Foundations of computing Volume 9

Christoph Dittmann

Parity Games, Separations, and the Modal μ-Calculus

Universitätsverlag der TU Berlin

Technische Universität Berlin

Christoph Dittmann

Parity Games, Separations, and the Modal μ-Calculus

Die Schriftenreihe *Foundations of computing*
der Technischen Universität Berlin wird
herausgegeben von:
Prof. Dr. Rolf Niedermeier,
Prof. Dr. Uwe Nestmann,
Prof. Dr. Stephan Kreutzer

Foundations of computing | 9

Christoph Dittmann

Parity Games, Separations, and the Modal μ-Calculus

Universitätsverlag der TU Berlin

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.dnb.de abrufbar.

Universitätsverlag der TU Berlin, 2017
http://verlag.tu-berlin.de

Fasanenstr. 88, 10623 Berlin
Tel.: +49 (0)30 314 76131 / Fax: -76133
E-Mail: publikationen@ub.tu-berlin.de

Zugl.: Berlin, Techn. Univ., Diss., 2017
1. Gutachter: Prof. Dr. Stephan Kreutzer
2. Gutachter: Prof. Dr. Erich Grädel
3. Gutachter: Prof. Dr. Uwe Nestmann
Die Arbeit wurde am 20. Januar 2017 an der Fakultät IV unter Vorsitz von Prof. Dr. Rolf Niedermeier erfolgreich verteidigt.

Druck: docupoint GmbH
Satz/Layout: Christoph Dittmann

Umschlagfoto:
Jaro Larnos | https://www.flickr.com/photos/jlarnos/8297692520/
CC BY 2.0 | https://creativecommons.org/licenses/by/2.0/

ISBN 978-3-7983-2887-7 (print)
ISBN 978-3-7983-2888-4 (online)

ISSN 2199-5249 (print)
ISSN 2199-5257 (online)

Zugleich online veröffentlicht auf dem institutionellen Repositorium der Technischen Universität Berlin:
DOI 10.14279/depositonce-5700
http://dx.doi.org/10.14279/depositonce-5700

Abstract

The topics of this thesis are the modal μ-calculus and parity games. The modal μ-calculus is a common logic for model-checking in computer science. The model-checking problem of the modal μ-calculus is polynomial time equivalent to solving parity games, a 2-player game on labeled directed graphs.

We present the first FPT algorithms (fixed-parameter tractable) for the model-checking problem of the modal μ-calculus on restricted classes of graphs, specifically on classes of bounded Kelly-width or bounded DAG-width. In this process we also prove a general decomposition theorem for the modal μ-calculus and define a useful notion of *type* for this logic.

Then, assuming a class of parity games has a polynomial time algorithm solving it, we consider the problem of extending this algorithm to larger classes of parity games. In particular, we show that joining games, pasting games, or adding single vertices preserves polynomial-time solvability. It follows that parity games can be solved in polynomial time if their underlying undirected graph is a tournament, a complete bipartite graph, or a block graph.

In the last chapter we present the first non-trivial *formal* proof about parity games. We explain a formal proof of positional determinacy of parity games in the proof assistant Isabelle/HOL.

Zusammenfassung

Die Themen dieser Dissertation sind der modale μ-Kalkül und Paritätsspiele. Der modale μ-Kalkül ist eine häufig eingesetzte Logik im Bereich des Model-Checkings in der Informatik. Das Model-Checking-Problem des modalen μ-Kalküls ist polynomialzeitäquivalent zum Lösen von Paritätsspielen, einem 2-Spielerspiel auf beschrifteten, gerichteten Graphen.

Wir präsentieren die ersten FPT-Algorithmen (fixed-parameter tractable) für das Model-Checking-Problem des modalen μ-Kalküls auf Klassen von Graphen mit beschränkter Kelly-Weite oder beschränkter DAG-Weite. Für diesen Zweck beweisen wir einen allgemeineren Zerlegungssatz für den modalen μ-Kalkül und stellen eine nützliche Definition von *Typen* für diese Logik vor.

Angenommen, eine Klasse von Paritätsspielen hat einen Polynomialzeit-Lösungs-Algorithmus, betrachten wir danach das Problem, diese Klassen zu erweitern auf eine Weise, sodass Polynomialzeit-Lösbarkeit erhalten bleibt. Wir zeigen, dass dies beim Join von Paritätsspielen, beim Pasting und beim Hinzufügen einzelner Knoten der Fall ist. Wir folgern daraus, dass das Lösen von Paritätsspielen in Polynomialzeit möglich ist, falls der unterliegende ungerichtete Graph ein Tournament, ein vollständiger bipartiter Graph oder ein Blockgraph ist.

Im letzten Kapitel präsentieren wir den ersten nichttrivialen *formalen* Beweis über Paritätsspiele. Wir stellen einen formalen Beweis für die positionale Determiniertheit von Paritätsspielen im Beweis-Assistenten Isabelle/HOL vor.

Contents

List of Figures

1 Introduction

Model checking is one of the most important topics in computer science and a successful application of methods of formal mathematics in practice. Introduced in the 1980's, nowadays this method is widely used to express and verify properties of programs and of protocols [EC80; QS82; Hol93; BK08]. Modern examples of implementations of model checking include the SPIN model checker and the NuSMV model checker [Spi; Nus]. Formally, model checking is the problem of deciding whether a given logical formula is true on a given mathematical structure.

The collection of all possible formulas and their semantics is called a *mathematical logic*. Finding a good logic is key to an effective and efficient model checking procedure. The logic needs to be expressive enough for the properties of interest but simple enough so that model checking remains feasible. A logic that seems to strike a very nice balance here is the *modal μ-calculus*, introduced by Dexter Kozen [Koz83]. It is a highly expressive logic, more expressive than the linear and branching temporal logics PDL [Koz83], LTL and CTL [EL86], CTL* and ECTL* [Dam94; BC96; CGR11]. At the same time the model-checking problem of the modal μ-calculus is easy enough to be feasible in practice.

The modal μ-calculus makes statements about individual vertices in a directed graph, for example "from this vertex a red vertex is reachable" or "from this vertex an infinite

walk along the arcs is possible", to name only a few. On the other hand, the modal μ-calculus is *bisimulation-invariant*, an equivalence relation between models (see Section 3.1.6). Bisimulation invariance implies that the modal μ-calculus is unable to express properties such as "this vertex has a loop" or "this vertex has degree 3".

Formally, the modal μ-calculus is an extension of *basic modal logic* with fixed points (see Section 3.1). Basic modal logic is a mathematical logic that can express statements such as "something is possible" or "something is necessary". The usual interpretation of "possible" and "necessary" is based on the edge relation on directed graphs. Some statement is *possible* on a vertex if there exists a successor where it is true. Conversely, a statement is *necessary* if it is true on all successors.

Basic modal logic is contained in first-order logic because we can readily reason about all successors if we have a universal quantifier. But it is easy to see that the modal μ-calculus is not contained in first-order logic because it follows from locality arguments that first-order logic is unable to express "from this vertex a red vertex is reachable".

However, "this vertex has a loop" is trivial to express in first-order logic. This shows that the modal μ-calculus is incomparable to first-order logic. In some aspects it is stronger and in some aspects it is weaker.

The model-checking problem for basic modal logic is a very easy task and trivially possible in polynomial time. For most other logics, however, the model-checking problem quickly becomes too hard in the general case. For example, the problem of evaluating quantified Boolean formulas is the canonical PSPACE-complete problem (see e.g., [AB09]). Clearly, a quantified Boolean formula is nothing more than a special case of a

first-order formula evaluated over a 2-element model containing "true" and "false".

In practice the formula, in some logic, often turns out to be small compared to the model in the the model-checking problem. So instead of looking at the general model-checking problem, we consider it as a *parameterized problem* where we define a *parameter*, in this case the formula. Then we look for *parameterized algorithms* that are fast when the parameter is fixed. For a fixed formula, the model-checking problem of the modal μ-calculus is trivially polynomial time solvable with a simple brute force algorithm. However, the degree of this polynomial depends on the formula.

For more efficient algorithms, we would like to limit the degree of the polynomial in the running time. We say that a parameterized problem is *fixed-parameter tractable* (FPT) if it becomes polynomial-time solvable for every fixed choice of the parameter and the degree of the polynomial is independent of the parameter (see Section 4.1.3 for the formal definition).

To put this work into the wider context, let us remark that very recently, Martin Grohe, Stephan Kreutzer and Sebastian Siebertz showed that first-order model-checking with the formula as the parameter is FPT on certain sparse classes of graphs [GKS14]. Their result is also optimal, because Anuj Dawar and Stephan Kreutzer showed that it is intractable on all larger "effective" classes that are closed under taking subgraphs, under the common complexity theoretic assumption FPT $\neq$ AW[$*$] [KD09, Theorem 6.1]. Zdeněk Dvořák, Daniel Král' and Robin Thomas later showed that it is intractable on all larger classes closed under taking subgraphs under the assumption that FPT $\neq$ W[1] [DKT13, Theorem 1.5].

Let us now move to the model-checking problem of the

modal μ-calculus, the main focus of this thesis. As mentioned above, the modal μ-calculus is not contained in first-order logic, so the result by Grohe et. al does not apply to this case.

The μ-calculus model-checking problem has an intimate relationship to 2-player games called *parity games*. A parity game is played on a directed graph with 2 players called "even" ($\Diamond$) and "odd" ($\Box$) who push a single token over the vertices along the arcs. The vertices are labeled with natural numbers called their *priorities*, and the total number of distinct priorities is finite. Each vertex is also labeled with the player who must move the token when it is on this vertex. If a player is unable to move because a vertex has no successors, then he loses.

The game continues for as long as possible, usually an infinite number of turns. Because we have finitely many distinct priorities, at least one of the priorities will occur infinitely often. We then look at the smallest priority p occurring infinitely often. If p is even, then player $\Diamond$ wins, and if it is odd, then player $\Box$ wins (see Section 3.2 for the formal definition).

Determining the winner of a parity game is polynomial-time equivalent to the μ-calculus model-checking problem. So instead of investigating the model-checking problem directly, in many cases it is both equivalent and easier to work with parity games.

It is known that the complexity of solving parity games is in $\mathrm{NP} \cap \mathrm{coNP}$. There exists a slightly better result by Marcin Jurdziński showing that the problem is in $\mathrm{UP} \cap \mathrm{coUP}$ [Jur98]. UP is the class of all problems that can be solved by a non-deterministic polynomial-time Turing machine with at most one accepting path. It follows that $\mathrm{P} \subseteq \mathrm{UP} \subseteq \mathrm{NP}$, and it is unknown if any of these inclusions are strict.

Membership to $\mathrm{NP} \cap \mathrm{coNP}$ already makes the problem of

solving parity games unlikely to be NP-complete because then we would have a proof of NP = coNP. Another indication for this is that Marcin Jurdziński, Mike Paterson and Uri Zwick found a subexponential algorithm [JPZ06]. As of 2016 it is still open whether solving parity games and thus μ-calculus model checking have polynomial-time algorithms.

