{G}$, then σ^* is itself winning.

Games admitting uniform oblivious strategies are called *simple*.

Definition 5.10.2 (Simple simulation games)**.** *A simulation game $\mathcal{G}$ is* simple *iff there exists a strategy σ^* for Duplicator which is oblivious w.r.t. $\mathcal{G}$.*

Therefore, in a simple game Duplicator can be assumed to *always* play according to a fixed oblivious strategy σ^*. So, we can plug-in σ^* into the parity games of Section 5.9, and we get 1-player parity games where only Spoiler plays. Since σ^* has exponential memory, we get a 1-player game of exponential size. The details are standard.

Deciding whether Spoiler wins in such solitary games can be reduced to the emptiness problem for nondeterministic parity automata. The last problem can be solved in NLOGSPACE: Indeed, [119] shows that the emptiness problem for NBAs is in NLOGSPACE, and there exist translations from parity to polynomially larger Büchi automata [75].

Therefore, by applying a logarithmic algorithm to an exponential automaton, we get a NPSPACE procedure. By Savitch's theorem [105], we can compute simple simulations in PSPACE. In the rest of this section, we show that certain simulation games are simple by constructing uniform oblivious strategies σ^* for Duplicator.

Direct simulation $(1, n)$ and $(n, 1)$-direct simulation games are simple if Duplicator does not control the left (right, resp.) pebble, that is, if $\mathcal{A}$ is an NBA ($\mathcal{B}$ is a UBA, resp.). Indeed, in such games Duplicator has a particularly elementary oblivious strategy σ^*:

- Always split pebbles maximally.
- For $(1, n)$: When the left pebble is accepting, throw away non-accepting right pebbles.
- For $(n, 1)$: When the right pebble is not accepting, throw away accepting left pebbles.

Remark 5.10.3. On the other side, in (n, n)-direct simulation games oblivious strategies do not exist anymore, since there might be trade-offs for Duplicator between discarding right or left pebbles.

Formally, for the $(1, n)$ case, let $\mathcal{A}$ be an NBA and $\mathcal{B}$ an ABA. Assume that the current configuration is $\langle q, \mathbf{s}\rangle$, and that Spoiler selects input symbol a, and successors $q' \in \Delta(q, a)$ and $\mathbf{s}' \in \bigotimes \Delta(\mathbf{s}^U, a)$. Then, σ^* is defined as to select $\mathbf{s}'' = \mathbf{s}' \cup \Delta(\mathbf{s}^E, a)$ if $q' \notin F$, and $\mathbf{s}'' = (\mathbf{s}' \cup \Delta(\mathbf{s}^E, a)) \cap F$ if $q' \in F$. The next configuration is $\langle q', \mathbf{s}''\rangle$. For the $(n, 1)$ case, let $\mathcal{B}$ be a UBA and $\mathcal{A}$ an ABA. Assume that the current configuration is $\langle \mathbf{q}, s\rangle$, and that Spoiler selects input symbol a, and successors $s' \in \Delta(s, a)$ and $\mathbf{q}' \in \bigotimes \Delta(\mathbf{q}^E, a)$. Then, σ^* is defined as to select $\mathbf{q}'' = \mathbf{q}' \cup \Delta(\mathbf{q}^U, a)$ if $s' \in F$, and $\mathbf{q}'' = (\mathbf{q}' \cup \Delta(\mathbf{q}^U, a)) \setminus F$ if $s' \notin F$. The next configuration is $\langle \mathbf{q}'', s'\rangle$.

Clearly, σ^* is a memoryless strategy. To show that it is oblivious, notice that $\sigma^{\mathrm{di}} \subseteq \sigma^*$ for any winning strategy σ^{di}. Intuitively, this holds since σ^* throws pebbles only when strictly necessary. Indeed, for the $(1, n)$-case, as long as there are no accepting left pebbles, σ^* splits pebbles maximally, and obviously subsumes σ^{di} during these rounds. Moreover, if there is any accepting left pebble, then any pebble that σ^* throws away has to be thrown away by σ^{di} as well, since the latter strategy is winning. Therefore, σ^* subsumes σ^{di} also in these rounds. The $(n, 1)$-case is analogous. Therefore, if there exists a winning strategy σ^{di}, then σ^* is itself winning, since it always satisfies the acceptance condition by construction, and it never gets stuck.

Fair simulation Recall that Duplicator has left-blind strategies in (n, k)-delayed and fair simulation (cf. Definition 5.4.14). Left-blind strategies are clearly oblivious for $(n, 1)$-delayed and fair simulation. Moreover, oblivious strategies for $(1, n)$-simulations induce oblivious strategies for (n, n)-simulations, by splitting pebbles maximally on the left. Therefore, we only address the $(1, n)$ case, for which we describe an oblivious strategy σ^* for Duplicator.

σ^* works in phases. During each phase, it splits pebbles maximally, while recording which pebbles have seen an accepting state since the beginning of the phase.

If the same configuration $\langle q, \mathbf{s}'\rangle$ with $q \in F$ appears $f(n)$ times, then drop pebbles which have not yet seen an accepting state, and the next phase starts.

$f(n)$ will be chosen large enough s.t. σ^* throws pebbles away later than any strategy of memory at most 2^{2n} (this suffices by Lemma 5.9.2). Clearly, recording of good pebbles can be achieved with at most 2^n memory, and detecting repetitions can be done with

at most $n \cdot 2^n \cdot f(n)$ memory. Below, we show that $f(n) = 2^{4n}$ in fact suffices, thus, overall $2^{O(n)}$ memory suffices for σ^*.

Assume that Duplicator has a winning strategy, and that Spoiler plays a fair path $\pi_0 = q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \cdots$. By Lemma 5.9.2, there exists a winning strategy $\hat{\sigma}$ of memory at most 2^{2n}. We show $\hat{\sigma} \subseteq \sigma^*$ for any such strategy. Let $\pi_1 = \mathbf{s}_0 \overset{a_0}{\Longrightarrow} \mathbf{s}_1 \overset{a_1}{\Longrightarrow} \cdots$ and $\pi'_1 = \mathbf{s}'_0 \overset{a_0}{\Longrightarrow} \mathbf{s}'_1 \overset{a_1}{\Longrightarrow} \cdots$ be two multipaths s.t. $\pi = \pi_0 \times \pi_1$ is $\hat{\sigma}$-conform and $\pi' = \pi_0 \times \pi'_1$ is σ^*-conform. Moreover, at round i, let m_i be the current state of the memory for $\hat{\sigma}$. We reason by induction on phases. Suppose the current phase starts at round i, and, by induction hypothesis, $\mathbf{s}_i \subseteq \mathbf{s}'_i$. During the entire phase, σ^* splits pebbles maximally, so the order is respected.

At the end of the phase, say at round k, the same configuration $\langle q, \mathbf{s}'_k \rangle$ with $q \in F$ has appeared $f(n)$ times. We determine $f(n)$ as to ensure that strategy $\hat{\sigma}$ has dropped bad pebbles, i.e., $\mathbf{s}_i \Longrightarrow_F^{\exists} \mathbf{s}_k$. For any $j \geq i$, let $\mathbf{b}_j = \{s \in \mathbf{s}_j \mid \mathbf{s}_i \not\Longrightarrow_F^{\exists} \{s\}\}$ be the set of pebbles which are not good at round j. Notice that $\mathbf{s}_i \Longrightarrow_F^{\exists} \mathbf{s}_k$ holds iff $\mathbf{b}_k = \emptyset$. We choose $f(n)$ large enough to ensure that, for some h s.t. $i \leq h \leq k$, simultaneously,

1. $\mathbf{s}_h$ is repeated twice (2^n),
2. $\mathbf{b}_h$ is repeated twice (2^n), and
3. m_h is repeated twice (2^{2n}).

Therefore, let $f(n) = 2^n \cdot 2^n \cdot 2^{2n} = 2^{4n}$. By construction, there exist h_0 and h_1 s.t. $i \leq h_0 < h_1 \leq k$ and $q = q_{h_0} = q_{h_1}$, $\mathbf{s}'_k = \mathbf{s}'_{h_0} = \mathbf{s}'_{h_1}$, $\mathbf{s}_{h_0} = \mathbf{s}_{h_1}$, $\mathbf{b}_{h_0} = \mathbf{b}_{h_1}$, and $m_{h_0} = m_{h_1}$. By contradiction, assume $\mathbf{s}_i \not\Longrightarrow_F^{\exists} \mathbf{s}_k$. Then, $\mathbf{s}_i \not\Longrightarrow_F^{\exists} \mathbf{s}_{h_0}$ by Lemma 5.4.1, and, by definition, $\mathbf{b}_{h_0} \neq \emptyset$. Let $w = a_{h_0} a_{h_0+1} \cdots a_{h_1-1}$ and $\rho = \langle q_{h_0}, \mathbf{s}_{h_0} \rangle \overset{a_{h_0}}{\Longrightarrow} \cdots \overset{a_{h_1-1}}{\Longrightarrow} \langle q_{h_1}, \mathbf{s}_{h_1} \rangle$. Spoiler forces the following play (since $\hat{\sigma}$ depends uniquely on the current configuration and memory state):

$$\pi = \langle q_0, \mathbf{s}_0 \rangle \overset{a_0}{\Longrightarrow} \cdots \overset{a_{h_0-1}}{\Longrightarrow} \rho^\omega$$

for which $q_i \in F$ for infinitely many i's. Let $h_2 = h_0 + 2 \cdot (h_1 - h_0)$, $h_3 = h_0 + 3 \cdot (h_1 - h_0)$, and so on. For any $l \geq 0$ and subset $\mathbf{b} \subseteq \mathbf{s}_{h_{l+1}}$, let

$$g(\mathbf{b}) = \bigcup_{s' \in \mathbf{b}} \{s \in \mathbf{s}_{h_l} \mid s \xrightarrow{w} s'\}$$

be the set of w-predecessors of $\mathbf{b}$ in $\mathbf{s}_{h_l}$. g does not depend on l since multipaths are obtained by unrolling the fixed multipath ρ. Bad pebbles can only have bad predecessors,

and, by definition, $\mathbf{b}_{h_{l+1}}$ is the maximal set of bad pebbles. I.e., for any $l \geq 0$,

$$\mathbf{b}_{h_{l+1}} = \nu \mathbf{b} \cdot \mathbf{b} \subseteq \mathbf{s}_{h_{l+1}} \wedge g(\mathbf{b}) \subseteq \mathbf{b}_{h_l} \tag{†}$$

We claim that $\mathbf{b}_{h_0} = \mathbf{b}_{h_1} = \mathbf{b}_{h_2} = \cdots$. We proceed by induction. The base case $\mathbf{b}_{h_0} = \mathbf{b}_{h_1}$ holds by assumption. Assume $\mathbf{b}_{h_{t-1}} = \mathbf{b}_{h_t}$. By (†), $\mathbf{b}_{h_{t+1}} = \nu \mathbf{b} \cdot \mathbf{b} \subseteq \mathbf{s}_{h_{t+1}} \wedge g(\mathbf{b}) \subseteq \mathbf{b}_{h_t}$. By inductive hypothesis, $\mathbf{b}_{h_t} = \mathbf{b}_{h_{t-1}}$ and, since $\mathbf{s}_{h_{t+1}} = \mathbf{s}_{h_t}$, $\mathbf{b}_{h_{t+1}} = \nu \mathbf{b} \cdot \mathbf{b} \subseteq \mathbf{s}_{h_t} \wedge g(\mathbf{b}) \subseteq \mathbf{b}_{h_{t-1}}$, which is (†) for $l = t - 1$. Thus, $\mathbf{b}_{h_{t+1}} = \mathbf{b}_{h_t}$.

Therefore, $\emptyset \neq \mathbf{b}_{h_0} = \mathbf{b}_{h_1} = \mathbf{b}_{h_2} = \cdots$, that is, $\mathbf{s}_i \not\Longrightarrow_F^{\exists} \mathbf{s}_{h_l}$ for any $l \geq 0$. Thus, $\pi_1 = \mathbf{s}_0 \xRightarrow{a_0} \mathbf{s}_1 \xRightarrow{a_1} \cdots$ is not an existentially fair multipath. Since Spoiler builds a fair path $\pi_0 = q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \cdots$, this contradicts that $\hat{\sigma}$ is winning. Hence, $\mathbf{s}_i \Longrightarrow_F^{\exists} \mathbf{s}_k$.

Since $\mathbf{s}_i \subseteq \mathbf{s}'_i$ by induction hypothesis, by Lemma 5.4.3, $\mathbf{s}'_i \Longrightarrow_F^{\exists} \mathbf{s}_k$. Therefore, good pebbles in $\mathbf{s}_k$ are good in $\mathbf{s}'_k$ as well. Let $\mathbf{s}''_k$ be the largest subset of $\mathbf{s}'_k$ s.t. $\mathbf{s}'_i \Longrightarrow_F^{\exists} \mathbf{s}''_k$. Then, $\mathbf{s}_k \subseteq \mathbf{s}''_k$ and $\mathbf{s}''_k$ is non-empty. Consequently, Duplicator discards all pebbles not in $\mathbf{s}''_k$, the green light flashes, and the invariant is preserved for the beginning of the next phase.

The invariant and the definition of σ^* guarantee that, if there exists a winning strategy, then σ^* is itself winning:

1. σ^* never gets stuck in an empty set since $\hat{\sigma} \subseteq \sigma^*$, and $\hat{\sigma}$ is winning.

2. Since π_0 is fair, configurations of the form $\langle q_i, \mathbf{s}_i \rangle$ with $q_i \in F$ appear arbitrarily many times. By the definition of σ^*, pebbles which have not seen an accepting state are always eventually dropped.

Therefore, π'_1 is existentially fair. Thus, σ^* is oblivious.

Delayed simulation σ^* works in phases as in fair simulation: During each phase, it splits pebbles maximally, while recording which pebbles have seen an accepting state since the oldest (*principal*) and the most recent (*fresh*) pending obligation (cf. Section 5.9.3).

If a configuration $\langle q, \mathbf{s}' \rangle$ appears $f(n)$ times <u>and</u> there is a principal pending obligation, then drop bad pebbles w.r.t. the principal obligation, initialize the next principal obligation as the current fresh obligation, and the next phase starts.

$f(n)$ can be taken to be 2^{4n} for the same reasons as for fair simulation. In particular, the factor 2^{2n} is used to force a repetition of the memory state of any 2^{2n}-finite memory

strategy, which suffices by Lemma 5.9.5. Strategy σ^* needs 2^{2n} memory for recording good pebbles w.r.t. the principal and fresh obligation, and $n \cdot 2^n \cdot f(n)$ for detecting repetitions. Therefore, $2^{O(n)}$ memory suffices also for delayed simulation.

The correctness argument is as for fair simulation, by showing $\hat{\sigma} \subseteq \sigma^*$ for any 2^{2n}-memory winning strategy $\hat{\sigma}$ (if any). The crucial observation is that, if there is a pending obligation, then $\hat{\sigma}$ has to drop pebbles after at most 2^{4n} rounds, otherwise Spoiler can force an eventually periodic play where the obligation is never fulfilled. By appropriately dropping pebbles, σ^* is guaranteed to fulfill any pending obligation, and, because this happens after 2^{4n} rounds since the obligation is raised, σ^* is ensured to have at least as many pebbles as $\hat{\sigma}$.

5.10.2 Domino-tiling games

We introduce certain tiling games which are used to prove the PSPACE and EXPTIME lower bounds in the following sections. We consider *Wang tiles* [120, 24]. Let $T = \{t_0, t_1, \ldots, t_{k-1}\}$ be a finite set of k different tiles. There are two compatibility relations $H, V \subseteq T \times T$, where H is the *horizontal* compatibility relation and V is the *vertical* compatibility relation. A row is a sequence of tiles. Fix an even number $m + 1$, and let $R = T^{m+1}$ be the set of all rows of length $m + 1$.