What makes this complexity so intriguing is that it is extremely rare for a problem to be in NP∩coNP with an unknown membership to P. Other such problems are integer factorization (but not primality testing, which is in P [AKS04]), stochastic games [Con92] and lattice problems [AR05] (approximating the shortest and closest vector in a lattice to within a factor of $\sqrt{n}$). If we also allow the class coAM instead of coNP, which, as stated by László Babai and Shlomo Moran, is "just above coNP", then we can extend this list with graph isomorphism and matrix group membership [BM88; Bab92]. Matrix group membership is the problem, given invertible matrices $A, B_1, \ldots, B_k$ over a finite field, of deciding whether A is in the group generated by $B_1, \ldots, B_k$.

As far as the author is aware, these are all of the common NP ∩ coNP-problems (NP ∩ coAM, respectively) with unknown membership to P. Usually problems in NP ∩ coNP turn out to be in P. This includes problems with solutions that are far from obvious, for example primality testing and linear programming, both of which have polynomial-time algorithms.

From the point of view of parameterized complexity, the usual parameter for parity games is the number of distinct priorities. With this parameter, it is FPT equivalent to the μ-calculus model-checking problem with the formula as the parameter. However, if a graph width measure is included in the parameter, then this width usually becomes unbounded

in the reduction from the model-checking problem to parity games. This does not happen in the other direction, where the structure for the model-checking problem will essentially be identical to the parity game. In this sense, FPT results for the model-checking problem are more general than FPT results for parity games.

In this thesis we work with the aim in mind of proving that the μ-calculus model-checking problem parameterized with the formula is in FPT. We do not solve this problem in its full generality, but we consider restricted classes of graphs, specifically bounded Kelly-width and bounded DAG-width, and prove fixed-parameter tractability of the μ-calculus model-checking problem on these classes with the formula and the width of the decomposition as the parameter. For the DAG-width we also need to include the size of the decomposition in the running time (but not in the parameter) because a DAG decomposition can have exponential size compared to the input graph [AKR14].

In this process we prove a more general decomposition theorem for the modal μ-calculus, interesting in its own right. Our theorem is similar in nature to the well-known theorems by Solomon Feferman and Robert L. Vaught from classical model theory [FV59]. Their classical and our new result are statements about the *type* of a vertex, that is, the set of formulas that are true at that vertex. The classical result says that with certain constructions such as disjoint unions and products, the type of a vertex in the resulting graph depends only on the types occurring in the factors and not on the factors themselves.

This theorem is very useful for algorithms because it allows dynamic programming, which is a well-known technique to solve a complicated problem by splitting the input into simpler

substructures, solving these subproblems, and finally combining the results. Johann A. Makowsky provides a good survey over the algorithmic applications of the Feferman-Vaught Theorem [Mak04].

In Chapter 5 we show that if a model has the form of a *directed separation*, that is, if it consists of two submodels $\mathcal{M}_1$ and $\mathcal{M}_2$ intersecting only in a small set X of vertices and with no arcs pointing from $\mathcal{M}_2 \setminus X$ to $\mathcal{M}_1 \setminus X$, then the modal μ-types of $\mathcal{M}_1$ can be computed essentially from $\mathcal{M}_1$ and from the types occurring in $\mathcal{M}_2$. While the result talks only about the μ-calculus, we use parity games extensively throughout the proof.

While in Chapter 5 we are mostly concerned with solving the model-checking problem directly, in Chapter 6 we consider the problem of solving parity games. We look at *graph operations* that preserve polynomial-time solvability. More specifically, if a class of parity games is solvable with a polynomial-time algorithm, then some larger classes are also solvable in polynomial-time, for example the class of graphs with one additional vertex per game or the class of graphs consisting of so-called *joins* of two games.

As an application, these results imply that parity games can be solved in polynomial time if their underlying undirected graph is an orientation of a complete graph, such as tournaments, or a complete bipartite graph, or a block graph, or a cactus graph. A graph is a block graph if every biconnected component is a clique, and it is a cactus graph if every edge lies on at most one cycle.

Another aspect of parity games that we are going to look at in Chapter 7 is the problem of writing formal proofs about them. Usually, people write mathematical proofs in a mix

of English, formulas, and pictures. We call this an *informal proof*. On a certain level, this is unsatisfying because humans are fallible and mistakes are bound to happen. And indeed mistakes happen, as evidenced for example by the many failed attempts of proving Fermat's Last Theorem or the numerous unsuccessful attempts of proving or refuting NP = P.

In recent years, it has become possible to use computers to verify mathematical proofs by writing a *formal proof* with a *proof assistant*. A proof assistant is a software that implements a calculus, allowing the user to check a mathematical proof for formal correctness in a given system of axioms and inference rules. Usually, a proof assistant also supports the user in finding a formal proof, for example by searching for parts of the proof automatically or by proposing lemmas that could be helpful in the current context.

However, presently proof assistants cannot automatically prove most statements that require insight. In particular, an informal proof may use an innocent phrase such as "it is easy to see that". Such a phrase can turn into hundreds of lines of formal proof because for a proof assistant it is not at all easy to see that the statement is true. Often, this implies that the statement indeed requires a non-trivial proof within the given system of axioms and inference rules, and that the informal proof arguably glossed over these details, fingers crossed that it will work out. Thus formalizing an informal proof may expose gaps in the proof that would otherwise go unnoticed.

Consequently, compared to a proof in English a formal proof gives an extremely high confidence in the truth of a statement. Additionally, every new formal proof advances the proof assistant because it grows the database used for the automatic proof search.

In Chapter 7, we present a formal proof about an aspect of parity games, their *positional determinacy*. A positional strategy is a strategy for a player where he only looks at the current state of the game and ignores the history of how the game got to that state. One fundamental property of parity games is that they are positionally determined (see e.g., [Zie98]). This means that if one player has no positional winning strategy from a given vertex, then the other player has one. The proof is non-trivial and uses transfinite induction to capture arbitrarily large parity games.

We formally prove in the widely used proof assistant Isabelle/HOL that parity games are positionally determined and thus (to our knowledge) we provide the first formalization and the first non-trivial formal result for parity games.

The thesis is organized as follows.

- In Chapter 2, we give a few basic definitions such as graphs and digraphs.
- In Chapter 3, we define the modal μ-calculus and parity games, and explain their relationship.
- In Chapter 4, we give an overview of what is known about the complexity of solving parity games and model-checking problem of the modal μ-calculus, and prove hardness results.
- In Chapter 5, we consider the μ-calculus model-checking problem on restricted classes of graphs (bounded DAG-width and bounded Kelly-width), prove a decomposition theorem for L_μ and provide fast algorithms for the restricted classes.

- In Chapter 6, we look at graph operations, such as adding a single vertex, that we can apply to parity games while preserving polynomial-time solvability.

- In Chapter 7, we explain a formalization of parity games and a formal proof of their positional determinacy, as well as first steps towards a formalization of tree decompositions.

The results of Chapter 5 have been published in [BDK14] and are joint work with Mikołaj Bojańczyk and Stephan Kreutzer. The results of Chapter 6 have been published in [DKT16] and are joint work with Stephan Kreutzer and Alexandru Tomescu. The results of Chapter 7 have been published in [Dit15; Dit16].

2 Preliminaries

We use standard notation for functions and graphs. See for example the books by Reinhard Diestel and by Jørgen Bang-Jensen and Gregory Gutin for good introductions [Die10; BJG09]. First, let us define some basic notation we are going to use.

We write $\mathbb{N} = \{0, 1, 2, \ldots\}$ for the set of natural numbers. A *partial function* or *function* $f : X \to Y$ is a set $f \subseteq X \times Y$ with the property that for every $x \in X$ there is at most one $y \in Y$ with $(x, y) \in f$. We will denote this unique y, if it exists, as $f(x)$.

partial function

For elements $x \in X, y \in Y$, we write $f[x \mapsto y]$ for the function

$f[x \mapsto y]$

$$f[x \mapsto y](z) = \begin{cases} y & \text{if } z = x \\ f(z) & \text{otherwise.} \end{cases}$$

The *domain* of f is the set $\mathrm{dom}(f) := \{x \in X \mid \exists y \in Y.\ (x, y) \in f\}$. The function f is *total* if $\mathrm{dom}(f) = X$. All functions in this thesis are total unless noted otherwise.

domain

total function

If $f : X \to Y$ is a partial function and $A \subseteq \mathrm{dom}(f)$, then we write $f(A) := \{y \in Y \mid \exists x \in A.\ f(x) = y\}$ for the *image* of A under f. For $B \subseteq Y$ we write $f^{-1}(B) := \{x \in \mathrm{dom}(f) \mid f(x) \in B\}$ for the *preimage* of B under f. For a single element $y \in Y$, we write $f^{-1}(y)$ instead of $f^{-1}(\{y\})$.

image

preimage

graph
vertices

Definition 2.1 An *undirected graph* G is a pair (V, E) where V is a set of *vertices* and

$$E \subseteq \{\{u, v\} \mid u, v \in V, u \neq v\}$$

edges

is a set of *edges*. We write $V(G)$ and $E(G)$ for the set of vertices and edges of G, respectively, if there is any chance of ambiguity. ⊣

complete graph

A few special undirected graphs are K_n, the *complete graph* with n vertices and $K_{n,m}$, the *complete bipartite graph* with $n + m$ vertices, n vertices in one part and m vertices in the other part of the partition.

digraph
arcs

Definition 2.2 A *directed graph (digraph)* G is a pair (V, E) where V is a set of *vertices* and $E \subseteq V \times V$ is a set of *arcs*. ⊣

loops

Note that in contrast to undirected graphs, directed graphs may have *loops*, that is, arcs of the form (v, v) for some $v \in V$. Neither undirected nor directed graphs have parallel edges or parallel arcs.

$u \to v$

For a directed graph G, we sometimes write $u \to v$ to say $(u, v) \in E(G)$ if G is clear from the context.

$G[X]$

For $X \subseteq V(G)$, we write $G[X]$ for the subgraph induced by X. This notation is the same for undirected and for directed graphs.

When the context makes it clear whether we have an undirected or directed graph, we usually simplify the notation and only say "graph". Graphs may be infinite unless otherwise noted. In particular, whenever algorithms are involved, the graphs are finite.

Definition 2.3 A *path* in a graph or a digraph G is a possibly infinite sequence $v_1, v_2, \ldots$ of vertices such that $v_i \to v_{i+1}$ holds for all pairs of consecutive vertices in the sequence. ⊣ path

Note that a path may have repeated vertices. To simplify the definition we also allow the empty path, but this will be of no importance because we always work with paths starting in a given vertex.

We often denote finite or infinite sequences of vertices such as $(v_1, v_2, \ldots)$ as $\overline{v}$ and write $v \in \overline{v}$ to say that $v \in \{v_1, v_2, \ldots\}$. $\overline{v}$

3 The Modal μ-Calculus and Parity Games

In this chapter we first define the logic that we are going to use throughout this thesis, the modal μ-calculus. In Section 3.2, we define parity games, a 2-player game on labeled directed graphs, and in Section 3.3 we explain their connection to the modal μ-calculus in terms of the model-checking game.