A *tiling* is a sequence of rows $r_0 r_1 \cdots \in R^\omega$, where each row r_i consists of the $m + 1$ tiles $r_i = c_{i,0} c_{i,1} \cdots c_{i,m}$. A tiling is a *valid* iff the compatibility relations H, V are satisfied: That is, for any $i \geq 0$ and $0 \leq j \leq m$, the following holds

- *Horizontal compatibility:* $H(c_{i,j}, c_{i,j+1})$ if $j < m$, and $H(c_{i,j}, c_{i+1,0})$ if $j = m$.
- *Vertical compatibility:* $V(c_{i,j}, c_{i+1,j})$.

Remark 5.10.4. The horizontal compatibility says that, if we arrange rows horizontally, then any two neighboring tiles satisfy H. This differs from [24], where border tiles are treated separately, and assumed to match a special-purpose horizontal compatibility relation H'. Our is a minor modification, and does not alter the complexity results.

A *domino-tiling game* starting at a distinguished row $r_0 \in R$ is played by two players, named Saboteur and Constructor, which alternate in rounds by choosing tiles. The first row is fixed, and equals the given r_0. Successive rows are built with tiles placed in a left-to-right fashion, with Constructor starting first. Once a row is completed, the game moves to the following one. The game stops only if no compatible tile can be

placed: In this case, Saboteur wins. Otherwise, the game never stops, and an infinite, rectangular and valid tiling is produced, and Constructor wins.

Given a finite set of tiles T, an even number $m + 1$ in unary, and an initial row $r_0 \in R$, determining whether Constructor has a winning strategy in the domino-tiling game starting at r_0 is EXPTIME-complete.

Theorem 5.10.5 (cf. [24]). *Solving 2-player tiling games is EXPTIME-complete.*

If we restrict the game to only one player, i.e., to just Constructor, then the problem reduces to PSPACE-complete.

Theorem 5.10.6 (cf. [24]). *Solving 1-player tiling games is PSPACE-complete.*

5.10.3 PSPACE-hardness of $(1, n)$-simulations

Direct simulation PSPACE-hardness of $(1, n)$-direct simulation on NBAs is immediate by reduction from trace inclusion: Given two nondeterministic finite state transition systems $\mathcal{A}$ and $\mathcal{B}$ (with no acceptance condition), one can decide whether the set of traces of $\mathcal{A}$ is included in that of $\mathcal{B}$ by interpreting $\mathcal{A}$ and $\mathcal{B}$ as automata with every state accepting, and then checking $\mathcal{A} \sqsubseteq^{\mathrm{di}}_{(1,n)} \mathcal{B}$, where $n = |\mathcal{B}|$. Since trace inclusion is PSPACE-hard [89], $\sqsubseteq^{\mathrm{di}}_{(1,n)}$ is PSPACE-hard.

Delayed and fair simulation We show PSPACE-hardness of $(1, n)$-delayed and fair multipebble simulation on NBAs by reducing from the one-player domino-tiling problem. This reduction will be extended to the two-player version in order to show EXPTIME-hardness of various multipebble simulation games in the following Sections 5.10.4 and 5.10.5.

The main idea is as follows. In the tiling game there is only one player, i.e., Constructor. Spoiler in the simulation game plays the role of Constructor in the tiling game. Duplicator ensures that only compatible tiles are put on the board, thus allowing Spoiler's play to induce a legal Constructor's play. When a new tile is added in the current row, Duplicator checks that the vertical and horizontal constraints are both satisfied.

- Horizontal constraints are easy to check by just comparing the previous tile with the new one.
- Vertical constraints need to remember the previous row, and Duplicator uses pebbles to achieve this.

If any constraint is violated, then Duplicator is allowed to go to a special accepting sink state capable of doing any action in Σ. The game always goes on forever, and Spoiler wins iff Duplicator has not reached the sink state (since Duplicator is accepting in the sink state). This implies that the simulation and the tiling game are equivalent, in the sense that Spoiler wins if, and only if, Constructor wins by building an infinite and valid tiling. Since one-player domino games are PSPACE-hard by Theorem 5.10.6, $(1, n)$-simulation is PSPACE-hard.

The reduction is the same for both delayed and fair simulation. Let the set of k tiles be $T = \{t_0, t_1, \ldots, t_{k-1}\}$, let $H, V \subseteq T \times T$ be the horizontal and vertical compatibility relations, respectively, and let $r_0 = c_0 c_1 \ldots c_m$ be the initial row of $m + 1$ tiles. Let the alphabet $\Sigma = T$ be just the set of tiles. The automaton $\mathcal{A}$ consists of a single universal accepting state $Q_{\mathcal{A}} = \{C\}$:

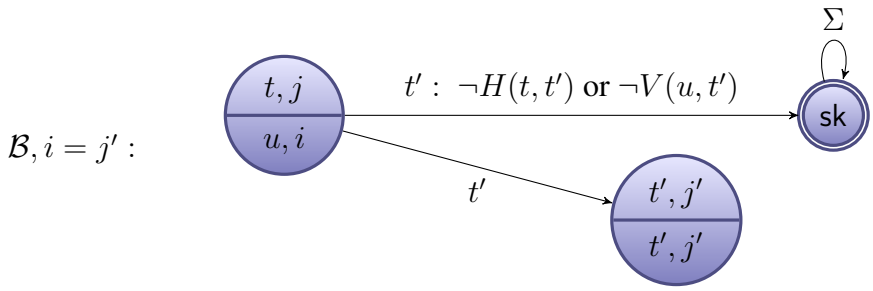
$\mathcal{A}$:

Therefore, Spoiler will always build a fair path in $\mathcal{A}$, and delayed simulation collapses to fair simulation in this case.

The automaton $\mathcal{B}$ has states in $Q_{\mathcal{B}} = (T \times [m+1]) \times (T \times [m+1]) \cup \{\mathsf{sk}\}$ consisting of pairs of tiles augmented with a position index, and a special sink state sk. Therefore, $n = ((m + 1) \cdot k)^2 + 1$. Intuitively, a pebble is on a non-sink state $\frac{t, j}{u, i} \in Q_{\mathcal{B}}$ when the last tile that has been laid is t and it is in the j-th tile of the current row. Moreover, if $i \leq j$, then the i-th tile of the current row is u; in particular, if $i = j$, then $u = t$. Otherwise, if $i > j$, then u is the i-th tile of the *past* row. Given a state as above, we say that $\langle t, j\rangle$ is the *current component* and that $\langle u, i\rangle$ is the *past component*.

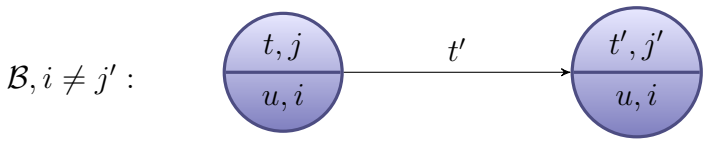
Fix a source state $\frac{t, j}{u, i}$, and let $j' = (j + 1) \mod (m + 1)$ be the successor index. On input symbol $t' \in T$, there are two types of transitions in $\mathcal{B}$.

(Trans-0) Case $i = j'$: The newly placed tile t' is checked against horizontal compatibility with t, and against vertical compatibility with u. If the check fails, then there is an immediate transition to the universal state sk. Otherwise, the current component is updated to record that the new tile t' is placed in position j'. The past component records this information as well, for future checks.



State sk is the unique accepting state in $\mathcal{B}$.

(Trans-1) Case $i \neq j'$: The current component is updated to just record that the new tile t' is placed in position j'. The past component stays unchanged:



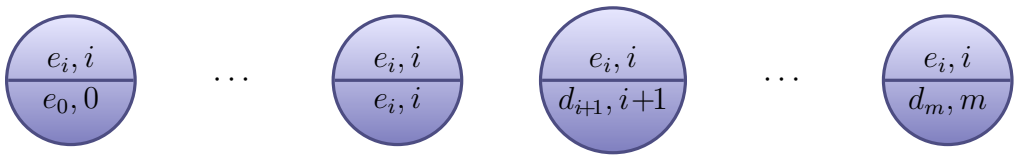
No check is made for horizontal or vertical compatibility in this case.

A formal definition of automaton $\mathcal{B}$ follows.

Definition 5.10.7. $\mathcal{B} = (Q_{\mathcal{B}}, \Sigma, \Delta_{\mathcal{B}}, F_{\mathcal{B}})$, *where*

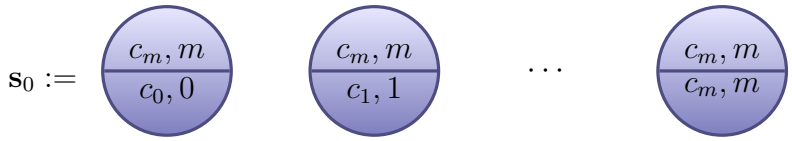
$$
\begin{aligned}
Q_{\mathcal{B}} &= (T \times [m+1]) \times (T \times [m+1]) \cup \{\mathsf{sk}\} \\
\Sigma &= T = \{t_0, t_1, \ldots, t_{k-1}\} \\
F_{\mathcal{B}} &= \{\mathsf{sk}\} \\
\Delta_{\mathcal{B}} &= \{(\begin{pmatrix} \langle t, j\rangle \\ \langle u, i\rangle \end{pmatrix}, t', \begin{pmatrix} \langle t', j'\rangle \\ \langle t', j'\rangle \end{pmatrix}) \mid t' \in T \textit{ and } i = j' = (j+1) \mod (m+1)\} \cup \\
&\quad \{(\begin{pmatrix} \langle t, j\rangle \\ \langle u, i\rangle \end{pmatrix}, t', \mathsf{sk}) \mid t' \in T, i = (j+1) \mod (m+1) \textit{ and:} \\
&\qquad\qquad\qquad\qquad\qquad\qquad \neg H(t, t') \textit{ or } \neg V(u, t')\} \cup \\
&\quad \{(\begin{pmatrix} \langle t, j\rangle \\ \langle u, i\rangle \end{pmatrix}, t', \begin{pmatrix} \langle u, i\rangle \\ \langle t', j'\rangle \end{pmatrix}) \mid t' \in T \textit{ and } i \neq j' = (j+1) \mod (m+1)\} \cup \\
&\quad \{(\mathsf{sk}, a, \mathsf{sk}) \mid a \in \Sigma\}
\end{aligned}
$$

Invariant The only way for Duplicator to win is to eventually reach the sink state sk, by taking some t'-transition to sk, where t' is an incorrectly placed tile (as in (Trans-0) above). As long as this does not happen, Constructor/Spoiler has only placed valid tiles, and the following invariant is preserved: At any round, pebbles in $\mathcal{B}$ are on states



if, and only if, the previous row contains tiles $r = d_0 d_1 \cdots d_m$, and the current row $r' = e_0 e_1 \cdots e_i$ has been filled up to position $i \leq m$. In particular, there are exactly $m + 1$ $\mathcal{B}$-pebbles on the board, all of which agree on the current component $\langle e_i, i \rangle$, i.e., the last laid pebble. Duplicator does not benefit from having more than $m + 1$ pebbles.

Given the initial row $r_0 = c_0 c_1 \cdots c_m$, pebbles in $\mathcal{B}$ are initially placed on states



From the discussion above, Constructor wins the domino-tiling problem iff Spoiler wins the delayed/fair simulation game starting from position $\langle C, \mathbf{s}_0 \rangle$.

Theorem 5.10.8. *Checking $(1, n)$-direct, delayed and fair simulation is PSPACE-hard.*

5.10.4 EXPTIME-hardness of $(1, n)$-simulations on ABAs

We show EXPTIME-hardness of $(1, n)$-delayed and fair simulation on ABAs by reduction from two-player domino-tiling games. We extend the reduction from one-player domino-tiling games of Section 5.10.3. The actions of Saboteur are modelled by giving extra choices to Duplicator. In this section we do so by introducing alternation in the $\mathcal{A}$ automaton, and we use a gadget that allows Duplicator/Saboteur to select the next input symbol/tile. The $\mathcal{B}$ automaton is the same as in the previous section, with some adjustment to enforce that Duplicator faithfully mimics Saboteur. Automaton $\mathcal{B}$ is still nondeterministic, while $\mathcal{A}$ has only universal and deterministic states. (In the next section, the actions of Saboteur will be modelled differently: Instead of introducing alternation in $\mathcal{A}$, we will prescribe a fixed but high number of $\mathcal{B}$-pebbles for Duplicator.)

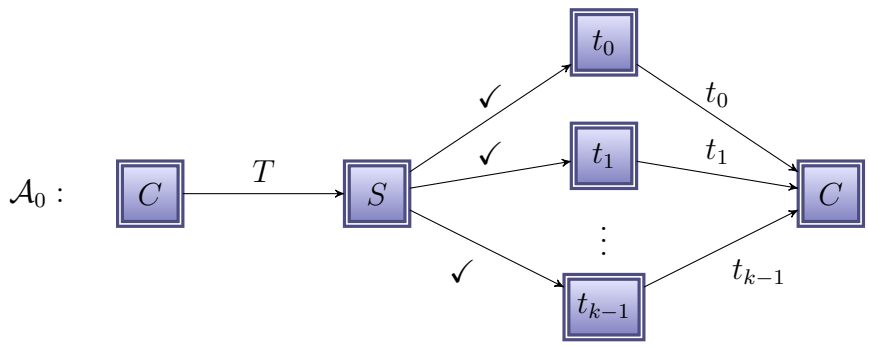
Recall that $T = \{t_0, t_1, \ldots, t_{k-1}\}$ is the set of k tiles, $H, V \subseteq T \times T$ are the horizontal and vertical compatibility relations, respectively, and $r_0 = c_0 c_1 \ldots c_m$ is the initial row. We describe the construction in an incremental fashion; the starting point is the the one of Section 5.10.3.

In general, when modelling the actions of Saboteur by actions of Duplicator, we need to address the following two abstract issues.

(ModIssue-0) Duplicator can select only valid tiles.

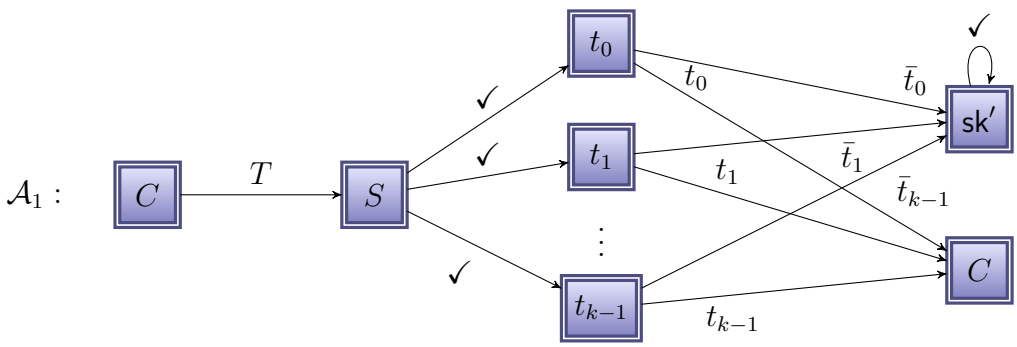
(ModIssue-1) If Duplicator cannot select any compatible tile, then Duplicator/Saboteur wins the respective game.

Consider the following modelling attempt (state C is repeated twice for readability).



State C is deterministic and models the choices of Spoiler/Constructor (as in Section 5.10.3), which is in charge of selecting the next input tile $t_i \in T$. Choices of Saboteur are modelled in a two-step process. State S is a proper universal state, which can only perform the special action ✓ and go to some state $t_i \in T$. Therefore, Spoiler can only choose symbol ✓, and Duplicator/Saboteur has to select a successor state t_i. Then, from state t_i, there is a characteristic transition $t_i \xrightarrow{t_i} C$ back to state C. Thus, if Duplicator went to state t_i, then Spoiler is forced to play action t_i, ending the two-step process. (For compatibility with this two-step process, we allow every state in $\mathcal{B}$ to perform action ✓—see later.)