If you are already familiar with the modal μ-calculus and parity games, you may safely skip most of this chapter except a few sections where we introduce new concepts or deviate from the usual definitions in preparation for Chapter 5. These important sections are

- Section 3.1.7 on pages 34–38, where we define the subformulas and the closure of a formula,
- Section 3.1.8 on pages 38–47, where we give various notions of the depth of a formula, and
- Section 3.3 on pages 58–69, where we define annotated formulas and the model-checking game in a slightly different, but equivalent, way than usual.

3.1 The Modal μ-Calculus

The modal μ-calculus was introduced by Dexter Kozen in 1983 [Koz83]. Based on standard modal logic, this logic is extended with fixed points and makes statements about labeled directed graphs. In the following we will usually abbreviate "fixed point" as "fixpoint".

3.1.1 Syntax

We use a common definition of the modal μ-calculus L_μ as presented in the comprehensive survey by Julian Bradfield and Colin Stirling [BS07]. Let us revisit these definitions.

signature **Definition 3.1** A *signature* σ is a finite or infinite set. We call the elements of σ *proposition symbols* or *atomic propositions*..⊣

Var Let us fix a countably infinite set Var. We call the elements of Var *fixpoint variables* and usually denote them with the capital letters $X, Y, \ldots$.

Definition 3.2 Let σ be a signature. We define the syntax of $L_\mu[\sigma]$ the modal μ-calculus $L_\mu[\sigma]$ inductively as follows.

- $\top$ ("top") $\in L_\mu[\sigma]$, $\bot$ ("bottom") $\in L_\mu[\sigma]$.
- For all $P \in \sigma$, $P \in L_\mu[\sigma]$.
- For all $X \in \text{Var}$, $X \in L_\mu[\sigma]$.
- For all $\varphi, \psi \in L_\mu[\sigma]$, $(\varphi \wedge \psi), (\varphi \vee \psi) \in L_\mu[\sigma]$.
- For all $\varphi \in L_\mu[\sigma]$, $\Box\varphi$ ("box φ"), $\Diamond\varphi$ ("diamond φ"), $\neg\varphi \in L_\mu[\sigma]$.

- For all $\varphi \in L_\mu[\sigma]$ and $X \in \mathrm{Var}$ such that X occurs only positively in φ, that is, in the scope of an even number of negations, $\mu X.\varphi, \nu X.\varphi \in L_\mu[\sigma]$. ⊣

We write L_μ instead of $L_\mu[\sigma]$ if the signature is clear from the context. We call a formula of the form $\top$, $\bot$, P, or X for some $P \in \sigma$, $X \in \mathrm{Var}$ an *atomic formula*.

atomic formula

The operators μ, ν are called *fixpoint operators*. We say that a fixpoint operator $\mu X.\varphi$ (or $\nu X.\varphi$) *binds* X *in* φ. A variable not bound by an enclosing fixpoint operator is *free*.

μ, ν
bound, free

The operators $\Box, \Diamond$ are the *modal operators*. We call the subset of L_μ without fixpoint operators, that is, the subset with only the modal operators and the Boolean operators, *basic modal logic*.

modal operators
basic modal logic

The *length of* φ, denoted as $|\varphi|$, is the number of symbols needed for writing out φ syntactically.

$|\varphi|$

We omit parentheses when there is no confusion. Note that already by the definition of the syntax, the unary operators $\neg, \Box, \Diamond$ have higher priority than the binary operators $\vee$ and $\wedge$. If we omit parentheses, then a fixpoint operator's scope always extends as far to the right as possible while the scope of $\neg, \Box, \Diamond$ extends as little to the right as possible. For example,

$$\mu X.\Diamond P \wedge \Box X = \mu X.\Big(\Diamond P \wedge \Box X\Big),$$

with the diamond $\Diamond$ applying only to P. We write equality $(=)$ here because we consider both sides of the equality sign to be denotations of syntactically the same formula.

We also introduce as abbreviations

$$(\varphi \to \psi) := \neg\varphi \vee \psi$$
$$(\varphi \leftrightarrow \psi) := (\varphi \to \psi) \wedge (\psi \to \varphi).$$

When it comes to omitting parentheses, both arrows have lower priority than $\vee, \wedge, \neg, \Box, \Diamond$ but higher priority than the fixpoint operators. This preserves the property that a fixpoint operator's extends as far to the right as possible and the scope $\neg, \Box, \Diamond$ extends as little to the right as possible. For example,

$$\mu X.Q \to \Diamond P \wedge \Box X = \mu X.\Big(Q \to (\Diamond P \wedge \Box X)\Big).$$

If we write a formula as $\varphi(X)$, the notation $\varphi(\psi)$ means that all free occurrences of X are substituted by ψ. An alternative way of writing this, which we will also use, is $\varphi[X/\psi]$. This is more precise in the case that φ also has free variables other than X.

3.1.2 Semantics

Let σ be a signature. The semantics of $L_\mu[\sigma]$ is defined on *σ-structures*, also known as *labeled transition systems* or *Kripke structures*.

σ-structure
$\mathcal{M}$
$\mathcal{V}^{\mathcal{M}}_{\text{Prop}}$

Definition 3.3 A *σ-structure* $\mathcal{M}$ is a directed graph $(V(\mathcal{M}), E(\mathcal{M}))$ together with a valuation $\mathcal{V}^{\mathcal{M}}_{\text{Prop}} : \sigma \to 2^{\mathcal{M}}$ of the atomic propositions. ⊣

Whenever we use $\mathcal{M}$ as a set, we mean its set of vertices $V(\mathcal{M})$.

Definition 3.4 Let $\varphi \in L_\mu[\sigma]$ and $\mathcal{M}, v$ be a σ-structure with a valuation $\mathcal{V}^{\mathcal{M}}_{\text{Prop}} : \sigma \to 2^{\mathcal{M}}$ together with another valuation

$\mathcal{V} : \mathrm{Var} \to 2^{\mathcal{M}}$ for the variables. The set $[\![\varphi]\!]^{\mathcal{M}}_{\mathcal{V}} \subseteq \mathcal{M}$ of vertices satisfying φ is defined inductively as follows.

$\mathcal{V}$
$[\![\varphi]\!]^{\mathcal{M}}_{\mathcal{V}}$

- $[\![\top]\!]^{\mathcal{M}}_{\mathcal{V}} = \mathcal{M}$ and $[\![\bot]\!] = \emptyset$.
- $[\![P]\!]^{\mathcal{M}}_{\mathcal{V}} = \mathcal{V}^{\mathcal{M}}_{\mathrm{Prop}}(P)$ for $P \in \sigma$.
- $[\![X]\!]^{\mathcal{M}}_{\mathcal{V}} = \mathcal{V}(X)$ for $X \in \mathrm{Var}$.
- $[\![\varphi \vee \psi]\!]^{\mathcal{M}}_{\mathcal{V}} = [\![\varphi]\!]^{\mathcal{M}}_{\mathcal{V}} \cup [\![\psi]\!]^{\mathcal{M}}_{\mathcal{V}}$.
- $[\![\varphi \wedge \psi]\!]^{\mathcal{M}}_{\mathcal{V}} = [\![\varphi]\!]^{\mathcal{M}}_{\mathcal{V}} \cap [\![\psi]\!]^{\mathcal{M}}_{\mathcal{V}}$.
- $[\![\neg\varphi]\!]^{\mathcal{M}}_{\mathcal{V}} = \mathcal{M} \setminus [\![\varphi]\!]^{\mathcal{M}}_{\mathcal{V}}$.
- $[\![\Diamond\varphi]\!]^{\mathcal{M}}_{\mathcal{V}} = \{v \in \mathcal{M} \mid$ there is a w with $v \to w$ and $w \in [\![\varphi]\!]^{\mathcal{M}}_{\mathcal{V}}\}$.
- $[\![\Box\varphi]\!]^{\mathcal{M}}_{\mathcal{V}} = \{v \in \mathcal{M} \mid$ for all w with $v \to w$ it holds that $w \in [\![\varphi]\!]^{\mathcal{M}}_{\mathcal{V}}\}$.
- $[\![\mu X.\varphi]\!]^{\mathcal{M}}_{\mathcal{V}} = \bigcap\{S \subseteq \mathcal{M} \mid S \supseteq [\![\varphi]\!]^{\mathcal{M}}_{\mathcal{V}[X \mapsto S]}\}$.
- $[\![\nu X.\varphi]\!]^{\mathcal{M}}_{\mathcal{V}} = \bigcup\{S \subseteq \mathcal{M} \mid S \subseteq [\![\varphi]\!]^{\mathcal{M}}_{\mathcal{V}[X \mapsto S]}\}$.

If $v \in [\![\varphi]\!]^{\mathcal{M}}_{\mathcal{V}}$, then we denote this by

$$\mathcal{M}, \mathcal{V}, v \models \varphi.$$

Often φ will have no free variables and thus the choice of $\mathcal{V}$ will be irrelevant. In this case we write

$$\mathcal{M}, v \models \varphi. \qquad \dashv$$

equivalent
$\varphi \equiv \psi$

Definition 3.5 We say that $\varphi, \psi \in L_\mu[\sigma]$ are *equivalent*, denoted $\varphi \equiv \psi$, if and only if

$$\mathcal{M}, \mathcal{V}, v \models \varphi \iff \mathcal{M}, \mathcal{V}, v \models \psi$$

holds for all σ-structures $\mathcal{M}, v$ and for all variable assignments $\mathcal{V}$. ⊣

Let us abbreviate the term used in the definition of the semantics of the fixpoint operators:

F^X_φ

$$F^X_\varphi : 2^{\mathcal{M}} \to 2^{\mathcal{M}},\ S \mapsto [\![\varphi]\!]^{\mathcal{M}}_{\mathcal{V}[X \mapsto S]}.$$

With this, we can write the semantics of the fixpoints as

$$[\![\mu X.\varphi]\!]^{\mathcal{M}}_{\mathcal{V}} = \bigcap \{S \subseteq \mathcal{M} \mid S \supseteq F^X_\varphi(S)\},$$
$$[\![\nu X.\varphi]\!]^{\mathcal{M}}_{\mathcal{V}} = \bigcup \{S \subseteq \mathcal{M} \mid S \subseteq F^X_\varphi(S)\}.$$

Lemma 3.6 *Let $\varphi(X) \in L_\mu[\sigma]$ such that X occurs only positively in φ. Then for all $S \subseteq T \subseteq \mathcal{M}$, it holds that $F^X_\varphi(S) \subseteq F^X_\varphi(T)$.*

Proof. This follows by a simple structural induction over φ. Here it is essential that X occurs only positively in φ. ■

Lemma 3.6 is usually stated as saying that F^X_φ is *monotone*.

Definition 3.7 Let A be a set and $F : 2^A \to 2^A$ be a function.
monotone
F is *monotone* if and only if for all $X \subseteq Y \subseteq A$ it holds that $F(X) \subseteq F(Y)$. ⊣

fixpoint
A *fixpoint* of a function F is an X with $F(X) = X$. By the following well-known theorem by Bronisław Knaster and

Alfred Tarski, a monotone map of sets (or more generally, a map on a complete lattice) has a unique least and a unique greatest fixpoint, and they can be expressed as we did in the semantics of $\mu X.\varphi$ and $\nu X.\varphi$, respectively.