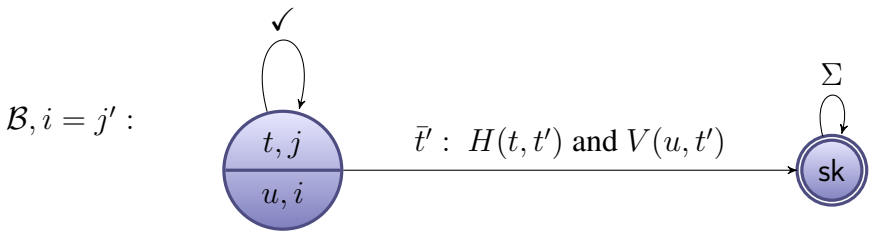
We address (ModIssue-0). Nothing prevents Duplicator from selecting an incompatible tile and winning immediately by going to state sk in $\mathcal{B}$ (cf. Section 5.10.3). To avoid this, we endow Spoiler with the capability of "punishing" Duplicator whenever the latter tries to place an incompatible tile. We associate to each action $t_i \in T$ a dual action $\bar{t}_i \in \overline{T}$, and we add in $\mathcal{A}$ transitions of the form $t_i \xrightarrow{\bar{t}_i} \mathsf{sk}'$, where sk' is a distinguished sink state in $\mathcal{A}$:



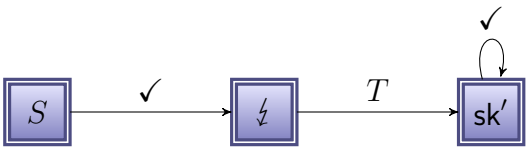
Intuitively, if Duplicator plays an incompatible tile t_i by going to state t_i, then Spoiler

can punish her by playing $\bar{t}_i$ and going to sk'. From sk', Spoiler would win by playing $\checkmark$ forever, since we make sure that no non-sink accepting $\checkmark$-loop exists in $\mathcal{B}$.

So far, nothing prevents Spoiler from always playing some barred action $\bar{t}_i$ and trivially winning. In order to prevent this, we need to modify $\mathcal{B}$ to ensure that, if t_i actually *was* a compatible tile, then Spoiler would lose if playing $\bar{t}_i$. We achieve this by adding in $\mathcal{B}$ the following additional transitions to the sink state (Case (Trans-0), where $j' := (j+1) \mod (m+1)$):

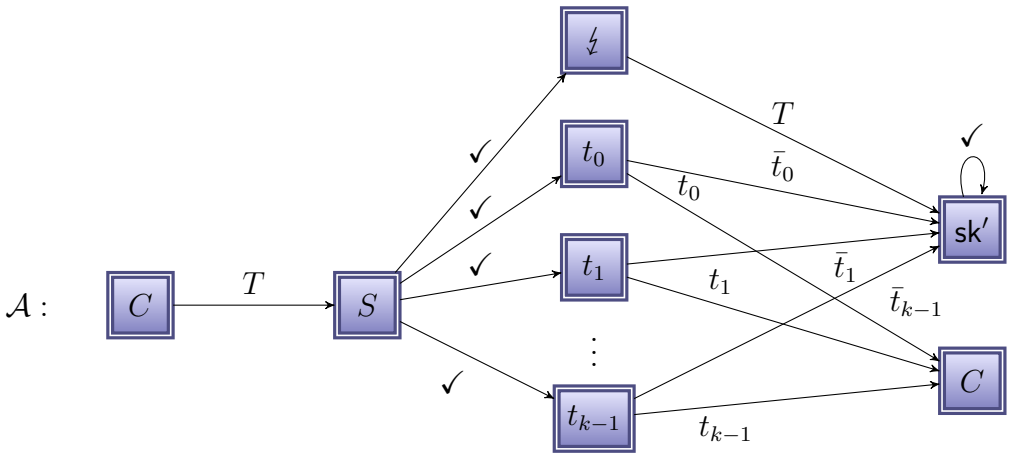


It remains to address (ModIssue-1): We need a way for Duplicator/Saboteur to declare that no compatible tile can be placed next. We create a new $\mathcal{A}$-state $\lightning$, together with the following transitions:

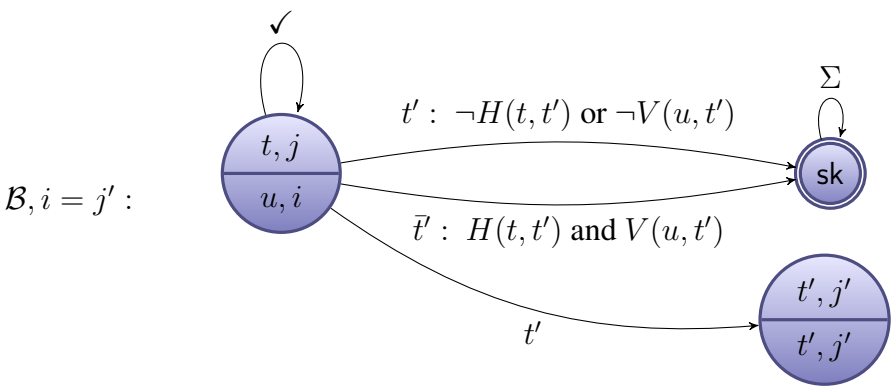


Intuitively, if Duplicator goes to state $\lightning$, then she claims that, for every tile placed next, she can prove that this tile is incompatible. From state $\lightning$, there are only t_i-transitions to the sink state sk', for every tile $t_i \in T$ (and not barred $\bar{t}_i$-transitions). Therefore, if no valid tile can be placed next, then Duplicator will go to state $\lightning$, from which Spoiler is forced to play an incompatible tile $t_i \in T$ and go to state sk'. Since t_i is incompatible, there exists some pebble on $\mathcal{B}$ that goes to the accepting state sk. From configuration $\langle \mathsf{sk}', \mathsf{sk} \rangle$, Duplicator wins. On the other hand, if Duplicator erroneously believes that there is no compatible tile that can be placed next, and she moves to $\lightning$ nonetheless, then Spoiler could play some compatible tile $t_i \in T$ and go to state sk'. Since t_i *was* compatible, no pebble in $\mathcal{B}$ can go to state sk. Therefore, Spoiler would play $\checkmark^\omega$ and win, since all $\mathcal{B}$-pebbles would be trapped into a non-accepting $\checkmark$-loop. Summing-up, Duplicator goes to state $\lightning$ iff there is no compatible tile to be placed next.

Overall, automaton $\mathcal{A}$ is as follows:



Automaton $\mathcal{B}$ has the same states as in Section 5.10.3. Transitions of kind (Trans-1) are also the same. On the other hand, transitions of kind (Trans-0) are as follows:



A formal definition of the two automata follows.

Definition 5.10.9. *Let* $\Sigma = T \cup \overline{T} \cup \{\checkmark\}$. *Then,* $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma, \Delta_{\mathcal{A}}, F_{\mathcal{A}})$, *where*

$$
\begin{aligned}
Q_{\mathcal{A}} &= \{C, S, \lightning, \mathsf{sk}'\} \cup T \\
F_{\mathcal{A}} &= Q_{\mathcal{A}} \\
\Delta_{\mathcal{A}} &= \{(C, t, S) \mid t \in T\} \cup \\
&\quad \{(S, \checkmark, x) \mid x \in T \cup \{\lightning\}\} \cup \\
&\quad \{(\lightning, t, \mathsf{sk}') \mid t \in T\} \cup \\
&\quad \{(t, t, C), (t, \bar{t}, \mathsf{sk}') \mid t \in T\} \cup \\
&\quad \{(\mathsf{sk}', \checkmark, \mathsf{sk}')\}
\end{aligned}
$$

and $\mathcal{B} = (Q_{\mathcal{B}}, \Sigma, \Delta_{\mathcal{B}}, F_{\mathcal{B}})$*, where*

$$Q_{\mathcal{B}} = (T \times [m+1]) \times (T \times [m+1]) \cup \{\mathsf{sk}\}$$

$$F_{\mathcal{B}} = \{\mathsf{sk}\}$$

$$\begin{aligned}
\Delta_{\mathcal{B}} = \{(&\begin{pmatrix}\langle t,j\rangle\\ \langle u,i\rangle\end{pmatrix}, t', \begin{pmatrix}\langle t',j'\rangle\\ \langle t',j'\rangle\end{pmatrix}) \mid t' \in T \textit{ and } i = j' = (j+1) \mod (m+1)\} \cup \\
&\{(\begin{pmatrix}\langle t,j\rangle\\ \langle u,i\rangle\end{pmatrix}, t', \mathsf{sk}) \mid t' \in T, i = (j+1) \mod (m+1) \textit{ and:} \\
&\qquad\qquad \neg H(t,t') \textit{ or } \neg V(u,t')\} \cup \\
&\{(\begin{pmatrix}\langle t,j\rangle\\ \langle u,i\rangle\end{pmatrix}, \bar{t}', \mathsf{sk}) \mid t' \in T, i = (j+1) \mod (m+1) \textit{ and:} \\
&\qquad\qquad H(t,t') \textit{ and } V(u,t')\} \cup \\
&\{(\begin{pmatrix}\langle t,j\rangle\\ \langle u,i\rangle\end{pmatrix}, t', \begin{pmatrix}\langle t',j'\rangle\\ \langle u,i\rangle\end{pmatrix}) \mid t' \in T \textit{ and } i \neq j' = (j+1) \mod (m+1)\} \cup \\
&\{(\begin{pmatrix}\langle t,j\rangle\\ \langle u,i\rangle\end{pmatrix}, \checkmark, \begin{pmatrix}\langle t,j\rangle\\ \langle u,i\rangle\end{pmatrix})\} \cup \\
&\{(\mathsf{sk}, a, \mathsf{sk}) \mid a \in \Sigma\}
\end{aligned}$$

Theorem 5.10.10. *Checking* $(1, n)$*-delayed and fair simulation on ABAs is EXPTIME-hard. Moreover,* $\mathcal{A}$ *can be taken with only universal states (UBA), and* $\mathcal{B}$ *an NBA.*

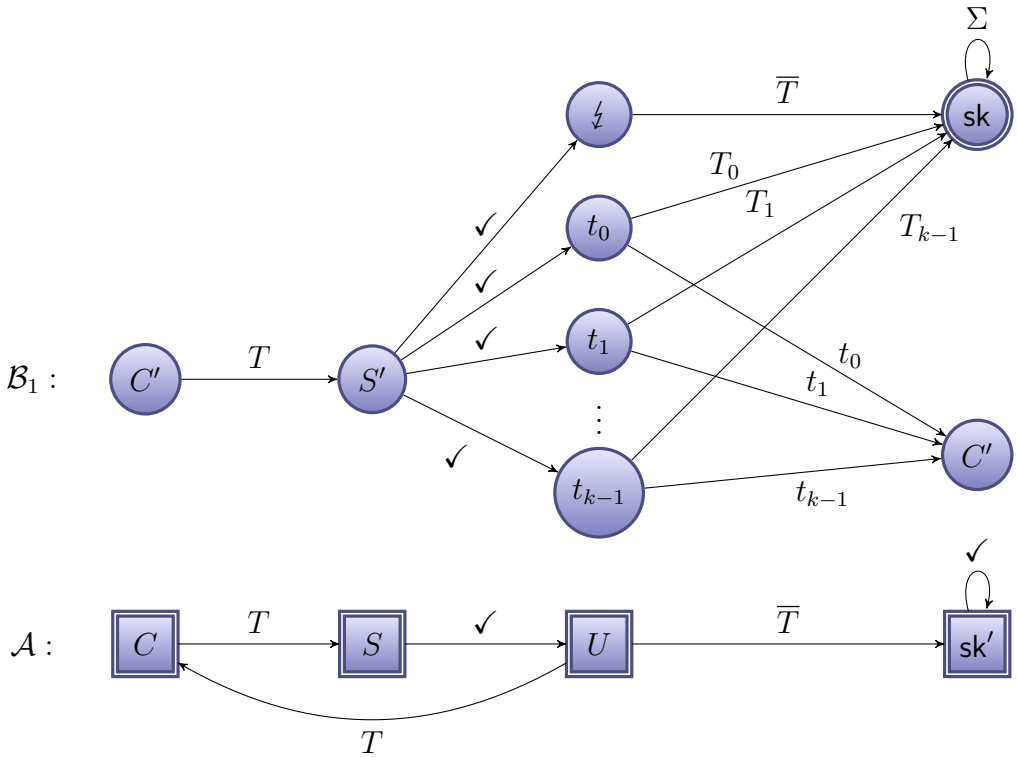
It is crucial that there exists exactly one left pebble on $\mathcal{A}$. For example, already with only two left pebbles, Duplicator could go to $\mathcal{A}$-states $\{t_0, t_1\}$, form which no common transition is available, and she would obviously win.

5.10.5 EXPTIME-hardness of $(1, \sqrt{n})$-simulations on NBAs

We show EXPTIME-hardness of $(1, \sqrt{n})$-delayed and fair multipebble simulation on NBAs. While in Section 5.10.4 the actions of Saboteur are modelled with universal states in $\mathcal{A}$, here we use extra existential states in $\mathcal{B}$. To force Duplicator to choose only one of such extra $\mathcal{B}$-states, we give Duplicator a high, but non-maximal $\sqrt{n}$ pebbles.

Recall $T = \{t_0, t_1, \ldots, t_{k-1}\}$ is the set of k tiles, $H, V \subseteq T \times T$ are the horizontal and vertical compatibility relations, respectively, and $r_0 = c_0 c_1 \ldots c_m$ is the initial row. Also, for any i, let $T_i = (T \cup \overline{T}) \setminus \{t_i, \bar{t}_i\}$. Again, we modify the construction of Section 5.10.3 in an incremental fashion. In particular, automaton $\mathcal{B}$ therein, is referred to as $\mathcal{B}_0$ in the following.

We model Saboteur's choices with gadget $\mathcal{B}_1$ below. The idea is to prevent Spoiler from playing a certain action $a \in \Sigma$ by adding in $\mathcal{B}$ an a-transition to the sink state sk.



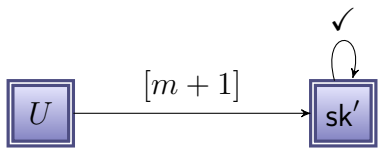
Intuitively, Duplicator/Saboteur selects the next tile to be placed by moving exactly one right pebble from S' to some state t_i. (We show later how to force Duplicator to select exactly one successor.) When Duplicator is on state t_i, Spoiler is forced to play either action t_i or $\bar{t}_i$; otherwise, the right pebble will end up in state sk, and Duplicator would win from there. If Duplicator believes that no valid tile can be placed next, then she goes to state $\lightning$. This forces Spoiler to play a tile $t \in T$, and Duplicator wins iff t is not valid. (Clearly, Duplicator will always keep a pebble on $\mathcal{B}_1$, since this gives her more chances to reach sk.)

We force Duplicator to select exactly one successor of S' with two devices:

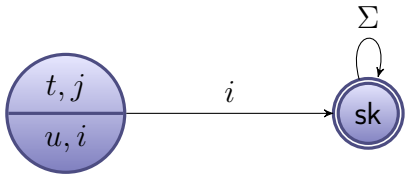
1. We allow Duplicator to use at most $m+2$ pebbles, $m+1$ of which are on ordinary states of shape $\frac{t,j}{u,i}$, and the remaining one is somewhere in $\mathcal{B}_1$.

2. To prevent Duplicator from moving any extra pebble from $\mathcal{B}_0$ to $\mathcal{B}_1$, we allow Spoiler to periodically check whether, for each $i \in [m+1]$, there exists a pebble on some state $\frac{t,j}{u,i}$. By the invariant of Section 5.10.3, Duplicator can always play as to satisfy these checks. To do so, we add the following transitions.

in $\mathcal{A}$:

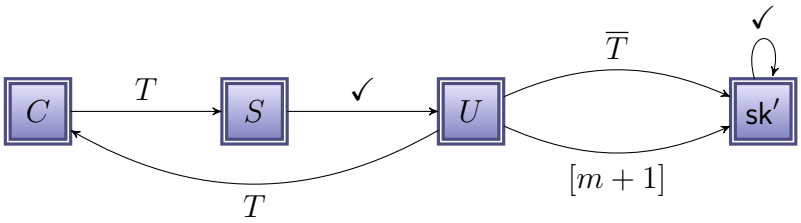


in $\mathcal{B}_0$:



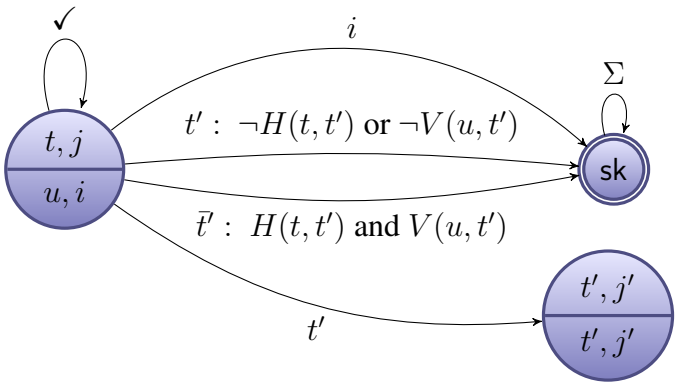
This solves (ModIssue-0) and (ModIssue-1). Summing up, the automaton $\mathcal{A}$ has the shape below.

$\mathcal{A}$:

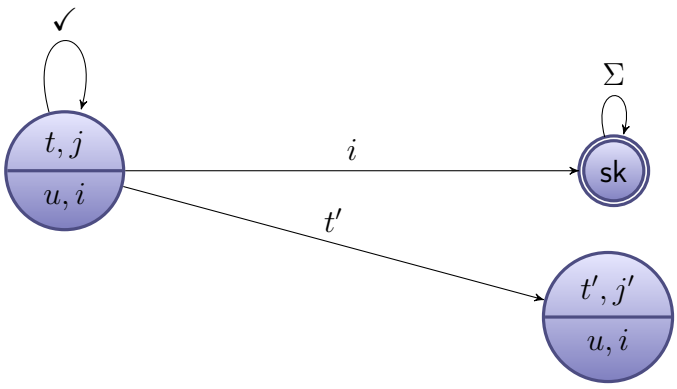


Automaton $\mathcal{B}$ is the union of $\mathcal{B}_0$ and $\mathcal{B}_1$, where $\mathcal{B}_1$ is as above, and $\mathcal{B}_0$ is as below (where $j' := (j+1) \mod (m+1)$).

(Trans-0) $\mathcal{B}_0, i = j'$:



(Trans-1) $\mathcal{B}_0, i \neq j'$:



A formal definition of the two automata follows.

Definition 5.10.11. *Let* $\Sigma = T \cup \overline{T} \cup \{\checkmark\} \cup [m+1]$. *Then,* $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma, \Delta_{\mathcal{A}}, F_{\mathcal{A}})$, *where*

$$
\begin{aligned}
Q_{\mathcal{A}} &= \{C, S, U, \mathsf{sk}'\} \\
F_{\mathcal{A}} &= Q_{\mathcal{A}} \\
\Delta_{\mathcal{A}} &= \{(C, t, S) \mid t \in T\} \ \cup \ \{(S, \checkmark, U)\} \cup \\
&\quad \{(U, t, C), (t, \bar{t}, \mathsf{sk}') \mid t \in T\} \ \cup \ \{(U, i, \mathsf{sk}') \mid i \in [m+1]\} \ \cup \ \{(\mathsf{sk}', \checkmark, \mathsf{sk}')\}
\end{aligned}
$$

and $\mathcal{B} = \mathcal{B}_0 \cup \mathcal{B}_1$, *where* $\mathcal{B}_0 = (Q_{\mathcal{B}_0}, \Sigma, \Delta_{\mathcal{B}_0}, F_{\mathcal{B}_0})$, *with*

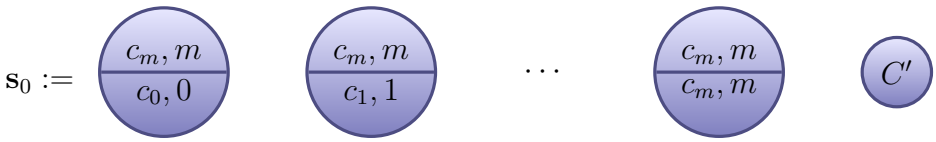
$$
\begin{aligned}
Q_{\mathcal{B}_0} &= (T \times [m+1]) \times (T \times [m+1]) \cup \{\mathsf{sk}\} \\
F_{\mathcal{B}_0} &= \{\mathsf{sk}\} \\
\Delta_{\mathcal{B}_0} &= \{(\begin{pmatrix} \langle t, j\rangle \\ \langle u, i\rangle \end{pmatrix}, t', \begin{pmatrix} \langle t', j'\rangle \\ \langle t', j'\rangle \end{pmatrix}) \mid t' \in T \textit{ and } i = j' = (j+1) \mod (m+1)\} \cup \\
&\quad \{(\begin{pmatrix} \langle t, j\rangle \\ \langle u, i\rangle \end{pmatrix}, t', \mathsf{sk}) \mid t' \in T, i = (j+1) \mod (m+1) \textit{ and:} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \neg H(t, t') \textit{ or } \neg V(u, t')\} \cup \\
&\quad \{(\begin{pmatrix} \langle t, j\rangle \\ \langle u, i\rangle \end{pmatrix}, \bar{t}', \mathsf{sk}) \mid t' \in T, i = (j+1) \mod (m+1) \textit{ and:} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad H(t, t') \textit{ and } V(u, t')\} \cup \\
&\quad \{(\begin{pmatrix} \langle t, j\rangle \\ \langle u, i\rangle \end{pmatrix}, t', \begin{pmatrix} \langle t', j'\rangle \\ \langle u, i\rangle \end{pmatrix}) \mid t' \in T \textit{ and } i \neq j' = (j+1) \mod (m+1)\} \cup \\
&\quad \{(\begin{pmatrix} \langle t, j\rangle \\ \langle u, i\rangle \end{pmatrix}, i, \mathsf{sk})\} \ \cup \ \{(\begin{pmatrix} \langle t, j\rangle \\ \langle u, i\rangle \end{pmatrix}, \checkmark, \begin{pmatrix} \langle t, j\rangle \\ \langle u, i\rangle \end{pmatrix})\} \ \cup \ \{(\mathsf{sk}, a, \mathsf{sk}) \mid a \in \Sigma\}
\end{aligned}
$$

and where $\mathcal{B}_1 = (Q_{\mathcal{B}_1}, \Sigma, \Delta_{\mathcal{B}_1}, F_{\mathcal{B}_1})$, *with*

$$
\begin{aligned}
Q_{\mathcal{B}_1} &= \{C', S', \lightning, \mathsf{sk}\} \cup T \\
F_{\mathcal{B}_1} &= \{\mathsf{sk}\} \\
\Delta_{\mathcal{B}_1} &= \{(C', t, S') \mid t \in T\} \ \cup \ \{(S', \checkmark, x) \mid x \in \{\lightning\} \cup T\} \ \cup \ \{(\lightning, \bar{t}, x) \mid t \in T\} \cup \\
&\quad \{(t_i, t_i, C'), (t_i, t, \mathsf{sk}) \mid t \in T_i = (T \cup \overline{T}) \setminus \{t_i, \bar{t}_i\}\} \ \cup \ \{(\mathsf{sk}, a, \mathsf{sk}) \mid a \in \Sigma\}
\end{aligned}
$$

Automaton $\mathcal{B}_0$ has $((m+1) \cdot k)^2 + 1$ states, while B_1 has $k+4$ states: Overall, $\mathcal{B}$ has $n := ((m+1) \cdot k)^2 + k + 5$ states.

The simulation game is played with a fixed number $m+2$ of pebbles. Given an initial row $r_0 = c_0 c_1 \cdots c_m$, the initial $m+1$ pebbles in $\mathcal{B}$ are like in Section 5.10.3, while the the remaining pebble is on C'.



From the considerations above, it follows that Constructor wins the domino-tiling game iff Spoiler wins the $(1, m+2)$-simulation game. (With n pebbles, Duplicator would always win.) Notice that $m + 2 = \sqrt{n-k-5}/k + 1$ is strictly less than n, the size of automaton $\mathcal{B}$. By adding enough dummy states to $\mathcal{B}$, we can obtain a larger automaton $\mathcal{B}'$ with $n' = k^2 n + k + 5$ states, for which $m + 2 = \sqrt{n'}$.

Theorem 5.10.12. *Checking $(1, \sqrt{n})$-delayed and fair simulation on ABA is EXPTIME-hard. Moreover, $\mathcal{A}$ can be taken to have only deterministic states (therefore, it can be an NBA or UBA), and $\mathcal{B}$ only existential states, i.e., an NBA.*

Chapter 6

Conclusions

Contents

6.1 Overview **245**

6.2 Applications **246**

6.3 Recapitulation and further work **247**

6.4 Advanced language inclusion checking **250**

6.4.1 Subsumption for automata over finite words 251

6.4.2 Subsumption in the rank-based approach 252

6.4.3 Subsumption in the Ramsey-based approach 253

6.4.4 Improved subsumption in Ramsey 256

6.4.5 Further work 263

6.5 Conclusions **264**

6.1 Overview

In this last chapter, we wrap-up by reviewing applications of our automata simplification procedures (Section 6.2), we illustrate ideas for further research (Section 6.3), and we put simulation preorders in perspective by pointing at recent work using simulations in advanced language inclusion algorithms (Section 6.4). Some final words in Section 6.5 conclude the chapter, and the thesis.

6.2 Applications

The methods developed in this thesis allow to reduce the size of automata. In this section, we briefly touch on three prominent applications where smaller automata are beneficial.

Universality and inclusion checking

Universality and inclusion checking are important problems in formal verification, with applications in the automata-theoretic approach to model-checking and in sanity checking of formal specifications (cf. Section 1.4.1). Known exact algorithms have a worst-case running time exponential in the number of states (and, since the problem is PSPACE-complete, this is unavoidable unless P = PSPACE). Therefore, reducing the size of automata before running expensive universality/inclusion checking procedures can substantially broaden the applicability of those methods.

For another use of simulations in providing powerful *subsumption preorders* for optimizing exact universality/inclusion checking procedures, see Section 6.4.3.

Automata complementation

The ability of complementing automata is a cornerstone feature of automata-theory. Automata complementation is necessary when translating from expressive temporal logics, e.g., monadic second-order logic, and it also has applications in checking the validity of LTL translation algorithms and in synthesis (cf. Section 5.1). Moreover, if practical complementation procedures are available, then language inclusion and universality can be reduced to the much easier language emptiness problem.

Therefore, in several applications the ability of obtaining small complement automata is crucial. Since complementation is an inherently exponential procedure, a "small" complement automaton can make the difference between being able or not to store in main memory a suitable representation for it. Thus, simplification procedures that reduce the size of the output automaton are essential to the feasibility of any complementation algorithm.

In this thesis, and in particular in Chapter 5, we develop techniques that can be used to simplify alternating automata. When alternating automata are used as intermediate representations for the complement automaton, reducing the former eventually results into a decrease in size of the latter. Has we have pointed out in Chapter 5, however, our

techniques cannot be immediately applied "out of the box" to complementation. In fact, we consider a restricted model of alternation with existential and universal states, while complementation procedures require a more general model of alternation with arbitrary boolean combinations. However, we believe our techniques can naturally be extended to the more expressive automata. Cf. also the discussion in Section 5.1.

Automata from temporal logic

Also translation algorithms for temporal logics (in particular, for LTL) can benefit from intermediate representation via alternating automata, and reducing the size of the latter can dramatically widen the applicability of LTL translation algorithms. The same remarks apply here as for complementation (see above, and Section 5.1).

6.3 Recapitulation and further work

We briefly recapitulate the contributions of the central chapters, and we touch upon some points that we believe are interesting for future work.

Chapter 3: Fixed-word simulations

We have shown that delayed fixed-word simulation is GFQ. However, it is PSPACE-complete to compute, so the first two questions try to deal with this high complexity.

1. *Tractable fragments of fixed-word simulations:* In Chapter 3, we have established that fixed-word delayed simulation is GFQ (cf. Theorem 3.4.2). Unfortunately, computing fixed-word simulation is computationally expensive (i.e., PSPACE-complete; cf. Theorem 3.4.5). However, by Corollary 2.4.8, any finer relation is GFQ as well. Therefore, one could look for tractable fragments of fixed-word delayed simulation.

2. *Rank-based algorithm for computing fixed-word delayed simulation:* In Section 3.4.4, we have reduced computing fixed-word delayed simulations to the universality problem of alternating Büchi automata. Another construction might be possible, by using ideas from the proof of Theorem 3.4.3. Indeed, in that proof we construct certain "small" ranks which are a sufficient and necessary condition for fixed-word simulation to hold (cf. Section 3.5). A direct construction of a

nondeterministic Büchi automaton guessing and checking these ranks might be possible.

Another possible line of investigation is more semantical, and aims at even coarser relations.

3. *Even coarser GFQ simulations:* Simulation-like GFQ relations even coarser than fixed-word delayed simulation might be possible. With Theorem 3.4.3, we have ruled out the possibility of obtaining one such relation by adding multiple pebbles. Nonetheless, alternative constructions yielding coarser GFQ relations might exists; finding these is important to understand the theoretical limits of quotienting Büchi automata.

Chapter 4: Jumping simulations

4. *Better understanding of how forward and backward simulations relate to each other:* The proxy simulation hierarchy comprises incomparable jumping-safe preorders. A priori, quotienting w.r.t. the induced GFQ equivalence might give very different results, and this seems to be an essential feature of the theory, i.e., forward and backward simulations are inherently incomparable.

   Practical experimentation can help finding good recipes for quotienting automata from real-life scenarios.

5. *Generalization to ABAs:* The theory of jumping simulations can be extended to alternating models. However, important notions like jumping-safety do not easily generalize in the presence of alternation, and more complex properties need to be introduced. Similar difficulties have been tackled with mediated preorder for ABAs [4].

6. *Coarser notions subsuming both jumping simulations and mediated preorder:* Jumping simulations and mediated preorder can both be described as jumping games, where jumps occur at certain rounds during the game; such jumps are prescribed in advance (cf. Section 4.7). This suggests the possibility of a general jumping scheme subsuming both notions, and which would yield an even coarser GFQ preorder.

Chapter 5: Multipebble simulations

First, and foremost, multipebble simulation can be studied for more expressive models of alternation.

7. *Multipebble simulations for more general alternating automata:* As we have already discussed in Sections 6.2 and 5.1, we consider a restricted model of alternation with existential and universal states, while established applications of alternating automata, such as in complementation and temporal logic translation procedures, require more general models of alternation (e.g., with arbitrary boolean combinations).

   Therefore, an important topic for future work is to generalize multipebble simulations to richer models of alternation. This can be done, in the case of alternating automata with epsilon transitions, along the lines of [54].

We have seen in Chapter 3 that multipebble simulations for NBAs under-approximate corresponding containment/fixed-word notions. A natural question is whether this carries over to alternating automata.

8. *GFQ direct containment over-approximating (k_0, k_1)-direct simulation on ABAs:* In nondeterministic models, multipebble direct simulation is GFQ because it under-approximates direct containment, which is GFQ by Theorem 3.3.4 (cf. Corollary 2.4.8). By Theorem 5.8.8, (k_0, k_1)-direct simulation is GFQ for ABAs (w.r.t. minimax quotients). However, it is not clear whether (k_0, k_1)-direct simulation is under-approximating some kind of GFQ direct containment on ABAs. Perhaps, one needs to consider a kind of *alternating* containment, in the spirit of [7].

9. *GFQ fixed-word delayed simulation over-approximating $(1, n)$-delayed simulation on ABAs:* For the delayed case, containment is not GFQ (cf. Example 3.3.2), and one has to resort to the finer fixed-word simulation. On ABAs, we have seen that only $(1, n)$-delayed simulation is GFQ (w.r.t. semielective quotients), therefore a trivial generalization of fixed-word delayed simulation to ABAs would not work, since it would presumably be at least as coarse as (n, n)-delayed simulation, which is not GFQ. In a sense, a restricted version of fixed-word simulation for ABAs should not allow Duplicator to use her extra power on the left.

   (In any case, it seems to be technically crucial that minimal successors of universal states in mixed classes are unique up to equivalence; cf. Lemma 5.8.18).

Another open problem is to compute multipebble simulations in practice.