Theorem 3.8 ([Tar55]) *Let A be a set and let $F : 2^A \to 2^A$ be a monotone function. Define*

$$\mu F := \bigcap \{X \subseteq A \mid X \supseteq F(X)\},$$
$$\nu F := \bigcup \{X \subseteq A \mid X \subseteq F(X)\}.$$

Then

1. *$F(\mu F) = \mu F$ and $F(\nu F) = \nu F$ and*
2. *for every set $X \subseteq A$ with $F(X) = X$ it holds that*

$$\mu F \subseteq X \subseteq \nu F.$$

In this sense the operators μ, ν compute the least and the greatest fixpoint, respectively.

This implies the following equivalences.

Corollary 3.9 *For all $\varphi \in L_\mu$ the following two equivalences hold.*

$$\mu X.\varphi \equiv \varphi[X/\mu X.\varphi]$$
$$\nu X.\varphi \equiv \varphi[X/\nu X.\varphi]$$

Proof. Indeed, $[\![\mu X.\varphi]\!] = F_\varphi^X([\![\mu X.\varphi]\!]) = [\![\varphi[X/\mu X.\varphi]]\!]$, and similar for $\nu X.\varphi$. ∎

We can also view these fixpoint operators via their approximations, which is useful for some proofs. For this and in a

few other places in this thesis, we are going to need ordinal numbers and transfinite induction. Because the definition of ordinal numbers and transfinite induction is outside the scope of this thesis, we would like to refer the reader to the very accessible book by Keith Devlin on contemporary set theory for a good introduction to this topic [Dev93].

α-approximant

Definition 3.10 Let α be an ordinal and $\varphi \in L_\mu$. We define the *α-approximant* $\mu X^\alpha.\varphi$ inductively as follows.

$\mu X^\alpha.\varphi$

$$\begin{aligned} [\![\mu X^0.\varphi]\!]^{\mathcal{M}}_{\mathcal{V}} &:= \emptyset \\ [\![\mu X^{\alpha+1}.\varphi]\!]^{\mathcal{M}}_{\mathcal{V}} &:= [\![\varphi[X/\mu X^\alpha.\varphi]]\!]^{\mathcal{M}}_{\mathcal{V}} \\ [\![\mu X^\beta.\varphi]\!]^{\mathcal{M}}_{\mathcal{V}} &:= \bigcup_{\gamma<\beta} [\![\mu X^\gamma.\varphi]\!]^{\mathcal{M}}_{\mathcal{V}} \qquad \text{for limit ordinals } \beta. \end{aligned}$$

For greatest fixpoints, we define the semantics of the α-approximant as follows.

$\nu X^\alpha.\varphi$

$$\begin{aligned} [\![\nu X^0.\varphi]\!]^{\mathcal{M}}_{\mathcal{V}} &:= \emptyset \\ [\![\nu X^{\alpha+1}.\varphi]\!]^{\mathcal{M}}_{\mathcal{V}} &:= [\![\varphi[X/\nu X^\alpha.\varphi]]\!]^{\mathcal{M}}_{\mathcal{V}} \\ [\![\nu X^\beta.\varphi]\!]^{\mathcal{M}}_{\mathcal{V}} &:= \bigcap_{\gamma<\beta} [\![\nu X^\gamma.\varphi]\!]^{\mathcal{M}}_{\mathcal{V}} \qquad \text{for limit ordinals } \beta. \ \dashv \end{aligned}$$

Lemma 3.11 *For every σ-structure $\mathcal{M}$, there exists an ordinal α such that*

$$[\![\mu X^\alpha.\varphi]\!]^{\mathcal{M}}_{\mathcal{V}} = [\![\mu X.\varphi]\!]^{\mathcal{M}}_{\mathcal{V}} \qquad \textit{and} \qquad [\![\nu X^\alpha.\varphi]\!]^{\mathcal{M}}_{\mathcal{V}} = [\![\nu X.\varphi]\!]^{\mathcal{M}}_{\mathcal{V}}.$$

Proof. First, we observe by ordinal induction and Lemma 3.6 that $\alpha \leq \beta$ implies $[\![\mu X^\alpha.\varphi]\!] \subseteq [\![\mu X^\beta.\varphi]\!]$. Next, suppose to

the contrary that $[\![\mu X^\alpha.\varphi]\!] \subsetneq [\![\mu X^{\alpha+1}.\varphi]\!]$ for all α. Then we have an injective mapping of the class of ordinals into $\mathcal{M}$ by mapping α to some $v \in \mathcal{M}$ with $v \in [\![\mu X^{\alpha+1}.\varphi]\!] \setminus [\![\mu X^\alpha.\varphi]\!]$. This is a contradiction because the ordinals form a proper class. So we find some α such that $[\![\mu X^\alpha.\varphi]\!] = [\![\mu X^{\alpha+1}.\varphi]\!]$.

This means that $[\![\mu X^\alpha.\varphi]\!]$ is a fixpoint of F^X_φ. So we have $[\![\mu X.\varphi]\!] \subseteq [\![\mu X^\alpha.\varphi]\!]$ because $[\![\mu X.\varphi]\!]$ is the least fixpoint.

What remains to show is $[\![\mu X^\alpha.\varphi]\!] \subseteq [\![\mu X.\varphi]\!]$. We show this for all α by ordinal induction. For $\alpha = 0$ and for limit ordinals this is trivial. For the case of successor ordinals, assume that $[\![\mu X^\alpha.\varphi]\!] \subseteq [\![\mu X.\varphi]\!]$. Then we have with Lemma 3.6

$$[\![\mu X^{\alpha+1}.\varphi]\!] = F^X_\varphi\Big([\![\mu X^\alpha.\varphi]\!]\Big) \subseteq F^X_\varphi\Big([\![\mu X.\varphi]\!]\Big) = [\![\mu X.\varphi]\!].$$

The proof for νX works analogously. ∎

3.1.3 Negation Normal Form

In every logic a natural task is to seek a normal form. Common examples from first-order logic are the prenex form (all quantifiers in front) and the Gaifman form (Boolean combination of basic-local sentences) [EF99]. For the modal μ-calculus, we will introduce only one normal form, the *negation normal form* where every negation is in front of an atomic formula. In the literature, this form is sometimes also called the *positive normal form*.

Definition 3.12 We say that a formula $\varphi \in L_\mu$ is in *negation normal form* if negations occur only in front of atomic formulas. ⊣

negation normal form

To work towards this normal form, let us first collect a few

fundamental equivalences.

Lemma 3.13 *For all $\varphi \in L_\mu$, the following equivalences hold.*

1. $\Box\varphi \equiv \neg\Diamond\neg\varphi$.
2. $\mu X.\varphi(X) \equiv \neg\nu X.\neg\varphi(\neg X)$.

Proof.

1. Let $\mathcal{M}$ be a σ-structure. We have

$$\begin{aligned}
[\![\Box\varphi]\!]^{\mathcal{M}}_{\mathcal{V}} &= \Big\{ v \in \mathcal{M} \;\Big|\; \text{for all } w \text{ with } v \to w \\
&\qquad \text{it holds that } w \in [\![\varphi]\!]^{\mathcal{M}}_{\mathcal{V}} \Big\} \\
&= \Big\{ v \in \mathcal{M} \;\Big|\; \text{there is no } w \text{ with } v \to w \\
&\qquad \text{and } w \notin [\![\varphi]\!]^{\mathcal{M}}_{\mathcal{V}} \Big\} \\
&= \mathcal{M} \setminus \Big\{ v \in \mathcal{M} \;\Big|\; \text{there is a } w \text{ with } v \to w \\
&\qquad \text{and } w \in [\![\neg\varphi]\!]^{\mathcal{M}}_{\mathcal{V}} \Big\} \\
&= \mathcal{M} \setminus [\![\Diamond\neg\varphi]\!]^{\mathcal{M}}_{\mathcal{V}} \\
&= [\![\neg\Diamond\neg\varphi]\!]^{\mathcal{M}}_{\mathcal{V}}.
\end{aligned}$$

2. Let $\mathcal{M}$ be a σ-structure and

$$v \in \bigcap \Big\{ S \subseteq \mathcal{M} \;\Big|\; S \supseteq [\![\varphi]\!]^{\mathcal{M}}_{\mathcal{V}[X \mapsto S]} \Big\}. \tag{3.1}$$

We need to show that

$$\bigcap \Big\{ S \subseteq \mathcal{M} \;\Big|\; S \supseteq [\![\varphi]\!]^{\mathcal{M}}_{\mathcal{V}[X \mapsto S]} \Big\} =$$

$$\mathcal{M} \setminus \bigcup \left\{ S \subseteq \mathcal{M} \;\middle|\; S \subseteq [\![\neg\varphi]\!]^{\mathcal{M}}_{\mathcal{V}[X \mapsto \mathcal{M} \setminus S]} \right\}.$$

Assume to the contrary that

$$v \in \bigcup \left\{ S \subseteq \mathcal{M} \;\middle|\; S \subseteq [\![\neg\varphi]\!]^{\mathcal{M}}_{\mathcal{V}[X \mapsto \mathcal{M} \setminus S]} \right\}.$$

Then there exists an $S \subseteq \mathcal{M}$ with $v \in S$ and

$$S \subseteq [\![\neg\varphi]\!]^{\mathcal{M}}_{\mathcal{V}[X \mapsto \mathcal{M} \setminus S]}. \tag{3.2}$$

We claim that

$$\mathcal{M} \setminus S \supseteq [\![\varphi]\!]^{\mathcal{M}}_{\mathcal{V}[X \mapsto \mathcal{M} \setminus S]}.$$

Indeed, for every w contained in the right side we have

$$w \notin [\![\neg\varphi]\!]^{\mathcal{M}}_{\mathcal{V}[X \mapsto \mathcal{M} \setminus S]},$$

which together with (3.2) proves the claim. Now $S' := \mathcal{M} \setminus S$ is a set of vertices contradicting (3.1).

The other direction follows similarly. ∎

It is obvious that $\neg\neg\varphi \equiv \varphi$ holds for all $\varphi \in L_\mu$, from which we get the following equivalences.

Corollary 3.14 *For all $\varphi \in L_\mu$, the following equivalences hold.*

1. $\neg\Box\varphi \equiv \Diamond\neg\varphi$.
2. $\neg\Diamond\varphi \equiv \Box\neg\varphi$.
3. $\neg\mu X.\varphi(X) \equiv \nu X.\neg\varphi(\neg X)$.

4. $\neg\nu X.\varphi(X) \equiv \mu X.\neg\varphi(\neg X)$.

We also trivially have that $\neg\neg\varphi \equiv \varphi$. By repeated application of these equivalences, we can push all negations deeper into the formula until they all are in front of atomic formulas. Thus we arrive at our desired normal form.

Corollary 3.15 *Every $\varphi \in L_\mu$ is equivalent to a formula in negation normal form. Furthermore, the negation normal form of φ can be computed in time $O(|\varphi|)$.*

3.1.4 Recursion

The semantics gives a rather abstract picture of how the fixpoints work. Here we would like to provide a different picture, possibly more intuitive. The same idea appears in the survey by Julian Bradfield and Colin Stirling [BS07].

Recalling Corollary 3.9, a fixpoint can be seen as a recursion by the equivalence

$$\mu X.\varphi \equiv \varphi[X/\mu X.\varphi].$$

So if we want to know whether $\mathcal{M}, v \models \mu X.\varphi$ holds we can instead ask whether $\mathcal{M}, v \models \varphi[X/\mu X.\varphi]$ holds.