10. *Antichain-based practical algorithms:* We have given two different kinds of algorithms for computing multipebble simulations:

    - *Theoretical algorithms:* In Section 5.10, we have presented algorithms of a theoretical nature, primarily geared towards providing matching upper- and lower-bounds.
    - *Practical algorithms:* In Section 5.9, we have given more practical algorithms, by characterizing multipebble simulations as fixpoints of certain monotone expressions in suitable transition systems.

    The algorithms of the second kind provide generic EXPTIME upper bounds on the problem, although they need to precompute and store in memory a transition system which is exponentially larger than the original automaton.

    However, the exponential transition systems used for computing multipebble simulations are naturally equipped with structural game simulations, which can be used to build symbolic representations of the sets involved in the fixpoint computation, in the style of *antichain methods* (cf. Section 6.4).

6.4 Advanced language inclusion checking

In this section, we discuss how simulations have recently found applications in optimized universality and inclusion checking algorithms for nondeterministic Büchi automata. Given two automata $\mathcal{A}$ and $\mathcal{B}$, the *language inclusion problem* amounts to check whether $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$; *language universality* is the special case when $\mathcal{L}(\mathcal{A}) = \Sigma^\omega$. For finite state automata (over finite or infinite words), using their closure properties one can build

1. an automaton $\mathcal{B}^c$ recognizing the complement language (say over infinite words) $\mathcal{L}(\mathcal{B}^c) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{B})$, and
2. a product automaton $\mathcal{A} \times \mathcal{B}^c$ recognizing the intersection $\mathcal{L}(\mathcal{A} \times \mathcal{B}^c) = \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B}^c) = \mathcal{L}(\mathcal{A}) \setminus \mathcal{L}(\mathcal{B})$,

and the inclusion problem can be reduced to an emptiness problem (which is easy):

$$\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B}) \quad \text{iff} \quad \mathcal{L}(\mathcal{A} \times \mathcal{B}^c) = \emptyset$$

Different complementation procedures for building $\mathcal{B}^c$ give rise to different inclusion algorithms. However, actually constructing the complement automaton is infeasible for any such procedure, since $\mathcal{B}^c$ can be exponentially larger than $\mathcal{B}$. Fortunately, to test $\mathcal{L}(\mathcal{A} \times \mathcal{B}^c) = \emptyset$, $\mathcal{B}^c$ needs not be constructed in its entirety, and clever exploration techniques that keep the complementation step implicit can be employed. These techniques crucially rely on the existence of structural *subsumption preorders* which exist by construction and need not be computed. Methods exploiting subsumption are sometimes called *antichain methods*, since they makes use of optimized representations of the state space based on antichains (i.e., sets with only incomparable elements), which serve as a symbolic representation of much larger sets.

For finite-word automata, there is just one complementation procedure [98, 68]. Antichain-based subsumption techniques have originally been developed to optimize such construction; see Section 6.4.1.

For Büchi automata, the situation is more articulated. The literature on complementation constructions for Büchi automata is rich and it will not be reviewed here; see, e.g., [117, 106] and references therein. Each complementation construction naturally gives rise to specific subsumption methods. In this section, we consider two such constructions, namely, the *rank* and *Ramsey-based* constructions, for which subsumption methods have been developed and proved useful in practice.

6.4.1 Subsumption for automata over finite words

The earliest reference on antichains methods we are aware of is [100] (see also the Ph.D thesis [99]), where it is observed that, when complementing a nondeterministic finite automaton (NFA), the full subset construction building the complete lattice of subsets can be avoided. Instead, a simplified construction maintaining only an antichain of $\subseteq$-minimal macrostates suffices (a macrostate in the subset automaton is just a set of states in the original automaton); in other words, the computation is done in an abstract lattice of antichains instead of the concrete lattice. In general, the former can be exponentially more succinct than the latter. Another early reference considering a similar technique, but for finite tree automata, is [112].

The antichain approach for the complementation of NFAs has been rediscovered again in [122], as an application of solving a more difficult (but related) problem about games of incomplete information [23] (see also [102] and Chapter 6 of [8]). More recently, the theory behind antichain methods has been worked out in more detail [37].

A crucial observation is the following, linking together antichains and simulations [35]:

Subset-like constructions come equipped with large structural simulations.

With "structural" we mean that these simulations are an inherent feature of subset constructions, and they do not need to be explicitly computed since they can be defined *a priori*. For example, in complementing NFAs, $\subseteq$-inclusion between macrostates (sets of states) is a structural simulation in the complement automaton.

As a recent improvement, [5] has shown how a simulation in the original automaton can be naturally lifted in a simulation in the complement automaton (the $\subseteq$-inclusion above corresponds to lifting the identity relation), further improving the already spectacular experimental results of [122].

In the next section, we present antichain methods for Büchi automata inclusion checking [35, 3].

6.4.2 Subsumption in the rank-based approach

The rank-based complementation construction has been proposed in [81], building on seminal work on progress measures [76, 77]. The idea is that rejecting computations can be marked with ranks measuring progress towards rejection. For finite automata, finite and *small* ranks suffice, and a complement automaton can be built which guesses ranks and checks them. From a complexity point of view, if $\mathcal{B}$ has n states, then the complement automaton $\mathcal{B}^c$ obtained with the rank construction has $2^{O(n \log n)}$ states in general, and which is tight by a matching lower bound [90].

Once a complement automaton $\mathcal{B}^c$ has been obtained, we are interested in checking the emptiness of the synchronized product $\mathcal{C} = \mathcal{A} \times \mathcal{B}^c$. Recall that a finite automaton has non-empty language iff, starting from an initial state, it is possible to visit a final state which is reachable from itself. In the rank-based approach, this is done with a simple fixpoint computation. Let $\mathbf{z}$ be the set of states which can visit accepting states infinitely often; $\mathbf{z}$ can be characterized as the following fixpoint:

$$\mathbf{z} = \nu \mathbf{x} \cdot \mu \mathbf{y} \cdot (\mathrm{Pre}(\mathbf{y}) \cup (F \cap \mathrm{Pre}(\mathbf{x})) \qquad (\dagger)$$

where $\mathrm{Pre}(\cdot)$ is the predecessor operator for $\mathcal{C}$ (cf. Section 2.4), and F is the set of final states therein. Then, $\mathcal{C}$ is non-empty iff $\mathbf{z}$ contains an initial state.

Of course, explicit computation of (†) is infeasible since $\mathcal{C}$ is too large to be constructed. [35, 36] has observed that a structural forward simulation $\sqsubseteq$ can be defined in

a natural way on $\mathcal{C}$ (the same observation has been previously made in [54]), and that the sets computed during the evaluation of (†) are upward-closed[1] w.r.t. $\sqsubseteq$. Thus, the computation can be carried symbolically by just storing and manipulating antichains of $\sqsubseteq$-minimal elements. With this technique, the performance can be improved by orders of magnitude w.r.t. previous work.

Antichains everywhere!

Antichain methods are now well-established in automata theory. They are based on a fundamental and natural observation about subset-like constructions and simulations, which has been rediscovered independently many times and which we believe will soon find its way into undergraduate books on automata.

The domain of applications of antichain-based methods is growing at an increasing pace. Examples include: tree automata inclusion checking [13], LTL-satisfiability and model-checking [32], LTL-realizability [46] (see also [47]), emptiness of alternating automata [56], QBF-satisfiability [15]; see also the tool ALASKA, which implements many of these techniques [33].

Finally, antichain methods can be utilized to compute multipebble simulations in practice, as we have pointed out in Section 6.3.

6.4.3 Subsumption in the Ramsey-based approach

The Ramsey-based approach has its origins in the original complementation construction by Büchi himself [18]. The idea is that, for a given automaton $\mathcal{B}$, it is possible to color all finite words from Σ^* with finitely many colors, s.t. words with the same color induce indistinguishable behavior in the automaton (where "indistinguishable" refers to certain reachability properties in $\mathcal{B}$). Eventually periodic infinite words of the form uv^ω, with $u, v \in \Sigma^*$, can also be colored, but with *pairs* of colors, by lifting the coloring of u and v. The way how the coloring is defined ensures that if two infinite words receive the same color, then they either both lie in $\mathcal{L}(\mathcal{B})$ or in $\Sigma^\omega \setminus \mathcal{L}(\mathcal{B})$; that is, the coloring respects the partitioning of Σ^ω into $\{\mathcal{L}(\mathcal{B}), \Sigma^\omega \setminus \mathcal{L}(\mathcal{B})\}$. Moreover, monochromatic sets of words are regular languages. Thus, a finite automaton $\mathcal{B}_{u,v}$ recognizing the equivalence class of uv^ω can be built. Since the coloring is finite, there are only finitely many different $\mathcal{B}_{u,v}$'s, and an automaton $\mathcal{B}^c$ can be built by assembling together all automata $\mathcal{B}_{u,v}$'s recognizing words $uv^\omega \notin \mathcal{L}(\mathcal{B})$. By construction, $\mathcal{B}^c$ is *sound*, in the

[1] In [35, 36], simulation is written the other way around, so their sets are actually downward-closed.

sense that it only recognizes words outside $\mathcal{L}(\mathcal{B})$. Moreover, $\mathcal{B}^c$ is also *complete*, in the sense that every word outside $\mathcal{L}(\mathcal{B})$ is in fact recognized by $\mathcal{B}^c$. This direction is more difficult, and Büchi established it by appealing to the Infinite Ramsey Theorem [101] (hence the name of the method). Namely, he proved that the coloring extends to all infinite words (not just to eventually periodic ones), i.e., *every* infinite word admits an eventually periodic coloring, and thus falls in the equivalence class of some uv^ω, and thus is accepted by some $\mathcal{B}_{u,v}$.

Early complementation constructions

The construction derived from Büchi's original paper [18] allows one to build a complement automaton $\mathcal{B}^c$ of doubly exponential size $2^{2^{O(n)}}$ (if $\mathcal{B}$ has n states). In [109], the same construction is revisited (see also [70]), and, by using similar ideas, a better construction is presented, with just a single exponential blow-up $2^{O(n^2)}$. If compared to the $2^{O(n \log n)}$ complexity of optimal constructions, like, e.g., the rank-based one, this is suboptimal, which perhaps explains why the construction remained dormant for about 20 years.

Termination analysis

Büchi's construction received renewed attention following the developments in the seemingly unrelated field of *program termination analysis*, when in 2001 [83] introduced the *size-change termination principle* (SCT). According to the SCT principle, a program has no infinite computation iff 1) the data it manipulates is well-founded, and 2) every infinite computation induces an infinitely decreasing sequence of values. [83] established a link between termination analysis and automata, by reducing SCT analysis to an inclusion problem between Büchi automata. However, after lamenting that no practical procedure was known for the latter problem, they developed a specialized algebraic approach for the SCT domain. Correctness once again relied on an application of Ramsey's Infinite Theorem, which is reminiscent of Büchi's original proof. This algebraic approach is at the heart of modern Ramsey-based inclusion checking algorithms for Büchi automata [51, 52, 2, 3].

Decision procedures for temporal logics

That there exists a "fundamental connection between termination analysis of programs and decision problems for ω-automata" has also been noted in [31]. The motivation

of [31] comes from decision procedures in temporal logics. They also avoid explicit automata complementation by adapting and generalizing the algebraic approach of [83] from SCT problems to validity of temporal formulas.

Back to complementation and language inclusion

The connection between SCT and Büchi automata is further strengthened in [50, 51], which lifts the algebraic approach of [83] to the full domain of inclusion problems for Büchi automata. More recently, the Ramsey-based approach has been also applied to checking inclusion of nondeterministic [53] and alternating parity automata [82].

Finally, [14] revisits the Ramsey-based complementation procedure itself. With a careful fine-tuning, it is shown that the worst case complexity can be reduced from $2^{O(n^2)}$ to $2^{O(n \log n)}$, thus bringing the Ramsey-based complementation procedure on a par with other optimal complementation methods. This result partly concludes the long story about the complexity of Büchi's complementation construction.

Subsumption

Like in the rank-based approach, subsumption is a key concept for taming the complexity of Ramsey-based algorithms. In the domain of SCT complexity, a subset-subsumption has been proposed [11], which corresponds to a subsumption for inclusion checking in a subclass of Büchi automata arising from SCT problems [50, 51]. Lifting subset-subsumption to all automata was left as an open problem. In [52], subset-subsumption is lifted to universality checking for the full class of Büchi automata; lifting subsumption to inclusion checking is left open.

In [2], we have extended the subset-subsumption of [51, 52] to full inclusion checking. Moreover, we have introduced a *simulation-based subsumption* which is coarser than the subset-based one, by lifting forward simulation to a subsumption preorder (subset-based subsumption corresponds to lifting the identity relation; see Section 6.4.4 below). Finally, in [3], we further generalize subsumption, by lifting simultaneously both *backward and forward simulation* into a subsumption preorder.

In the next section, we explain in more detail the Ramsey-based approach to inclusion checking, and we show how simulation relations give rise to advanced subsumption relations.

6.4.4 Improved subsumption in Ramsey

We explain the Ramsey-based approach to universality and inclusion checking, following the presentation of [3]. For simplicity, we actually deal with universality checking—but the theory can be generalized to inclusion. Our aim is to convince the reader that simulation preorders can serve as fundamental building blocks in powerful subsumption relations.

Fix a NBA $\mathcal{B}$. Our goal is to check universality of $\mathcal{B}$. A counter-example to universality of $\mathcal{B}$ is an infinite word $w \notin \mathcal{L}(\mathcal{B})$. Since ω-regular languages are closed under complementation, and a regular language is non-empty iff it contains an eventually periodic word, it suffices to look for eventually periodic counter-examples. Thus, universality checking of $\mathcal{B}$ amounts to find two finite words $u, v \in \Sigma^+$ s.t. $uv^\omega \notin \mathcal{L}(\mathcal{B})$. However, Σ^+ is an infinite object which is difficult to work with. Eventually, we want to approximate this infinite semigroup with a small, finite one.

The main observation is that, once $\mathcal{B}$ is fixed, not all finite words need to be considered separately. Fix a non-empty finite word $w \in \Sigma^+$. For every pair of states p and q in $\mathcal{Q}$, exactly one of the following three things may happen (this idea goes back to Büchi's theorem [18]; see also [95] and Chapter 5 of [49]):

$\perp$: There is no path from p to q over w.

0: There is a path from p to q over w passing through an accepting state.

1: There is a path from p to q over w, but no path passes through an accepting state.

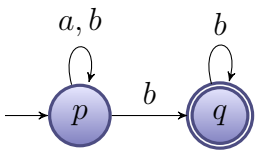
Accordingly, we define a domain of *observables* $\mathbb{D} = \{\perp, 0, 1\}$. Different words inducing the same observables for every pair of states are indistinguishable from $\mathcal{B}$'s point of view, and can be considered equivalent for universality checking purposes.

Let $M = \mathbb{D}^{Q \times Q}$ be the set of $Q \times Q$-matrices with values in $\mathbb{D}$. For every $w \in \Sigma^+$, we define a matrix $\mu_w \in M$, s.t., for every pair of states $(p, q) \in Q \times Q$, cell $\mu_w(p, q)$ is defined as follows:

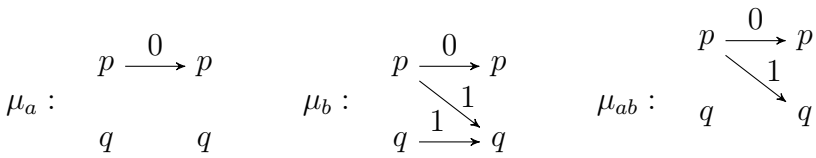
$$\mu_w(p,q) = \begin{cases} \perp & \text{if } \neg(p \xrightarrow{w} q) \\ 1 & \text{if } p \xrightarrow{w}_F q \\ 0 & \text{if } p \xrightarrow{w} q \text{ and } \neg(p \xrightarrow{w}_F q) \end{cases}$$

Therefore, μ is a mapping from Σ^+ to M. An implementation will not usually store $\perp$-cells, and matrix μ_w is represented as a set of $\{0, 1\}$-labelled edges in $Q \times \{0, 1\} \times Q$:

$$\mu_w \equiv \{p \xrightarrow{x} q \mid \text{if } x = \mu_w(p, q) \neq \perp\}$$



(a) Automaton $\mathcal{B}$



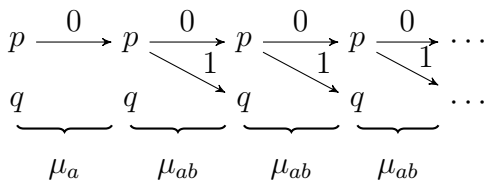
(b) Matrices over $\mathcal{B}$

Figure 6.1: Example automaton $\mathcal{B}$ and matrices

Since the two representations are equivalent, we sometimes identify matrices with the corresponding set of edges.

As a running example, consider the automaton $\mathcal{B}$ in Figure 6.1(a). Example matrices are in Figure 6.1(b). Intuitively, a matrix μ_w simultaneously represents all possible behaviors of the automaton over the word w; it can be seen as a refinement of the subset construction, where in the latter only the destination endpoints would be recorded (the ones on the right). Matrices are called with various names in the literature, like *graphs* [83, 51, 2] and *boxes* [31].

Matrices alone can be used to witness that $\mathcal{B}$ is not universal. For example, to show that the infinite word $w = a(ab)^\omega$ is not in $\mathcal{L}(\mathcal{B})$, it suffices to concatenate the respective matrices,



and notice that there is no infinite path starting at the initial state p labelled with infinitely many 1's. This idea can be generalized to test any pair of matrices μ_u and μ_v for a counterexample of the form uv^ω.

Definition 6.4.1 (Test)**.** *Given two matrices μ_0 and μ_1, let* $\mathsf{test}(\mu_0, \mu_1) = \top$ *if there*

exists an infinite path

$$\pi = \underbrace{p \xrightarrow{x} q_0}_{\in\mu_0} \underbrace{\xrightarrow{y_0} q_1}_{\in\mu_1} \underbrace{\xrightarrow{y_1} q_2}_{\in\mu_1} \underbrace{\xrightarrow{y_2}}_{\in\mu_1} \cdots$$

with $p \in I$*, and* $y_i = 1$ *for infinitely many* i*'s. Otherwise, let* $\mathsf{test}(\mu_0, \mu_1) = \bot$.

In the literature, this test operation is variously called *double-graph search* [51], *two-arc test* [52] or *lasso finding test* [2]. Testing can be done in polynomial time by solving a simple graph-theoretic problem. The test is sound in the following sense.

Lemma 6.4.2. *For every* $u, v \in \Sigma^+$, $uv^\omega \in \mathcal{L}(\mathcal{B})$ *iff* $\mathsf{test}(\mu_u, \mu_v) = \top$.

In our running example, $\mathsf{test}(\mu_a, \mu_{ab}) = \bot$, therefore $a(ab)^\omega \notin \mathcal{L}(\mathcal{B})$; a positive case is $\mathsf{test}(\mu_b, \mu_b) = \top$, thus $b^\omega \in \mathcal{L}(\mathcal{B})$.

This seems to suggest an NP algorithm for solving non-universality: Guess two matrices μ_0 and μ_1, which are small objects, and verify in polynomial time $\mathsf{test}(\mu_0, \mu_1) = \bot$. Unfortunately, not all matrices failing the test are actual counterexamples to universality, since an arbitrary matrix does not necessarily encode meaningful information. For example, in the automaton $\mathcal{B}$ any matrix containing a non-$\bot$ edge $q \xrightarrow{0,1} p$ is not valid, in the sense that no word w can realize this behavior. To remedy to this problem, we introduce the subset of *valid matrices*.

Definition 6.4.3 (Valid matrices)**.** *A matrix* $\mu_0 \in M$ *is* valid *iff it is in the range of* μ*, i.e., iff there exists* $w \in \Sigma^+$ *s.t.* $\mu_0 = \mu_w$*. Let* M^v *be the set of valid matrices.*

We can modify our purported NP algorithm to additionally check that μ_0 and μ_1 are in M^v. However, this last operation is PSPACE-complete[2], and it has better to be so under standard complexity-theoretic assumptions (since universality is PSPACE-complete).

A more practical method is to adopt a generative approach, and iteratively construct every valid matrix via a composition operation. To this end, we endow the set of matrices M with a semigroup structure. First, turn $\mathbb{D}$ into a semiring, by adding a multiplication "$\cdot$" and an addition "$+$" operation. Intuitively, the multiplication describes how observables are composed along a path, while the addition reflects how the observables from different paths are combined together (like in weighted automata). They are defined according to the two Cayley tables below:

[2]Checking that a matrix is valid is equivalent to the emptiness problem for the intersection of a family of NFAs, which is PSPACE-complete.

$\cdot$	$\bot$	0	1
$\bot$	$\bot$	$\bot$	$\bot$
0	$\bot$	0	1
1	$\bot$	1	1

$+$	$\bot$	0	1
$\bot$	$\bot$	0	1
0	0	0	1
1	1	1	1

With these definitions, the value for $\mu_{w_0 \cdot w_1}(p, q)$ can be computed recursively based on the values for $\mu_{w_0}(p, \cdot)$ and $\mu_{w_1}(\cdot, q)$:

$$(\mu_{w_0 \cdot w_1})(p, q) = \sum_{r \in Q} \mu_{w_0}(p, r) \cdot \mu_{w_1}(r, q) \tag{†}$$

This is a semantic property of mapping μ. Intuitively, there might be different ways of getting from p to q, and by the property above it suffices to find the "best way" (according to the semantics of "+") over all intermediate states r.

Based on this observation, we *define* a product operation in the semigroup of matrices M using the formula (†) above [95].

Definition 6.4.4 (Product of matrices)**.** *For two matrices μ_0 and μ_1, their* product $\mu_0 \cdot \mu_1$ *is defined as follows, for every $p, q \in Q$:*

$$(\mu_0 \cdot \mu_1)(p, q) := \sum_{r \in Q} \mu_0(p, r) \cdot \mu_1(r, q)$$

(This product operation coincides with the usual matrix product over the semiring $\mathbb{D}$.)

In the running example, valid matrices are depicted in Figure 6.1(b), their products are shown in Figure 6.2(a), and the full Cayley table is given in Figure 6.2(b).

It is easy to show that multiplication is an associative operation, and that valid matrices are preserved under multiplication. Therefore, valid matrices with multiplication form a semigroup. Additionally, since μ is a morphism of semigroups $\Sigma^+ \mapsto M^v$ (by definition) and Σ^+ is (finitely) generated by Σ, M^v is generated by $M_1^v := \{\mu_a \mid a \in \Sigma\}$. In other words, every valid matrix $\mu_{a_0 a_1 \cdots a_{k-1}}$ can be obtained as a product of 1-letter valid matrices $\mu_{a_0} \mu_{a_1} \cdots \mu_{a_{k-1}}$.

This suggests a generate&test approach to universality checking: Since Σ^+ is finitely generated by Σ, we can compute all valid matrices in $\mathbb{M}$ by multiplication from 1-letter matrices $\mu_a, a \in \Sigma$. Then, it suffices to test all pairs of generated matrices for counterexamples to universality. On automaton $\mathcal{B}$, we obtain the valid matrices in Figure 6.1(b).

$$\underbrace{\begin{tikzcd} p \arrow[r, "0"] & p \arrow[r, "0"] & p \\ q & q & q \end{tikzcd}}_{\mu_a} \cdot \underbrace{\begin{tikzcd} p \arrow[r, "0"] & p \\ q & q \end{tikzcd}}_{\mu_a} = \underbrace{\begin{tikzcd} p \arrow[r, "0"] & p \\ q & q \end{tikzcd}}_{\mu_a}$$

$$\underbrace{\begin{tikzcd} p \arrow[r, "0"] & p \arrow[r, "0"] & p \\ q & q \arrow[r, "1"] \arrow[ur, "1"] & q \end{tikzcd}}_{\mu_a} \cdot \underbrace{\begin{tikzcd} p \arrow[r, "0"] & p \arrow[r, "0"] & p \\ q & q \arrow[r, "1"] \arrow[ur, "1"] & q \end{tikzcd}}_{\mu_b} = \underbrace{\begin{tikzcd} p \arrow[r, "0"] & p \\ q \arrow[ur, "1"] & q \end{tikzcd}}_{\mu_{ab}}$$

$$\underbrace{\begin{tikzcd} p \arrow[r, "0"] & p \arrow[r, "0"] & p \\ q \arrow[r, "1"] \arrow[ur, "1"] & q \arrow[r, "1"] \arrow[ur, "1"] & q \end{tikzcd}}_{\mu_b} \cdot \underbrace{\begin{tikzcd} p \arrow[r, "0"] & p \arrow[r, "0"] & p \\ q \arrow[r, "1"] \arrow[ur, "1"] & q \arrow[r, "1"] \arrow[ur, "1"] & q \end{tikzcd}}_{\mu_b} = \underbrace{\begin{tikzcd} p \arrow[r, "0"] & p \\ q \arrow[r, "1"] \arrow[ur, "1"] & q \end{tikzcd}}_{\mu_b}$$

$$\underbrace{\begin{tikzcd} p \arrow[r, "0"] & p \arrow[r, "0"] & p \\ q \arrow[r, "1"] \arrow[ur, "1"] & q \arrow[r, "1"] \arrow[ur, "1"] & q \end{tikzcd}}_{\mu_b} \cdot \underbrace{\begin{tikzcd} p \arrow[r, "0"] & p \arrow[r, "0"] & p \\ q & q & q \end{tikzcd}}_{\mu_a} = \underbrace{\begin{tikzcd} p \arrow[r, "0"] & p \\ q & q \end{tikzcd}}_{\mu_a}$$

(a) Matrix products

$\cdot$	μ_a	μ_b	μ_{ab}
μ_a	μ_a	μ_{ab}	μ_{ab}
μ_b	μ_a	μ_b	μ_{ab}
μ_{ab}	μ_a	μ_{ab}	μ_{ab}

(b) Multiplication table

Figure 6.2: Example multiplication of matrices

Subsumption

The problem with a naïve implementation of the algorithm above is that M^v is still exponentially large, and, in general, one cannot hope to enumerate all valid matrices. This is where subsumption enters into the scene: It is possible to define certain *subsumption preorders* which allow to prune the search space considerably, making the algorithm practical.

Definition 6.4.5 (Subsumption of edges [3])**.** *For two edges $e = p \xrightarrow{x} q$ and $e' = p' \xrightarrow{x'} q'$ in $Q \times \mathbb{D} \times Q$, let $e \sqsubseteq e'$ iff $p \sqsubseteq^{\mathrm{bw}} p'$, $x + x' = x'$ (i.e., iff $x \leq x'$ in the order $\perp < 0 < 1$.), and $q \sqsubseteq^{\mathrm{di}} q'$.*

Intuitively, subsumption-larger edges can simulate smaller ones in witnessing an accepting loop. We use subsumption in two ways:

- Section 6.4.4.1: In minimization of matrices, by removing subsumption-smaller edges.
- Section 6.4.4.2: In comparing matrices, by lifting the subsumption from edges to matrices, and discarding subsumption-larger matrices.

Remark 6.4.6. The subsumption preorder presented here has been proposed in [3]. It subsumes the following notions which have been previously considered:

- The original subset-subsumption $\subseteq$ from [11] (subsequently adopted by [51, 52]): For two edges $e = p \xrightarrow{x} q$ and $e' = p' \xrightarrow{x'} q'$, let $e \subseteq e'$ iff $p = p'$, $x + x' = x'$, and $q = q'$.
- The improved subsumption $\sqsubseteq'$ based on direct simulation from [2]: For two edges $e = p \xrightarrow{x} q$ and $e' = p' \xrightarrow{x'} q'$, let $e \sqsubseteq' e'$ iff $p = p'$, $x + x' = x'$, and $q \sqsubseteq^{\mathrm{di}} q'$.

6.4.4.1 Minimization of matrices

The first observation is that not every edge in a matrix is necessary, and we can simplify matrices by removing those edges which are subsumed by $\sqsubseteq$-larger ones. Matrices with only maximal edges are called *minimized matrices* (they cannot be further simplified). Let M^m be the set of minimized matrices.

Ideally, we would like to manipulate only matrices from M^m (instead of all valid matrices). To do so, composition and test on minimized matrices should faithfully mimic the corresponding operations on original matrices. However, the minimization

operation removes edges in a non-monotonic way w.r.t. test and multiplication; for example, the test might pass on a pair (μ_0, μ_1) and suddenly fail for their minimization (μ'_0, μ'_1), because an edge used to build an infinite path is no longer present.

To remedy to this problem, we define two alternative notions of test and composition on minimized matrices, which take into consideration edges deleted during minimization. The two notions are called *jumping*, since they allow the computation to jump to certain intermediate states (this is related to the notion of jumping simulation from Chapter 4).

Definition 6.4.7 (Jumping test)**.** *Given two matrices μ_0 and μ_1, $\mathsf{jtest}(\mu_0, \mu_1)$ holds iff there exists an infinite jumping path*

$$\pi = \underbrace{p \xrightarrow{x} q_0}_{\in\mu_0} \sqsubseteq^{\mathrm{bw}} \underbrace{q'_0 \xrightarrow{y_0} q_1}_{\in\mu_1} \sqsubseteq^{\mathrm{bw}} \underbrace{q'_1 \xrightarrow{y_1} q_2}_{\in\mu_1} \sqsubseteq^{\mathrm{bw}} \underbrace{q'_2 \xrightarrow{y_2}}_{\in\mu_1} \cdots$$

with $p \in I$, and $y_i = 1$ for infinitely many i's.

Definition 6.4.8 (Jumping product of matrices)**.** *For two matrices μ_0 and μ_1, their* jumping product *$\mu_0 \circ \mu_1$ is defined as follows, for every $p, q \in Q$:*

$$(\mu_0 \circ \mu_1)(p, q) := \sum_{r_0 \sqsubseteq^{\mathrm{bw}} r_1} \mu_0(p, r_0) \cdot \mu_1(r_1, q)$$

Jumping product is also an associative operation, and the set of matrices with jumping product is a semigroup. The jumping operations adequately mimic the non-jumping ones by the following lemma [3].

Lemma 6.4.9. *For two valid matrices μ_0 and μ_1, let μ'_0 and μ'_1 be their respective minimizations. Then,*

P1) Preservation of test: $\mathsf{test}(\mu_0, \mu_1) = \mathsf{jtest}(\mu'_0, \mu'_1)$.

P2) Preservation of composition: $\mu'_0 \circ \mu'_1$ is a minimization[3] *of $\mu_0 \cdot \mu_1$.*

Minimization is beneficial since it produces sparser matrices, which are easier to handle; moreover, different valid matrices might be represented by the same minimized matrix, which reduces the search space. By lifting the subsumption from edges to matrices, we can prune the search space even more, as explained next.

[3]We are simplifying a technicality here, since composition is actually preserved only in a weaker sense. To obtain formally correct statements, replace "minimization" with "representative", where the notion of *representative* (cf. [3]) generalizes that of minimization. For simplicity, we do not introduce representatives here.

6.4.4.2 Subsumption of matrices

Also matrices can be compared with each other, by lifting the subsumption preorder.

Definition 6.4.10 (Subsumption of matrices)**.** *For two matrices μ_0 and μ_1 in M, let $\mu_0 \preceq \mu_1$ iff, for any edge e in μ_0, there exists an edge e' in μ_1 s.t. $e \sqsubseteq e'$.*

It can be shown that subsumption is monotone w.r.t. jumping composition and test, in the following sense [3].

Lemma 6.4.11. *For minimized matrices $\mu_0 \preceq \mu_0'$ and $\mu_1 \preceq \mu_1'$,*

M1) Monotonicity of test: $\mathsf{jtest}(\mu_0, \mu_1) \leq \mathsf{jtest}(\mu_0', \mu_1')$.

M2) Monotonicity of composition: $\mu_0 \circ \mu_1 \preceq \mu_0' \circ \mu_1'$.

Whenever two comparable matrices $\mu_0 \preceq \mu_1$ are generated, we discard the larger one μ_1, since no counter-example is lost; this is sound since subsumption preserves counter-examples both in the present and in the future:

- *In the present:* By condition M1), if a pair (μ_0', μ_1') fails the test, then every pair (μ_0, μ_1) s.t. $\mu_0 \preceq \mu_0'$ and $\mu_1 \preceq \mu_1'$ fails the test as well.
- *In the future:* By condition M2), subsumption is preserved under composition, therefore also "future generations" will preserve counter-examples.