The idea now is that a least fixpoint contains only what is absolutely necessary, whereas a greatest fixpoint contains everything it can. In the evaluation of the fixpoints, this means that a μ-fixpoint contains only the elements for which we can prove in a finite number of steps that they must be included.

To explain this concept, let us consider one of the easiest cases. Let $\mathcal{M}$ be a σ-structure consisting of a cycle with two vertices v, w, as shown in Figure 3.1. We obtain the following

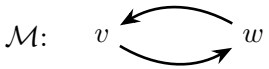


Figure 3.1: A cycle $\mathcal{M}$ of 2 vertices

equivalences.

$$\begin{aligned} & \mathcal{M}, v \models \mu X.\Diamond X \\ \iff & \mathcal{M}, v \models \Diamond \mu X.\Diamond X \\ \iff & \mathcal{M}, w \models \mu X.\Diamond X \\ \iff & \mathcal{M}, w \models \Diamond \mu X.\Diamond X \\ \iff & \mathcal{M}, v \models \mu X.\Diamond X. \end{aligned}$$

Clearly, this evaluation goes on forever. We cannot prove in a finite number of steps that v must be included in the fixpoint, so it is not included and thus $\mathcal{M}, v \not\models \mu X.\Diamond X$. On the other hand, $\mathcal{M}, v \models \nu X.\Diamond X$ because ν denotes a greatest fixpoint and we cannot refute the inclusion of v in a finite number of steps.

The very similar formula $\mu X.\Box X$ behaves rather differently. Because $\Box\psi$, independent of ψ, is true on leaves (vertices without successors), $\mu X.\Box X$ is true on all leaves. So all leaves must be included in the least fixpoint. But now there may be vertices whose successors are all leaves. These must be included in the fixpoint as well because on them $\Box X$ holds.

Inductively we see that $\mu X.\Box X$ is true on a vertex v if and only if all paths starting in v have finite length.

3.1.5 Examples

Let us look at a few more examples to gain some intuition for the expressivity of the modal μ-calculus.

$\mu X.\Diamond X$
: Always false, as we saw above.

$\neg\nu X.\Box X$
: Always false. This formula is equivalent to the previous one.

$\mu X.\Box X$, $\neg\nu X.\Diamond X$
: All paths from the current vertex are finite.

$\mu X.\Box\bot \vee \Diamond X$
: There is a finite maximal path from the current vertex.

$\mu X.\Box\bot \vee \Diamond\Diamond X$
: There is a finite maximal path of even length from the current vertex.

$\mu X.\varphi \vee (\psi \wedge \Diamond X)$
: There is a path where φ holds on the last vertex and ψ holds until the vertex before the last.

$\nu X.\mu Y.(P \wedge \Box X) \vee \Box Y$
: All infinite paths contain an infinite number of P-vertices.

  The idea is that we can descend only finitely often into Y; eventually the left disjunct must be reached. Then P holds at the current vertex and on all successors the whole formula holds again. This must happen an infinite number of times (if there are successors), so on all infinite paths there must be an infinite number of P-vertices.

$\nu X.\mu Y.((P \wedge \Box X) \vee \Box Y) \wedge \Diamond \top$
: On all infinite paths we see P an infinite number of times and all paths are infinite.

$\mu Y.\nu X.(P \wedge \Box X) \vee \Box Y$
: All infinite paths eventually contain only P-vertices.

We also observe that the last formula implies $\nu X.\mu Y.(P \wedge \Box X) \vee \Box Y$. Indeed, this implication of swapped fixpoints holds in general.

Lemma 3.16 ([BS07, page 746])

$$(\mu X.\nu Y.\varphi) \to \nu Y.\mu X.\varphi$$

is a tautology for all $\varphi \in L_\mu[\sigma]$.

However, the opposite direction does not hold.

Lemma 3.17 *There exists a formula* $\varphi \in L_\mu[\sigma]$ *and a* σ*-structure* $\mathcal{M}, v$ *such that*

$$\mathcal{M}, v \not\models (\nu X.\mu Y.\varphi) \to \mu Y.\nu X.\varphi.$$

Proof. Choose $\varphi := (P \wedge \Box X) \vee \Box Y$ and the structure from Figure 3.1 extended with a valuation of P such that P is true only at v. Then we have

$$\mathcal{M}, v \models \nu X.\mu Y.(P \wedge \Box X) \vee \Box Y$$

because we descend through X and Y alternately an infinite number of times and the outermost fixpoint is a ν fixpoint.

However, we have

$$\mathcal{M}, v \not\models \mu Y.\nu X.(P \wedge \Box X) \vee \Box Y$$

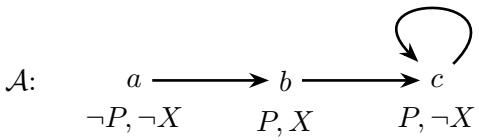


Figure 3.2: The structure $\mathcal{A}$ for Example 3.18

because this time descending an infinite number of times through Y is not allowed because it is the outermost fixpoint. But we also cannot avoid Y because we can use only the left disjunct where P holds, and P does not hold at w. ■

Example 3.18

$$\mu X.\nu Y.(\Box Y \wedge \Box X) \vee (\neg P \wedge \Box Y) \equiv \mu X.\nu Y.(P \wedge \Box X) \vee (\neg P \wedge \Box Y)$$

Both formulas are equivalent to $\mu X.\nu Y.\Box X \vee (\neg P \wedge \Box Y)$ and all three have the meaning "P occurs only finitely often on each reachable path". Again the reason for this is that we cannot expand X infinitely often because it is bound by a μ operator, and the binding formula of Y is nested inside.

However, the greatest fixpoints by themselves are not equivalent. That is,

$$\nu Y.(P \wedge \Box X) \vee (\neg P \wedge \Box Y) \not\equiv \nu Y.\Box X \vee (\neg P \wedge \Box Y).$$

For example, take a 3-vertex model $\mathcal{A}, a$ with vertex set $\{a, b, c\}$ and arcs $a \to b \to c$ and $c \to c$, as shown in Figure 3.2. Let b, c be P-vertices and $\mathcal{A}, b \models X$ (and negative on the remaining vertices). Then

$$\mathcal{A}, a \not\models \nu Y.(P \wedge \Box X) \vee (\neg P \wedge \Box Y)$$

but

$$\mathcal{A}, a \models \nu Y.\Box X \vee (\neg P \wedge \Box Y).$$

3.1.6 Bisimulation Invariance

One notable property of the modal μ-calculus is that it is invariant under *bisimulations*, an equivalence relation between vertices of σ-structures.

Definition 3.19 A *bisimulation* between two σ-structures $\mathcal{M}_1$, $\mathcal{M}_2$ is a binary relation $R \subseteq V(\mathcal{M}_1) \times V(\mathcal{M}_2)$ satisfying the following properties for all $(v, w) \in R$. bisimulation

1. v and w satisfy the same atomic formulas.
2. For every successor v' of v in $\mathcal{M}_1$ there exists a successor w' of w in $\mathcal{M}_2$ such that $(v', w') \in R$.
3. For every successor w' of w in $\mathcal{M}_2$ there exists a successor v' of v in $\mathcal{M}_1$ such that $(v', w') \in R$.

We write $\mathcal{M}_1, v_0 \sim \mathcal{M}_2, w_0$, read "$\mathcal{M}_1, v_0$ and $\mathcal{M}_2, w_0$ are *bisimilar*", if there exists a bisimulation R between $\mathcal{M}_1$ and $\mathcal{M}_2$ with $(v_0, w_0) \in R$. ⊣ bisimilar

Is is easy to check that $\sim$ is an equivalence relation.

Definition 3.20 We call a formula $\varphi \in L_\mu[\sigma]$ *bisimulation invariant* if for all σ-structures $\mathcal{M}_1, v$ and $\mathcal{M}_2, w$ with $\mathcal{M}_1, v \sim \mathcal{M}_2, w$ it holds that bisimulation invariant

$$\mathcal{M}_1, v \models \varphi \iff \mathcal{M}_2, w \models \varphi. \qquad \dashv$$

Bisimulations and bisimilar structures are fundamental to modal logic because most modal logics are bisimulation invariant, that is, they cannot distinguish between bisimilar structures. Even more surprisingly, invariance under bisimulation *characterizes* many modal logics. For example, consider a formula of first-order logic with one free variable that is invariant under bisimulation, with bisimulation invariance defined analogously to Definition 3.20. Johan van Benthem famously showed that every such formula has an equivalent formula in basic modal logic [Ben84; BB07].

More recently, David Janin and Igor Walukiewicz showed that L_μ is the bisimulation invariant fragment of monadic second order logic (MSO) [JW96].

For us, bisimulations will play a role in Section 5.4. Because the proof is short, let us show that every formula in L_μ is bisimulation invariant. First, we need that substitution preserves bisimulation invariance.

Lemma 3.21 *Let $\varphi(X), \psi \in L_\mu[\sigma]$ be bisimulation invariant. Then $\varphi[X/\psi]$ is bisimulation invariant.*

Proof. We assume without loss of generality that X is the only free variable in φ and that ψ has no free variables.

Let $\mathcal{M}_1$, $\mathcal{M}_2$ be two σ-structures with a bisimulation R. Then for all $(v, w) \in R$ it holds that $v \in [\![\psi]\!]^{\mathcal{M}_1}$ if and only if $w \in [\![\psi]\!]^{\mathcal{M}_2}$ because ψ is bisimulation invariant.

Let $\mathcal{M}_1'$, $\mathcal{M}_2'$ be the $\sigma \cup \{X\}$-structures with X-valuations

$$[\![X]\!]^{\mathcal{M}_1} := [\![\psi]\!]^{\mathcal{M}_1},$$
$$[\![X]\!]^{\mathcal{M}_2} := [\![\psi]\!]^{\mathcal{M}_2}.$$

It follows that R is also a bisimulation between $\mathcal{M}_1'$ and $\mathcal{M}_2'$.

Because φ is bisimulation invariant, it follows that for all $(v, w) \in R$ we have

$$v \in [\![\varphi]\!]^{\mathcal{M}_1'} \iff w \in [\![\varphi]\!]^{\mathcal{M}_2'}.$$

But this is equivalent to saying

$$v \in [\![\varphi[X/\psi]]\!]^{\mathcal{M}_1} \iff w \in [\![\varphi[X/\psi]]\!]^{\mathcal{M}_2}$$

for all $(v, w) \in R$. This shows that $\varphi[X/\psi]$ is bisimulation invariant. ∎

Theorem 3.22 ([BS07, Theorem 4]) *Every $\varphi \in L_\mu[\sigma]$ if bisimulation invariant.*

Proof. We prove this by structural induction. The only non-trivial case is the case $\mu X.\varphi$ (the case $\nu X.\varphi$ follows analogously), with the assumption that φ is bisimulation-invariant.

Clearly, we have that $\mu X^0.\varphi \equiv \bot$ is bisimulation invariant. If $\mu X^\alpha.\varphi$ is bisimulation invariant for some ordinal α, then $\mu X^{a+1}.\varphi = \varphi[X/\mu X^\alpha.\varphi]$ is bisimulation invariant by Lemma 3.21.