We obtain an optimized generate&test algorithm, combining minimization and subsumption: Generate all $\preceq$-minimal elements in M^m by jumping multiplication and minimization from 1-letter minimized matrices, and test all pairs of generated matrices against the jumping test; whenever two comparable matrices $\mu_0 \preceq \mu_1$ are discovered, discard μ_1. Minimization and subsumption contribute greatly to the practicality of the algorithm; see the results in [3]. A website is available containing the source code of the implementation and all the experimental results [1].

6.4.5 Further work

In the previous section, we have shown how the Ramsey-based approach to language inclusion checking can benefit from a subsumption relation based on direct and backward simulation. An important open question is whether the generalized simulation relations studied in this thesis can be used to define an even coarser subsumption for the Ramsey-based approach (and for other approaches as well). From a practical point of view, jumping simulations (cf. Chapter 4) are a particularly suitable candidate, since they can be efficiently computed.

6.5 Conclusions

In this thesis, we have studied several generalized simulation-like relations, for both nondeterministic and alternating Büchi automata, and we have investigated their properties, as suggested by the simulation desiderata. In particular, we have focused on quotienting, and we have argued that the equivalences induced by our generalized simulations can be used to reduce the size of automata. The reduction in size always improves on previously known notions, and, in most cases, the reduction factor can be arbitrarily larger. Finally, we have indicated that simulation preorders can be used to substantially improve advanced language inclusion checking algorithms for NBAs.

Bibliography

[1] http://www.languageinclusion.org 263

[2] Abdulla, P., Chen, Y.F., Clemente, L., Holik, L., Hong, C.D., Mayr, R., Vojnar, T.: Simulation Subsumption in Ramsey-Based Büchi Automata Universality and Inclusion Testing. In: Touili, T., Cook, B., Jackson, P. (eds.) Computer Aided Verification. LNCS, vol. 6174, pp. 132–147. Springer Berlin / Heidelberg, Berlin, Heidelberg (2010), http://dx.doi.org/10.1007/978-3-642-14295-6_14 13, 254, 255, 257, 258, 261

[3] Abdulla, P., Chen, Y.F., Clemente, L., Holik, L., Hong, C.D., Mayr, R., Vojnar, T.: Advanced Ramsey-based Buechi Automata Inclusion Testing. In: International Conference on Concurrency Theory (Sep 2011) 13, 252, 254, 255, 256, 261, 262, 263

[4] Abdulla, P., Chen, Y.F., Holik, L., Vojnar, T.: Mediating for Reduction. In: Foundations of Software Technology and Theoretical Computer Science. pp. 1–12. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2009) 74, 109, 110, 248

[5] Abdulla, P.A., Chen, Y.F., Holik, L., Mayr, R., Vojnar, T.: When Simulation Meets Antichains. In: Tools and Algorithms for the Construction and Analysis of Systems. Paphos, Cyprus (2010), http://hal.inria.fr/inria-00460294/en/ 5, 252

[6] Aceto, L., Ingólfsdóttir, A., Larsen, K.G., Srba, J.: Reactive systems: Modelling, Specification and Verification. Cambridge University Press (Jul 2007) 4

[7] Alur, R., Henzinger, T.A., Kupferman, O., Vardi, M.Y.: Alternating Refinement Relations. In: International Conference on Concurrency Theory. Lecture Notes in Computer Science, vol. 1466, pp. 163–178. Springer-Verlag (1998) 249

[8] Apt, K., Graedel, E. (eds.): Lectures in Game Theory for Computer Scientists. Cambridge University Press (2011) 251

[9] Arnold, A., Dicky, A., Nivat, M.: A note about minimal non-deterministic automata. Bulletin EATCS 47, 166–169 (Jun 1992) 6

[10] Aziz, A., Singhal, V., Swamy, G.M., Brayton, R.K.: Minimizing Interacting Finite State Machines. Tech. Rep. UCB/ERL M93/68, UoC, Berkeley (1993) 9, 31

[11] Ben Amram, A.M., Lee, C.S.: Program termination analysis in polynomial time. ACM Trans. Program. Lang. Syst. 29 (Jan 2007), http://dx.doi.org/10.1145/1180475.1180480 255, 261

[12] von Bomhard, T.: Minimization of Tree Automata. Master's thesis, Universität des Saarlandes (2008) 10

[13] Bouajjani, A., Habermehl, P., Holik, L., Touili, T., Vojnar, T.: Antichain-Based Universality and Inclusion Testing over Nondeterministic Finite Tree Automata. In: Implementation and Applications of Automata. pp. 57–67. CIAA '08, Springer-Verlag, Berlin, Heidelberg (2008), http://dx.doi.org/10.1007/978-3-540-70844-5_7 5, 253

[14] Breuers, S., Löding, C., Olschewski, J.: Improved Ramsey-based Buechi Complementation. In: Foundations of Software Science and Computation Structures. FOSSACS'12 (2012) 255

[15] Brihaye, T., Bruyère, V., Doyen, L., Ducobu, M., Raskin, J.F.: Antichain-Based QBF Solving. In: Bultan, T., Hsiung, P.A. (eds.) Automated Technology for Verification and Analysis. Lecture Notes in Computer Science, vol. 6996, pp. 183–197. Springer (2011) 253

[16] Büchi, J.R.: Weak Second-Order Arithmetic and Finite Automata. Zeitschrift für Mathematische Logik und Grundlagen der Mathematik 6(1-6), 66–92 (1960), http://dx.doi.org/10.1002/malq.19600060105 4

[17] Büchi, J.R.: Using Determinancy of Games to Eliminate Quantifiers. In: Karpinski, M. (ed.) Fundamentals of Computation Theory. Lecture Notes in Computer Science, vol. 56, pp. 367–378. Springer (Sep 1977) 26

[18] Büchi, J.R.: On a Decision Method in Restricted Second-Order Arithmetic. In: International Congress on Logic, Methodology, and Philosophy of Science. pp. 1–11. Stanford University Press (1962) 253, 254, 256

[19] Bustan, D., Grumberg, O.: Simulation-based minimization. ACM Trans. Comput. Logic 4(2), 181–206 (Apr 2003), http://dx.doi.org/10.1145/635499.635502 24, 31, 32, 107

[20] Bustan, D., Grumberg, O.: Applicability of fair simulation. Informatio and Computation 194, 1–18 (Oct 2004), http://portal.acm.org/citation.cfm?id=1036108 9

[21] Carrez, C.: On the Minimalization of Non-deterministic Automaton. Tech. rep., Laboratoire de Calcul de la Faculté des Sciences de l'Université de Lille (1970) 6

[22] Chandra, A.K., Kozen, D.C., Stockmeyer, L.J.: Alternation. J. ACM 28(1), 114–133 (Jan 1981), http://dx.doi.org/10.1145/322234.322243 10

[23] Chatterjee, K., Doyen, L., Henzinger, T.A., Raskin, J.F.: Algorithms for Omega-Regular Games with Imperfect Information. In: Ésik, Z. (ed.) Computer Science Logic. Lecture Notes in Computer Science, vol. 4207, pp. 287–302. Springer (2006) 251

[24] Chlebus, B.S.: Domino-tiling games. Journal of Computer and System Sciences 32, 374–392 (Jun 1986), http://portal.acm.org/citation.cfm?id=9118.9125 232, 233

[25] Cipra, B.: How Number Theory Got the Best of the Pentium Chip. Science 267, 175+ (Jan 1995) 3

[26] Clarke, E.M., Emerson, E.A.: Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic. In: Logic of Programs: Workshop, Yorktown Heights, NY, May 1981. Springer (1981) 4

[27] Clarke, E.M., Wing, J.M., Alur, R., Cleaveland, R., Dill, D., Emerson, A., Garland, S., German, S., Guttag, J., Hall, A., Henzinger, T., Holzmann, G., Jones, C., Kurshan, R., Leveson, N., McMillan, K., Moore, J., Peled, D.,

Pnueli, A., Rushby, J., Shankar, N., Sifakis, J., Sistla, P., Steffen, B., Wolper, P., Woodcock, J., Zave, P.: Formal methods: state of the art and future directions. ACM Computing Surveys 28(4), 626–643 (1996), citeseer.ist.psu.edu/clarke96formal.html 4

[28] Clemente, L.: Büchi Automata Can Have Smaller Quotients. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) Automata, Languages and Programming, Lecture Notes in Computer Science, vol. 6756, chap. 20, pp. 258–270. Springer Berlin / Heidelberg, Berlin, Heidelberg (2011), http://arxiv.org/pdf/1102.3285 36, 73, 101

[29] Clemente, L., Mayr, R.: Multipebble Simulations for Alternating Automata - (Extended Abstract). In: International Conference on Concurrency Theory. LNCS, vol. 6269, pp. 297–312. Springer-Verlag (2010), http://dx.doi.org/10.1007/978-3-642-15375-4_21 xi, 119, 121, 134, 159, 160, 174, 175, 176, 224

[30] Coe, T.: Inside the Pentium FDIV bug. Dr. Dobbs Journal (Apr 1996) 3

[31] Dax, C., Hofmann, M., Lange, M.: A Proof System for the Linear Time mu-Calculus. In: Foundations of Software Technology and Theoretical Computer Science (2006) 254, 255, 257

[32] De Wulf, M., Doyen, L., Maquet, N., Raskin, J.F.: Antichains: alternative algorithms for LTL satisfiability and model-checking. In: Tools and algorithms for the construction and analysis of systems. pp. 63–77. TACAS'08, Springer-Verlag, Berlin, Heidelberg (2008), http://dl.acm.org/citation.cfm?id=1792734.1792743 253

[33] De Wulf, M., Doyen, L., Maquet, N., Raskin, J.F.: ALASKA: Antichains for Logic, Automata and Symbolic Kripke Structures Analysis. In: Automated Technology for Verification and Analysis. pp. 240–245 (2008) 253

[34] Dill, D.L., Hu, A.J., Wont-Toi, H.: Checking for Language Inclusion Using Simulation Preorders. In: Computer Aided Verification. LNCS, vol. 575. Springer-Verlag (1991), http://dx.doi.org/10.1007/3-540-55179-4_25 9, 26, 29

[35] Doyen, L., Raskin, J.F.: Improved algorithms for the automata-based approach to model-checking. In: Tools and Algorithms for the Construction and Analysis of Systems. pp. 451–465. TACAS'07, Springer-Verlag, Berlin, Heidelberg (2007), http://portal.acm.org/citation.cfm?id=1763552 5, 252, 253

[36] Doyen, L., Raskin, J.F.: Antichains for the Automata-Based Approach to Model-Checking. Logical Methods in Computer Science 5(1) (2009) 5, 252, 253

[37] Doyen, L., Raskin, J.F.: Antichain Algorithms for Finite Automata. In: Tools and Algorithms for the Construction and Analysis of Systems. pp. 2–22 (2010) 5, 251

[38] Edelman, A.: The mathematics of the Pentium division bug. SIAM Review 39, 54–67 (1997) 3

[39] Ehlers, R.: Minimising Deterministic Buechi Automata Precisely Using SAT Solving. In: Strichman, O., Szeider, S. (eds.) Theory and Applications of Satisfiability Testing (SAT 2010). LNCS, vol. 6175, pp. 326–332. Springer-Verlag (2010) 6

[40] Ehlers, R., Finkbeiner, B.: On the Virtue of Patience: Minimizing Buechi Automata. In: 17th International SPIN Workshop on Model Checking of Software (SPIN 2010). pp. 129–145. No. 6349 in LNCS, Springer Verlag (2010) 6

[41] Emerson, E.A., Jutla, C.S.: Tree automata, Mu-Calculus and determinacy. In: Foundations of Computer Science. pp. 368–377. SFCS '91, IEEE Computer Society, Washington, DC, USA (1991), http://dx.doi.org/10.1109/SFCS.1991.185392 222

[42] Etessami, K.: A Hierarchy of Polynomial-Time Computable Simulations for Automata. In: International Conference on Concurrency Theory. LNCS, vol. 2421, pp. 131–144. Springer-Verlag (2002), http://dx.doi.org/10.1007/3-540-45694-5_10 10, 11, 13, 36, 38, 39, 40, 41, 53, 121, 123, 130, 137, 158, 211, 223

[43] Etessami, K., Wilke, T., Schuller, R.: Fair Simulation Relations, Parity Games, and State Space Reduction for Buechi Automata. In: Orejas, F., Spirakis, P., van Leeuwen, J. (eds.) Automata, Languages and Programming, Lecture Notes in Computer Science, vol. 2076, pp. 694–707. Springer Berlin / Heidelberg (2001),

http://dx.doi.org/10.1007/3-540-48224-5_57 8, 9, 11, 27, 29, 31, 49, 56

[44] Etessami, K., Wilke, T., Schuller, R.A.: Fair Simulation Relations, Parity Games, and State Space Reduction for Büchi Automata. SIAM J. Comput. 34(5), 1159–1175 (2005), http://epubs.siam.org/sam-bin/dbq/article/42067 9, 94, 116, 121, 129

[45] Fagin, R., Stockmeyer, L., Vardi, M.Y.: On monadic NP vs. monadic co-NP. Information and Computation pp. 78–92 (Jul 1995), http://dx.doi.org/10.1109/SCT.1993.336544 8

[46] Filiot, E., Jin, N., Raskin, J.F.: An Antichain Algorithm for LTL Realizability. In: Computer Aided Verification. pp. 263–277. CAV '09, Springer-Verlag, Berlin, Heidelberg (2009), http://dx.doi.org/10.1007/978-3-642-02658-4_22 253

[47] Filiot, E., Jin, N., Raskin, J.F.: Antichains and compositional algorithms for LTL synthesis. Formal Methods in System Design 39, 261–296 (2011), http://dx.doi.org/10.1007/s10703-011-0115-3 253

[48] Finkelstein, A., Dowell, J.: A comedy of errors: the London Ambulance Service case study. In: Software Specification and Design. pp. 2–4 (Mar 1996), http://dx.doi.org/10.1109/IWSSD.1996.501141 2

[49] Flum, J., Grädel, E., Wilke, T.: Logic and automata: History and perspectives. Amsterdam University Press (2008) 256

[50] Fogarty, S.: Buechi Containment and Size-Change Termination. Master's thesis, Rice University (2008) 255

[51] Fogarty, S., Vardi, M.: Büchi Complementation and Size-Change Termination. In: Kowalewski, S., Philippou, A. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science, vol. 5505, pp. 16–30. Springer Berlin / Heidelberg (2009), http://dx.doi.org/10.1007/978-3-642-00768-2_2 5, 254, 255, 257, 258, 261

[52] Fogarty, S., Vardi, M.Y.: Efficient Büchi Universality Checking. In: Tools and Algorithms for the Construction and Analysis of Systems. pp. 205–220 (2010) 254, 255, 258, 261

[53] Friedmann, O., Lange, M.: Ramsey-Based Analysis of Parity Automata. In: Tools and Algorithms for the Construction and Analysis of Systems (2012) 255

[54] Fritz, C.: Constructing Buechi automata from linear temporal logic using simulation relations for alternating Buechi automata. In: Implementation and application of automata. pp. 35–48. CIAA'03, Springer-Verlag, Berlin, Heidelberg (2003), http://portal.acm.org/citation.cfm?id=1760237 125, 127, 249, 253

[55] Fritz, C., Wilke, T.: Simulation Relations for Alternating Büchi Automata. Theor. Comput. Sci. 338(1-3), 275–314 (2005), http://dx.doi.org/10.1016/j.tcs.2005.01.016 10, 11, 13, 31, 32, 33, 95, 96, 120, 121, 123, 126, 127, 129, 130, 143, 159, 178, 183, 195, 196, 197, 207, 208, 209, 213, 216, 217

[56] Ganty, P., Maquet, N., Raskin, J.F.: Fixed point guided abstraction refinement for alternating automata. Theor. Comput. Sci. 411, 3444–3459 (Aug 2010), http://dx.doi.org/10.1016/j.tcs.2010.05.037 253