If α is a limit ordinal, then

$$\mathcal{M}_1, v \models \mu X^\alpha.\varphi \iff \mathcal{M}_2, w \models \mu X^\alpha.\varphi$$

for all $\mathcal{M}_1, v$, $\mathcal{M}_2, w$ holds by the definition of the α-approximant, Definition 3.10 on page 22.

So we have that $\mu X^\alpha.\varphi$ is bisimulation invariant for all α. By Lemma 3.11, this implies the bisimulation invariance of $\mu X.\varphi$. ∎

3.1.7 Subformulas and Closure

Usually the definition of a subformula is very simple. However, we are going to need a slightly more involved definition. As a motivating example, consider the formula $\varphi := P \wedge \mu X.P \vee \Diamond X$. Clearly, the set of subformulas would be

$$\{\varphi,\ P,\ \mu X.P \vee \Diamond X,\ P \vee \Diamond X,\ \Diamond X,\ X\}.$$

However, this ignores part of the structure of φ: The subformula "P" appears twice in different locations, and one occurrence is under a fixpoint operator. It also loses information about X: Although not the case here, the variable X might occur under multiple fixpoint operators all binding X, so knowing that "X" is a subformula is useless information.

Separating these occurrences and tracking information about fixpoint variables will be crucial in Chapter 5, so we need to define the set of subformulas in a more intricate way.

One simple way to achieve this is to consider φ as a string and equip every subformula with its position in the string φ. This distinguishes identical subformulas by their position in φ, and connects fixpoint variables to their enclosing fixpoint operator. To make this well-defined, recall that in the syntax of L_μ in Definition 3.2, we introduced parentheses. So φ from the example above would formally be written $\varphi = (P \wedge \mu X.(P \vee \Diamond X))$. There is only one way to put these parentheses according to the syntax definition.

Observe also that no two subformulas start at the same position in this string. So it would already be sufficient to remember only the indices where subformulas start instead of the whole formula. However, this would be very cumbersome, so

we talk about indexed subformulas instead of only the indices.

For reasons that will be obvious in a moment, we are also going to exclude single fixpoint variables from our set of subformulas.

Definition 3.23 For $\varphi \in L_\mu$, let $\mathrm{sub}(\varphi)$ be the set of all indexed subformulas without formulas of the form X for fixpoint variables X. Indices start at 1. That is, $\mathrm{sub}(\varphi)$

$$\begin{aligned}\mathrm{sub}(\varphi) := \{(\psi, i) \mid\ & \psi \text{ is a subformula of } \varphi \\ & \text{starting at position } i \text{ in the string } \varphi \\ & \text{and } \psi \text{ is not a fixpoint variable}\}.\end{aligned}$$

Let $\mathrm{sub}^+(\varphi) = \mathrm{sub}(\varphi) \setminus \{(\varphi, 0)\}$ be the set of proper subformulas. $\mathrm{sub}^+(\varphi)$ ⊣

Definition 3.24 For an occurrence of a fixpoint variable X in a formula φ, its *definition in* φ is the enclosing fixpoint $(\mu X.\psi, i) \in \mathrm{sub}(\varphi)$ (or $(\nu X.\psi, i) \in \mathrm{sub}(\varphi)$) that binds this occurrence of X. For a formula $(\psi, i) \in \mathrm{sub}(\varphi)$, let $\mathrm{closure}_\varphi(\psi, i) = (\psi', i)$ be such that ψ' is the formula ψ with all free variables replaced by their definitions until there are no more free variables. definition in φ $\mathrm{closure}_\varphi(\psi, i)$ ⊣

Definition 3.25 Define

$$\mathrm{CL}(\varphi) := \{\mathrm{closure}_\varphi(\psi, i) \mid (\psi, i) \in \mathrm{sub}(\varphi)\} \qquad \mathrm{CL}(\varphi)$$

and

$$\mathrm{CL}^+(\varphi) := \mathrm{CL}(\varphi) \setminus \{(\varphi, 0)\}\,. \qquad \dashv \quad \mathrm{CL}^+(\varphi)$$

Lemma 3.26 $\text{closure}_\varphi : \text{sub}(\varphi) \to \text{CL}(\varphi)$ *is a bijection that is the identity on the second component.*

Proof. The function is surjective and preserves the second component. For injectivity, observe that $(\psi, i), (\chi, j) \in \text{sub}(\varphi)$ with $(\psi, i) \neq (\chi, j)$ implies $i \neq j$ and thus $\text{closure}_\varphi(\psi, i) \neq \text{closure}_\varphi(\chi, j)$. ∎

Here it is useful that we have no single fixpoint variables X in $\text{sub}(\varphi)$ because otherwise the definition of X would appear twice in $\text{CL}(\varphi)$, which would be inconvenient for some proofs. However, this is only a matter of style.

Example 3.27 Consider the formula $\varphi := (P \wedge \mu X.(P \vee \Diamond X))$ we mentioned earlier. As explained before, we do not omit the parentheses here. Because $(P \vee \Diamond X)$ has position 7 in the string "$(P \wedge \mu X.(P \vee \Diamond X))$" (indices start at 1), we have $((P \vee \Diamond X), 7) \in \text{sub}(\varphi)$ and

$$\text{closure}_\varphi\big((P \vee \Diamond X), 7\big) = \big((P \vee \Diamond(\mu X.(P \vee \Diamond X))), 7\big).$$

The full set of subformulas of φ is

$$\begin{aligned}\text{sub}(\varphi) = \Big\{&(\varphi, 1), (P, 2), (\mu X.(P \vee \Diamond X), 4),\\ &((P \vee \Diamond X), 7), (P, 8), (\Diamond X, 10)\Big\}\end{aligned}$$

and the closure is, with the abbreviation $\psi := \mu X.(P \vee \Diamond X)$,

$$\begin{aligned}\text{CL}(\varphi) = \Big\{&(\varphi, 1), (P, 2), (\mu X.(P \vee \Diamond X), 4),\\ &((P \vee \Diamond \psi), 7), (P, 8), (\Diamond \psi, 10)\Big\}.\end{aligned}$$

In order to not get entangled in unusual notation, we will usually omit the index and simply write $\psi \in \mathrm{sub}(\varphi)$ instead of $(\psi, i) \in \mathrm{sub}(\varphi)$ when there is no confusion. However, from now on we always implicitly assume that elements of $\mathrm{sub}(\varphi)$ or $\mathrm{CL}(\varphi)$ are equipped with its index in order to distinguish identical subformulas.

Except for the indices, the definition of $\mathrm{CL}(\varphi)$ is something well-known in disguise. In the literature, we find the definition of the *Fischer-Ladner closure* [SE89, Definition 4.1].

Definition 3.28 (Fischer-Ladner Closure) The *Fischer-Ladner closure* $\mathrm{FLC} : L_\mu[\sigma] \to 2^{L_\mu[\sigma]}$ is defined inductively as follows. $\mathrm{FLC}(\varphi)$

$$\begin{aligned}
\mathrm{FLC}(P) &:= \{P\} \\
\mathrm{FLC}(\varphi \wedge \psi) &:= \{\varphi \wedge \psi\} \cup \mathrm{FLC}(\varphi) \cup \mathrm{FLC}(\psi) \\
\mathrm{FLC}(\varphi \vee \psi) &:= \{\varphi \vee \psi\} \cup \mathrm{FLC}(\varphi) \cup \mathrm{FLC}(\psi) \\
\mathrm{FLC}(\neg\varphi) &:= \{\neg\varphi\} \cup \mathrm{FLC}(\varphi) \\
\mathrm{FLC}(\Box\varphi) &:= \{\Box\varphi\} \cup \mathrm{FLC}(\varphi) \\
\mathrm{FLC}(\Diamond\varphi) &:= \{\Diamond\varphi\} \cup \mathrm{FLC}(\varphi) \\
\mathrm{FLC}(\mu X.\varphi) &:= \{\mu X.\varphi\} \cup \mathrm{FLC}(\varphi[X/\mu X.\varphi]) \\
\mathrm{FLC}(\nu X.\varphi) &:= \{\nu X.\varphi\} \cup \mathrm{FLC}(\varphi[X/\nu X.\varphi]).
\end{aligned}$$ ⊣

Our notion of closure is essentially the same as the Fischer-Ladner closure.

Lemma 3.29 *For all* $\varphi \in L_\mu[\sigma]$,

$$\mathrm{FLC}(\varphi) = \{\psi \mid (\psi, i) \in \mathrm{CL}(\varphi)\}.$$

Let us also define the closure of a set of formulas. In this set we do not need the index i, different from $\mathrm{CL}(\varphi)$.

Definition 3.30 For a set of formulas $L \subseteq L_\mu$, define

CL(L)
$$\mathrm{CL}(L) := \{\psi \mid \varphi \in L,\ (\psi, i) \in \mathrm{CL}(\varphi)\}. \qquad \dashv$$

3.1.8 Formula Depth

Definition 3.31 Let $\varphi \in L_\mu[\sigma]$ be a formula. The *standard depth* $\mathrm{depth}(\varphi)$ of φ is defined inductively as follows.

standard depth

$$\begin{aligned}
\mathrm{depth}(P) = \mathrm{depth}(X) &= 0 \\
&\text{for } P \in \sigma,\ X \in \mathrm{Var} \\
\mathrm{depth}(\neg\varphi) &= \mathrm{depth}(\varphi) \\
\mathrm{depth}(\varphi \wedge \psi) &= \max\{\mathrm{depth}(\varphi), \mathrm{depth}(\psi)\} \\
\mathrm{depth}(\varphi \vee \psi) &= \max\{\mathrm{depth}(\varphi), \mathrm{depth}(\psi)\} \\
\mathrm{depth}(\Box\varphi) = \mathrm{depth}(\Diamond\varphi) &= 1 + \mathrm{depth}(\varphi) \\
\mathrm{depth}(\mu X.\varphi) = \mathrm{depth}(\nu X.\varphi) &= 1 + \mathrm{depth}(\varphi) \\
&\text{for } X \in \mathrm{Var}.
\end{aligned}$$ $\dashv$

Lemma 3.32 *Let σ and* Var *be finite. For a given $n \in \mathbb{N}$ there are only finitely many formulas of standard depth n up to logical equivalence.*

Proof. For $n = 0$ this is true because there are only finitely many Boolean combinations of the atomic propositions and fixpoint variables.

For $n > 0$, the Boolean combinations of formulas of the finite set

$$\{\Box\varphi, \Diamond\varphi, \mu X.\varphi, \nu X.\varphi \mid \mathrm{depth}(\varphi) = n - 1, X \in \mathrm{Var}\}$$

provide all possible formulas of standard depth n up to logical equivalence. ■

Definition 3.33 Let $\overline{X} = (X_1, \ldots, X_n)$ be a finite sequence of fixpoint variables. A formula $\varphi \in L_\mu$ is called *consistent with* $\overline{X}$ if

consistent with $\overline{X}$

1. all fixpoint variables of φ are in the sequence and
2. every X_i is bound only in μ-subformulas or only in ν-subformulas of φ and
3. in every subformula ψ of φ that binds a fixpoint variable X_i, only the variables $X_1, \ldots, X_i$ can appear freely in ψ.

⊣

It is common to define a partial order on the variables by saying X comes before Y if Y is bound inside of some μX or νX binding where Y occurs freely. However, this is not a well-defined partial order on every formula. As a simple counter-example, consider

$$(\mu X.\mu Y.X \wedge Y) \wedge (\mu Y.\mu X.X \wedge Y).$$

There is no way to order X and Y in the above formula by their syntactic nesting.

A common way to avoid this problem is to require that every variable is quantified at most once in the formula. However, this does not work for us because we want to use the equivalence $\mu X.\varphi \equiv \varphi[X/\mu X.\varphi]$, and the right-hand side will usually have more than one binding of X.

We avoid this problem instead by considering only certain nice formulas.

consistent **Definition 3.34** A formula $\varphi \in L_\mu$ is *consistent* if there is some linear order $\overline{X} = (X_1, \ldots, X_n)$ of variables such that φ is consistent with $\overline{X}$. A formula that is not consistent is *inconsistent*. ⊣

In particular, every formula with at most one binding per variable is trivially consistent by choosing a linear order that is a completion of the partial order induced by the nesting of the fixpoints.

It is also easy to see that every formula can be made consistent with some sequence $\overline{X} = (X_1, \ldots, X_n)$ of sufficient length by renaming the variables. So we can usually assume that we have consistent formulas. Let us prove a few operations that preserve consistency with a fixed sequence $\overline{X}$.

Lemma 3.35 *Let $\overline{X} = (X_1, \ldots, X_n)$, $\varphi \in L_\mu$ be consistent with $\overline{X}$ and $\psi \in \mathrm{sub}(\varphi)$. Then ψ is consistent with $\overline{X}$.*

Proof. None of the consistency conditions can be violated by going to a subformula, so this follows immediately. ■

Lemma 3.36 *Let $\overline{X} = (X_1, \ldots, X_n)$, $\varphi \in L_\mu$ be consistent with $\overline{X}$ and $\psi \in \mathrm{CL}(\varphi)$. Then ψ is consistent with $\overline{X}$.*

Proof. We prove this by structural induction over φ. In this proof we ignore the indices in $\mathrm{CL}(\varphi)$ because they are irrelevant for consistency. If $\varphi = \chi_1 \wedge \chi_2$, then clearly χ_1 and χ_2 are consistent with $\overline{X}$. We also have

$$\mathrm{CL}(\varphi) = \{\varphi\} \cup \mathrm{CL}(\chi_1) \cup \mathrm{CL}(\chi_2),$$

so every $\psi \in \mathrm{CL}(\varphi)$ is consistent with $\overline{X}$ either because $\psi = \varphi$ or by the induction hypothesis applied to χ_1 or χ_2, respectively.

The other Boolean operators and $\Box, \Diamond$ are similar, so we skip them and focus on the case of a μ fixpoint. The case of a ν fixpoint follows analogously.

Assume that $\varphi = \mu X_i.\chi$ and let $\psi \in \mathrm{CL}(\varphi)$. We assume that $\psi \neq \varphi$ because otherwise ψ is already consistent with $\overline{X}$. In order to apply the induction hypothesis, we need to show that $\chi[X_i/\varphi]$ is consistent with $\overline{X}$.

Clearly, every X_j in $\chi[X_i/\varphi]$ is bound only in μ or only in ν subformulas, and every fixpoint variable of $\chi[X_i/\varphi]$ is among $X_1, \ldots, X_n$.

Let us check the last consistency condition. Let $\mu X_j.\chi' \in \mathrm{sub}(\chi[X_i/\varphi])$ (the case ν follows analogously). We need to show that the free variables of χ' are among $X_1, \ldots, X_j$. We distinguish two cases.

Case 1: $\mu X_j.\chi'$ is a subformula of φ. Then we are done by using the assumption that φ is consistent with $\overline{X}$ and the previous lemma.

Case 2: $\mu X_j.\chi'$ is not a subformula of φ. In this case, there exists some $\mu X_j.\chi'' \in \mathrm{sub}(\chi)$ not within the scope of an X_i-binding such that $(\mu X_j.\chi'')[X_i/\varphi] = \mu X_j.\chi'$. We use $\mathrm{sub}(\chi)$ here so that the outermost X_i-binding in $\varphi = \mu X_i.\chi$ does not count. Because $\mu X_j.\chi'$ is not a subformula of φ, X_i is free in χ''. Note that we need to assume that $\mu X_j.\chi''$ is not within the scope of an X_i-binding because $\varphi = \mu X_i.\chi$ might bind X_i more than once.

Because φ is consistent with $\overline{X}$, so is $\mu X_j.\chi''$. So the free variables of χ'' are among $X_1, \ldots, X_j$. Because X_i is free in χ'', this implies $i \leq j$. Then the free variables of $(\mu X_j.\chi'')[X_i/\varphi]$ are among $X_1, \ldots, X_j$, as required. ∎

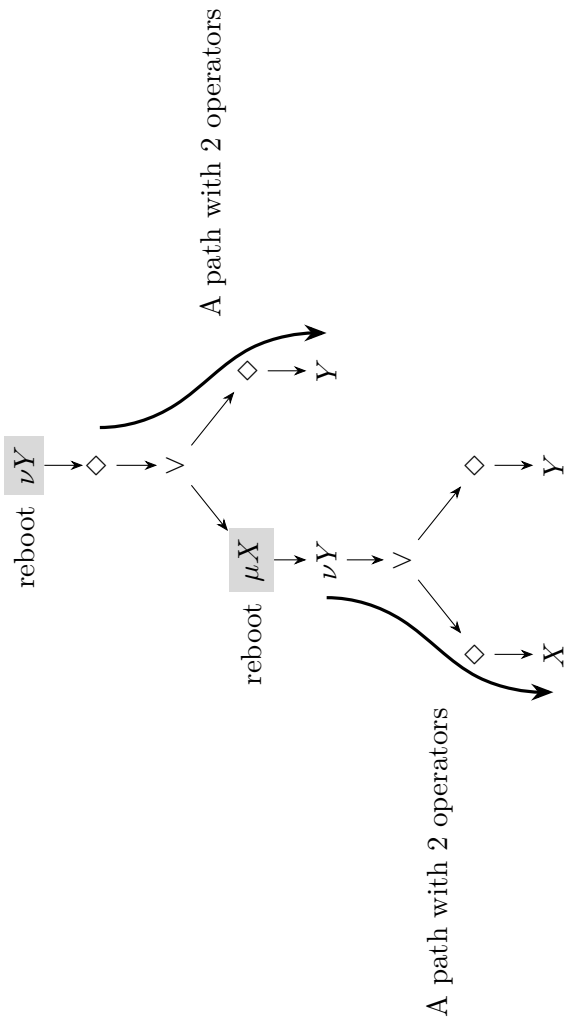


Figure 3.3: An example for reboots and reboot-free paths in the syntax tree

Definition 3.37 Let φ be a formula consistent with $\overline{X}$ and let x be a vertex in the syntax tree labeled with some formula of the form $\mu X_i.\psi$ or $\nu X_i.\psi$. We call x a *reboot* if no ancestor of x binds a fixpoint variable among $X_1, \ldots, X_i$. reboot

The $\overline{X}$*-depth* of a formula is the biggest number of occurrences of operators from the set $\Box, \Diamond, \mu, \nu$ that can be found on a path in the syntax tree that does not visit reboot vertices. The $\overline{X}$-depth is undefined if the formula is not consistent with $\overline{X}$. $\overline{X}$-depth ⊣

Intuitively, a reboot is a subformula φ of the form $\varphi = \mu X_i.\psi$ or $\varphi = \nu X_i.\psi$ without free variables. This is because by consistency, such a ψ can only have $X_1, \ldots, X_i$ as free variables, but it is a reboot only if it is not within the scope of some formula binding one of these variables.

Figure 3.3 shows the formula $\varphi = \nu Y.\psi = \nu Y.\Diamond(\mu X.\nu Y.\Diamond X \vee \Diamond Y) \vee \Diamond Y$, which has (X, Y)-depth 2. Note that φ is inconsistent with (Y, X). The definition is designed so that going from $\nu Y.\psi$ to $\psi[Y/\nu Y.\psi]$ does not increase the (X, Y)-depth.

Lemma 3.38 *For all $\alpha X_i.\varphi \in L_\mu$ ($\alpha \in \{\mu, \nu\}$) consistent with $\overline{X} = (X_1, \ldots, X_k)$, the $\overline{X}$-depth of $\alpha X_i.\varphi$ is not smaller than the $\overline{X}$-depth of $\varphi[X_i/\alpha X_i.\varphi]$.*

Proof. First, we observe that by Lemma 3.36, $\varphi[X_i/\alpha X_i.\varphi]$ is consistent with $\overline{X}$.

Because $\alpha X_i.\varphi$ is consistent with $\overline{X}$, the free variables of φ are among $X_1, \ldots, X_i$. We show that every occurrence of $\alpha X_i.\varphi$ in $\varphi[X_i/\alpha X_i.\varphi]$ is a reboot. Suppose to the contrary that some $\alpha X_i.\varphi \in \mathrm{sub}(\varphi[X_i/\alpha X_i.\varphi])$ is not a reboot. Then there exists a subformula $\beta X_j.\psi \in \mathrm{sub}(\varphi[X_i/\alpha X_i.\varphi])$ ($\beta \in \{\mu, \nu\}$) containing $\alpha X_i.\varphi$, with $1 \leq j \leq i$ and $\beta X_j.\psi \neq \alpha X_i.\varphi$.

Let $\psi' \in \mathrm{sub}(\alpha X_i.\varphi)$ be the subformula such that

$$(\beta X_j.\psi')[X_i/\alpha X_i.\varphi] = \beta X_j.\psi.$$

In particular, X_i occurs free in $\beta X_j.\psi'$, so we have $1 \leq j < i$. Moreover, because $\alpha X_i.\varphi$ is consistent, the free variables of ψ' are among $X_1, \ldots, X_j$, contradicting the fact that X_i is free in $\beta X_j.\psi'$.

So every $\alpha X_i.\varphi \in \mathrm{sub}(\varphi[X_i/\alpha X_i.\varphi])$ is a reboot, which proves the lemma. ∎

By the definition of the closure operator CL, we get that the $\overline{X}$-depth is closed under application of CL.

Corollary 3.39 *Let $\varphi \in L_\mu$ a formula of $\overline{X}$-depth at most d. Then every $\psi \in \mathrm{CL}(\varphi)$ has $\overline{X}$-depth at most d.*

μ-depth

Definition 3.40 A *μ-depth* is a pair $\delta = (\overline{X}, d)$ where $\overline{X}$ is a sequence of fixpoint variables and d is a natural number. A

consistent with δ

formula is called *consistent with δ* if it is consistent with $\overline{X}$ and its $\overline{X}$-depth is at most d.

L-type

For a set $L \subseteq L_\mu$, define the *L-type* of a vertex in a structure to be the set of formulas from L that are true at the vertex.

δ-type

The *δ-type* of a vertex in a structure is its L-type, with L being the set of all formulas consistent with δ. ⊣

The δ-type of a vertex is finite thanks to the following lemma.