[57] Gleick, J.: A bug and a crash - sometimes a bug is more than a nuisance. New York Times Magazine (Dec 1996), http://www.around.com/ariane.html 3

[58] Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games: A Guide to Current Research, Lecture Notes in Computer Science, vol. 2500. Springer (2002) 5, 116

[59] Gramlich, G., Schnitger, G.: Minimizing NFA's and Regular Expressions. Journal of Computer and System Sciences 73(6), 908–923 (2007), http://dx.doi.org/10.1016/j.jcss.2006.11.002 6

[60] Greenfield, A.: Everyware - The dawning age of ubiquitous computing. New Riders (Mar 2006) 2

[61] Gurevich, Y., Harrington, L.: Trees, Automata, and Games. In: Symposium on the Theory of Computing. pp. 60–65. ACM, New York, NY, USA (1982) 21

[62] Gurumurthy, S., Bloem, R., Somenzi, F.: Fair Simulation Minimization. In: Computer Aided Verification. LNCS, vol. 2404, pp. 610–624. Springer-Verlag (2002), http://portal.acm.org/citation.cfm?id=647771.734423 30

[63] Gurumurthy, S., Kupferman, O., Somenzi, F., Vardi, M.Y.: On complementing nondeterministic büchi automata. In: Correct Hardware Design and Verification Methods. Lecture Notes in Computer Science, vol. 2860, pp. 96–110 (2003) 124

[64] Halfhill, T.R.: The truth behind the Pentium bug. BYTE 20(3), 163–164 (1995) 3

[65] Harrison, J.: Handbook of Practical Logic and Automated Reasoning. Cambridge University Press (2009) 3

[66] Henzinger, T.A., Kupferman, O., Rajamani, S.K.: Fair Simulation. Information and Computation 173, 64–81 (2002), http://dx.doi.org/10.1006/inco.2001.3085 9, 26, 29

[67] Henzinger, T.A., Rajamani, S.K.: Fair Bisimulation. In: Tools and Algorithms for the Construction and Analysis of Systems. LNCS, vol. 1785, pp. 299–314. Springer-Verlag (2000), http://portal.acm.org/citation.cfm?id=646484.691752 9, 30

[68] Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley (1979) 251

[69] Hopcroft, J.E.: An n log n algorithm for minimizing states in a finite automaton. Tech. rep., Stanford University, Stanford, CA, USA (1971), http://portal.acm.org/citation.cfm?id=891883 6

[70] Jean-Pierre, Pecuchet: On the complementation of Buechi automata. Theoretical Computer Science 47(0), 95–98 (1986), http://www.sciencedirect.com/science/article/pii/0304397586901362 254

[71] Jiang, T., Ravikumar, B.: Minimal NFA Problems are Hard. In: Albert, J., Monien, B., Artalejo, M. (eds.) International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science, vol. 510, pp. 629–640. Springer Berlin / Heidelberg (1991), http://dx.doi.org/10.1007/3-540-54233-7_169 6

[72] Joel, Spencer: Threshold spectra via the Ehrenfeucht game. Discrete Applied Mathematics 30(2-3), 235–252 (1991), http://www.sciencedirect.com/science/article/pii/0166218X91900482 8

[73] Juvekar, S., Piterman, N.: Minimizing Generalized Buechi Automata. In: Computer Aided Verification. LNCS, vol. 4414, pp. 45–58. Springer-Verlag (2006), http://dx.doi.org/10.1007/11817963_7 10, 118

[74] Kameda, T., Weiner, P.: On the State Minimization of Nondeterministic Finite Automata. IEEE Trans. Comput. 19, 617–627 (Jul 1970), http://dx.doi.org/10.1109/T-C.1970.222994 6

[75] King, V., Kupferman, O., Vardi, M.Y.: On the Complexity of Parity Word Automata. In: Foundations of Software Science and Computation Structures. pp. 276–286. FoSSaCS '01, Springer-Verlag, London, UK (2001), http://portal.acm.org/citation.cfm?id=704696 228

[76] Klarlund, N.: Progress measures and finite arguments for infinite computations. Ph.D. thesis, Cornell University (1990) 252

[77] Klarlund, N.: Progress measures for complementation of omega-automata with applications to temporal logic. In: Foundations of computer science. pp. 358–367. SFCS '91, IEEE Computer Society, Washington, DC, USA (1991), http://dx.doi.org/10.1109/SFCS.1991.185391 252

[78] Knuth, D.E.: Two notes on notation. Am. Math. Monthly 99(5), 403–422 (May 1992), http://arxiv.org/abs/math/9205211 16

[79] Kupferman, O., Vardi, M.: Safraless decision procedures. In: Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on. pp. 531 – 540 (oct 2005) 124

[80] Kupferman, O., Vardi, M.: Verification of Fair Transition Systems. In: Computer Aided Verification, LNCS, vol. 1102, pp. 372–382. Springer-Verlag (1996), http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.29.9654 23, 55, 56

[81] Kupferman, O., Vardi, M.: Weak Alternating Automata Are Not That Weak. ACM Trans. Comput. Logic 2, 408–429 (Jul 2001), http://dx.doi.org/10.1145/377978.377993 57, 59, 63, 124, 252

[82] Lange, M.: Size-change termination and satisfiability for linear-time temporal logics. In: Frontiers of combining systems. pp. 28–39. FroCoS'11, Springer-

Verlag, Berlin, Heidelberg (2011), http://dl.acm.org/citation.cfm?id=2050784.2050788 255

[83] Lee, C.S., Jones, N.D., Ben Amram, A.M.: The size-change principle for program termination. SIGPLAN Not. 36, 81–92 (Jan 2001), http://dx.doi.org/10.1145/373243.360210 254, 255, 257

[84] Leveson, N.G., Turner, C.S.: An Investigation of the Therac-25 Accidents. Computer (IEEE) 26(7), 18–41 (Jul 1993) 2

[85] Löding, C.: Efficient minimization of deterministic weak omega-automata. Information Processing Letters 79, 105–109 (2001) 6

[86] Lynch, N.A., Vaandrager, F.W.: Forward and Backward Simulations. Part I: Untimed Systems. Information and Computation 121(2), 214–233 (1995), http://dx.doi.org/10.1006/inco.1995.1134 21, 39

[87] Manna, Z., Pnueli, A.: A hierarchy of temporal properties (invited paper, 1989). In: Principles of Distributed Computing. pp. 377–410. PODC '90, ACM, New York, NY, USA (1990), http://doi.acm.org/10.1145/93385.93442 221

[88] Martin, D.: Borel Determinacy. Ann. Math. 102, 363–371 (1975) 19, 22

[89] Meyer, A.R., Stockmeyer, L.J.: The equivalence problem for regular expressions with squaring requires exponential space. In: Switching and Automata Theory. pp. 125–129. IEEE Computer Society, Washington, DC, USA (1972), http://portal.acm.org/citation.cfm?id=1438639 233

[90] Michel, M.: Complementation is more difficult with automata on infinite words 252

[91] Milner, R.: An algebraic definition of simulation between programs. In: International joint conference on artificial intelligence. pp. 481–489. IJCAI'71, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1971), http://portal.acm.org/citation.cfm?id=1622926 8

[92] Miyano, S., Hayashi, T.: Alternating Finite Automata on omega-Words. Theoretical Computer Science 32, 321–330 (1984) 121, 124, 177, 178, 179

[93] Nicely, T.R.: Enumeration to 10Î4 of the twin primes and Brun's constant. Virginia Journal of Science 46, 195–204 (1996) 3

[94] Nori, S.: How many engineers does it take to change a lightbulb? http://www.boiledbeans.net/2009/04/20/how-many-engineers-does-it-take-to-change-a-lightbulb/ (Apr 2009) 3

[95] Perrin, D., Pin, J.E.: Infinite Words - Automata, Semigroups, Logic and Games, Pure and Applied Mathematics, vol. 141. Elsevier (2004) 256, 259

[96] Pnueli, A.: The temporal logic of programs. In: Foundations of Computer Science. SFCS '77, vol. 0, pp. 46–57. IEEE Computer Society, Washington, DC, USA (Oct 1977), http://dx.doi.org/10.1109/SFCS.1977.32 3

[97] Queille, J.P., Sifakis, J.: Specification and verification of concurrent systems in CAESAR. In: International Symposium on Programming. pp. 337–350 (1981) 4

[98] Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM J. Res. Dev. 3, 114–125 (Apr 1959), http://dx.doi.org/10.1147/rd.32.0114 4, 251

[99] Raimi, R.S.: Environment Modeling and Efficient State Reachability Checking. Ph.D. thesis, The University of Texas at Austin (1999) 9, 28, 251

[100] Raimi, R., Hojati, R., Namjoshi, K.S.: Environment modeling and language universality. ACM Trans. Des. Autom. Electron. Syst. 5, 705–725 (Jul 2000), http://doi.acm.org/10.1145/348019.348572 251

[101] Ramsey, F.P.: On a Problem of Formal Logic. Proceedings of the London Mathematical Society s2-30(1), 264–286 (1930), http://plms.oxfordjournals.org/content/s2-30/1/264.short 254

[102] Raskin, J.F., Chatterjee, K., Doyen, L., Henzinger, T.A.: Algorithms for Omega-Regular Games with Imperfect Information. Logical Methods in Computer Science 3(3), 1–23 (2007) 251

[103] Rozier, K.Y., Vardi, M.Y.: LTL satisfiability checking. In: Model checking software. pp. 149–167. SPIN'07, Springer-Verlag, Berlin, Heidelberg (2007), http://portal.acm.org/citation.cfm?id=1770548 5

[104] Safra, S.: On the complexity of omega-automata. In: Foundations of Computer Science. vol. 0, pp. 319–327. IEEE Computer Society, Los Alamitos, CA, USA (1988), http://dx.doi.org/http://doi.ieeecomputersociety.org/10.1109/SFCS.1988.21948 220

[105] Savitch, W.J.: Relationships between nondeterministic and deterministic tape complexities. Journal of Computer and System Sciences 4(2), 177–192 (1970), http://www.sciencedirect.com/science/article/pii/S002200007080006X 228

[106] Schewe, S.: Buechi Complementation Made Tight. In: Albers, S., Marion, J.Y. (eds.) Theoretical Aspects of Computer Science. Leibniz International Proceedings in Informatics (LIPIcs), vol. 3, pp. 661–672. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2009), http://drops.dagstuhl.de/opus/volltexte/2009/1854 124, 251

[107] Schewe, S.: Beyond Hyper-Minimisation—Minimising DBAs and DPAs is NP-Complete. In: Lodaya, K., Mahajan, M. (eds.) Foundations of Software Technology and Theoretical Computer Science. LIPIcs, vol. 8, pp. 400–411. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2010), http://drops.dagstuhl.de/opus/volltexte/2010/2881 6

[108] Sebastiani, R., Tonetta, S.: More Deterministic vs. Smaller Büchi Automata for Efficient LTL Model Checking. In: Geist, D., Tronci, E. (eds.) Correct Hardware Design and Verification Methods, Lecture Notes in Computer Science, vol. 2860, pp. 126–140. Springer Berlin / Heidelberg (2003), http://dx.doi.org/10.1007/978-3-540-39724-3_12 6

[109] Sistla, A.P., Vardi, M.Y., Wolper, P.: The complementation problem for Bu¨chi automata with applications to temporal logic. Theor. Comput. Sci. 49, 217–237 (Jan 1987), http://dx.doi.org/10.1016/0304-3975(87)90008-9 254

[110] Somenzi, F., Bloem, R.: Efficient Buechi Automata from LTL Formulae. In: International Conference on Concurrency Theory, LNCS, vol. 1855, pp. 248–263. Springer-Verlag (2000), http://dx.doi.org/10.1007/10722167_21 10, 28, 29, 31

[111] Stockmeyer, L.J., Meyer, A.R.: Word Problems Requiring Exponential Time (Preliminary Report). In: Symposium on the Theory of Computing. pp. 1–9. STOC '73, ACM, New York, NY, USA (1973), http://dx.doi.org/10.1145/800125.804029 5, 37

[112] Tozawa, A., Hagiya, M.: XML schema containment checking based on semi-implicit techniques. In: Implementation and application of automata. pp. 213–225. CIAA'03, Springer-Verlag, Berlin, Heidelberg (2003), http://dl.acm.org/citation.cfm?id=1760230.1760252 251

[113] Tsai, M.H., Fogarty, S., Vardi, M.Y., Tsay, Y.K.: State of Buechi Complementation (Full Version). In: Proceedings of the 15th international conference on Implementation and application of automata. pp. 261–271. CIAA'10, Springer-Verlag, Berlin, Heidelberg (Jun 2010) 124

[114] Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification. In: Logic in Computer Science. pp. 332–344 (Jun 1986) 4, 5

[115] Vardi, M.: Alternating Automata and Program Verification. In: Computer Science Today, LNCS, vol. 1000, pp. 471–485. Springer-Verlag (1995), http://dx.doi.org/10.1007/BFb0015261 55, 124, 125, 126

[116] Vardi, M.Y.: Branching vs. linear time: Final showdown. In: Tools and Algorithms for the Construction and Analysis of Systems. Lecture Notes in Computer Science, vol. 2031, pp. 1–22 (2001) 4

[117] Vardi, M.Y.: The Buechi complementation saga. In: Theoretical aspects of computer science. pp. 12–22. STACS'07, Springer-Verlag, Berlin, Heidelberg (2007), http://portal.acm.org/citation.cfm?id=1763427 124, 251

[118] Vardi, M.Y., Wolper, P.: Reasoning about infinite computations. Inf. Comput. 115(1), 1–37 (Nov 1994), http://dx.doi.org/10.1006/inco.1994.1092 5

[119] Vardi, M.Y., Wolper, P.: Reasoning about infinite computations. Information and Computation 115(1), 1–37 (Nov 1994), http://dx.doi.org/10.1006/inco.1994.1092 228

[120] Wang, H.: Proving theorems by pattern recognition II. Bell System Tech. Journal 40, 1–42 (1961) 232

[121] Wolfe, A.: Intel fixes A Pentium FPU glitch. Electronic Engineering Times (Nov 1994) 3

[122] Wulf, M.D., Doyen, L., Henzinger, T.A., Raskin, J.F.: Antichains: A New Algorithm for Checking Universality of Finite Automata. In: Ball, T., Jones, R.B. (eds.) Computer Aided Verification. LNCS, vol. 4144, pp. 17–30. Springer (2006), http://dblp.uni-trier.de/db/conf/cav/cav2006.html#WulfDHR06 5, 251, 252

Index

ABA, *see* alternating Büchi automaton
alternating Büchi automaton, 19
automata minimization, 5
automata theory, 4

coherent sequences, 84
containment, 40

fixed-word simulation, 43
  complexity, 52
  multipebble, 54
    collapse, 56
formal methods, 2

games, 16
GFQ, *see* Good for Quotienting
Good for Quotienting, 22

jumping simulation, 72
  computing, 111
  hierarchies, 94
  jumping-safety preservation, 88
  non-decreasingness, 77
  reflexivity, 87
  vs. mediated preorder, 104
jumping transition, path, containment, and language, 71
jumping-safe preorder, 72
  preservation, 88

language
  containment, 20
  inclusion, 20
    Ramsey-based approach, 240

mediated preorder, 104
model-checking, 3
multipath, 18
multipebble simulation
  complexity, 214
  computing, 208
  fixed-word, 54
  for ABAs, 121
  for NBAs, 35
  hierarchy, 127
  uniform games, 136

NBA, *see* nondeterministic Büchi automaton
nondeterministic Büchi automaton, 18

quotient
  existential, 22
  Good for Quotienting, 22
  max, 31
  minimax, 186
  naïve, 21
  semielective, 197
  universal, 22

Ramsey-based inclusion checking, 240

simulation
  desiderata, 8
  fixed-word, 43
  for Büchi automata, 9
  jumping, 72
  multipebble, 35
  multipebble fixed-word, 54
simulation preorder, 7
  desiderata, 8
  for ABAs, 118
  for NBAs, 22
  quotienting with, 28
  simplifying transitions with, 29
subset constructions
  fair subset construction, 175
  Miyano-Hayashi construction, 168
subsumption methods, 237

undordered cartesian product, 14
uniform simulation games, 136