Lemma 3.41 *For every μ-depth δ and finite set of propositional variables, up to logical equivalence there are finitely many formulas in these propositional variables that are consistent with δ.*

Proof. Let $\delta = (\overline{X}, d)$ with $\overline{X} = (X_1, \ldots, X_n)$. We prove the statement by induction over the nesting depth of reboots. First, let us observe that we can have at most n reboots nested inside each other because a reboot μX_i (or νX_i) can contain only other reboots μX_j (or νX_j) with $j < i$ by the definition of reboot.

If a formula φ consistent with δ has no reboots, then we have $\mathrm{depth}(\varphi) \leq d$ by the definition of the standard depth. By Lemma 3.32, there are only finitely many formulas of standard depth at most d, which proves the base case.

For the induction step, let $k < n$ and suppose that the statement is true for the set of δ-consistent formulas with a reboot nesting depth of at most k. Let φ be a formula with reboot nesting depth of $k+1$. We see that if φ is of the form $\varphi = \mu X_i.\psi$ or $\varphi = \nu X_i.\psi$ for some i and some ψ of reboot nesting depth k, then there are only finitely many such φ by the induction hypothesis.

If φ does not start with a fixpoint operator, then we can write φ as

$$\varphi = \psi(Y_1, \ldots Y_l)[Y_1/\psi_1, \ldots, Y_l/\psi_l]$$

for a formula ψ in basic modal logic (that is, without fixpoint operators) and some formulas $\psi_1, \ldots, \psi_l$ of reboot nesting depth at most $k+1$ such that every ψ_i for $1 \leq i \leq l$ starts with a fixpoint operator. That is, the ψ_i are the first reboot vertices in the syntax tree of φ. Observe that $\mathrm{depth}(\psi) \leq d$ because φ is consistent with δ, so by Lemma 3.32 there is only a finite number of choices for ψ up to logical equivalence. By the previous paragraph, there is also only a finite number of choices for the ψ_i up to logical equivalence.

So in total we conclude that there is only a finite number of formulas φ of reboot nesting depth at most $k+1$ up to logical equivalence. ∎

Although the set in the statement of the above lemma is finite, its size is non-elementary with respect to δ. This follows from the fact that the number of non-equivalent L_μ formulas up to a certain standard depth n is non-elementary in n, even without fixpoint operators.

Lemma 3.42 *Let $\sigma = \{P, Q\}$. Then there exists a sequence $\Phi_0, \Phi_1, \ldots \subseteq L_\mu[\sigma]$ of sets of pairwise non-equivalent formulas such that*

1. $|\Phi_{i+1}| = 2^{|\Phi_i|}$ *for all* $i \in \mathbb{N}$ *and*
2. $\mathrm{depth}(\varphi) \leq i$ *for every* $\varphi \in \Phi_i$ *and*
3. *no formula uses a fixpoint operator* μ *or* ν.

Proof. We define

$$\Phi_0 := \{P, Q\}, \qquad \Phi_{i+1} := \left\{ \bigwedge_{\varphi \in \Phi} \Diamond\varphi \;\middle|\; \Phi \subseteq \Phi_i \right\}.$$

The conjunction over the empty set has the usual meaning of $\top$. It is obvious that these Φ_i satisfy the three listed conditions. It remains to show that no two formulas in Φ_i are equivalent. We do this by induction over i.

Clearly Φ_0 contains no pair of equivalent formulas. Let $i > 0$ and $\Phi, \Phi' \subseteq \Phi_{i-1}$ be such that $\Phi \neq \Phi'$. We need to show that

$$\bigwedge_{\varphi \in \Phi} \Diamond\varphi \not\equiv \bigwedge_{\varphi \in \Phi'} \Diamond\varphi.$$

Assume without loss of generality that there is a $\varphi_0 \in \Phi$ with $\varphi_0 \notin \Phi'$. Then all structures $\mathcal{M}, v$ satisfying $\bigwedge_{\varphi \in \Phi} \Diamond\varphi$ must have a successor w of v such that $\mathcal{M}, w \models \varphi_0$. On the other hand we can construct a structure $\mathcal{M}', v$ satisfying $\bigwedge_{\varphi \in \Phi'} \Diamond\varphi$ where v does not have such a successor, because by the induction hypothesis, all $\varphi \in \Phi_{i-1}$ are pairwise non-equivalent. ∎

3.2 Parity Games

3.2.1 Definition

A parity game is a game played by two players, called $\Diamond$ ("even") and $\Box$ ("odd"). For $i \in \{\Diamond, \Box\}$, we denote by $\bar{i}$ the element of $\{\Diamond, \Box\} \setminus \{i\}$.

Definition 3.43 (Parity Game) A *parity game* $P = (V, E, V_\Diamond, \omega)$ is a directed graph (V, E) with $V_\Diamond \subseteq V$ and a function $\omega : V \to \mathbb{N}$ mapping vertices to priorities with $|\omega(V)| < \infty$. We write $V_\Box := V \setminus V_\Diamond$. parity game $V_\Diamond, V_\Box, \omega$

Definition 3.44 (Play) A *play* in a parity game is a maximal path. ⊣ play

A play starts on some vertex v (we often denote a game with a chosen starting vertex as (P, v)). If the current vertex is in $V_\Diamond$, then it is player $\Diamond$'s turn, otherwise it is player $\Box$'s turn. In their turn, the players must choose an outgoing arc and the endpoint becomes the current vertex for the next turn. If a player cannot make a move, he loses. Otherwise, the game continues indefinitely.

Let us define the winner of a finite or infinite path.

winning path

Definition 3.45 (Winning Path) A path $\overline{v} \in V^* \cup V^\omega$ is *winning for player* $\diamond$ if and only if

- $\overline{v} = (v_1, \ldots, v_n)$ is finite and $v_n \in V_\Box$, or
- $\overline{v}$ is infinite and the minimum priority occurring infinitely often on $\overline{v}$ is even.

A path not winning for player $\diamond$ is winning for player $\Box$. ⊣

Note that the minimum always exists because of $|\omega(V)| < \infty$. By the pigeon hole principle, for every infinite path there exists at least one element in $\omega(V)$ that appears infinitely often on the path.

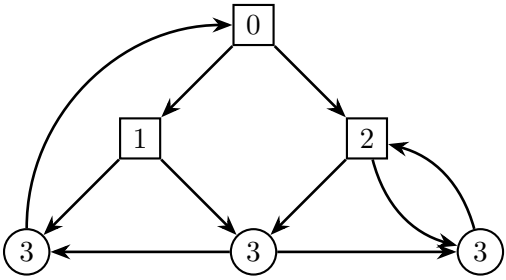
Figure 3.4a shows an example of a small parity game. We always draw vertices in $V_\diamond$ as circles and vertices in $V_\Box$ as rectangles to make them easily distinguishable. The numbers in the vertices are their priorities. Starting from the top vertex, player $\diamond$ wins this game: No matter what player $\Box$'s choices are, player $\diamond$ can ensure that the minimum priority visited infinitely often is either 0 or 3.

In the following definitions, let $P = (V, E, V_\diamond, \omega)$ be a parity game and i be a player.

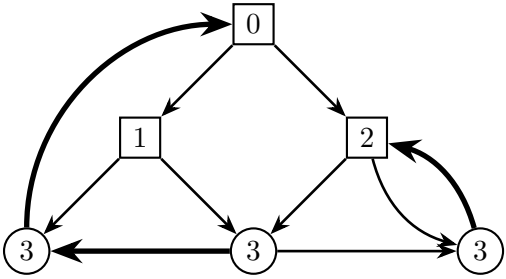
strategy

Definition 3.46 (Strategy) A *strategy* for player i is a partial function $\pi : V^*V_i \to V$ such that for all finite paths $(v_1, \ldots, v_k) \in V^*V_i$ with $\overline{v} \in \mathrm{dom}(\pi)$, we have $(v_k, \pi(\overline{v})) \in E$.⊣

So a strategy for player i selects a successor depending on the play up to the current point.



(a)



(b) With a strategy for player $\diamond$

Figure 3.4: A simple parity game

Definition 3.47 (π-Conforming Path) Let π be strategy for player i. We say that a path $\overline{v} = (v_1, v_2, \ldots) \in V^+ \cup V^\omega$ (finite or infinite) is *π-conforming* if $v_{k+1} = \pi(v_1, \ldots, v_k)$ holds for all its proper initial segments $(v_1, \ldots, v_k) \in V^* V_i$.

π-conforming

A *π-conforming play* is a maximal π-conforming path. ⊣

Definition 3.48 (Winning Strategy) A strategy π is *winning for player i from the vertex v* if every π-conforming play starting in v is winning for player i. ⊣

winning strategy

Definition 3.49 (Winning a Game) If there exists a strategy π winning for player i from a vertex v, then we say that player i *wins the game* (P, v).

i wins (P, v)

Similarly, if there exists a strategy π winning for player i from a set of vertices U, then we say that player i *wins on* U. ⊣

When we talk about strategies and do not explicitly mention the player, we assume that the strategy is meant for player $\diamond$.

3.2.2 Positional Determinacy

There is a special kind of strategy that only depends on the last vertex.

Definition 3.50 (Positional Strategy) A strategy π is *positional* if

$$\pi(v_1, \ldots, v_n) = \pi(w_1, \ldots, w_m)$$

holds for all paths $\overline{v}, \overline{w} \in \mathrm{dom}(\pi)$ with $v_n = w_m$. ⊣

We denote positional strategies for player i as functions $\pi : V_i \to V$. Positional strategies are computationally very nice because they have polynomial size. Moreover, it is possible to

check in polynomial time whether a positional strategy is a winning strategy.

Theorem 3.51 *There exists a polynomial-time algorithm that decides if a given positional strategy $\pi : V_i \to V$ is a winning strategy for player i for a given parity game (P, v).*

Proof. Let $\pi : V_i \to V$ be a strategy for player i. Without loss of generality, we assume $i = \diamond$. Let P_π be the game P restricted to the arcs conforming to π. That is, $P_\pi = (V, E_\pi, V_\diamond, \omega)$ with

$$E_\pi := \{(v, w) \in E \mid \text{if } v \in V_\diamond, \text{ then } \pi(v) = w\}.$$

Clearly, every π-conforming play in P is also a π-conforming play in P_π and vice-versa, so every vertex $v \in V$ has the same winner in P and P_π when restricted to π-conforming plays. Furthermore, P_π is polynomial-time computable from P and π.

Next, we observe that P_π is a one-player game for player $\square$. All player $\square$ needs to do in order to win from a vertex $v \in V$ is

1. find a path from v to a dead end $w \in V_\diamond$ or
2. find a path from v to a cycle $v_1, \ldots, v_k = v_1$ where $\min_{1 \leq i \leq k} \omega(v_i)$ is odd.

The first condition can trivially be checked in polynomial time. In order to check the second condition in polynomial time, one way to do it is to compute a function $f : V \to \{\top, \bot\}$ with

$$f(v) := \begin{cases} \top & \text{if there is a cycle } v = v_1, \ldots, v_k = v \text{ with} \\ & \text{minimum priority } \omega(v) \text{ and } \omega(v) \text{ is odd,} \\ \bot & \text{otherwise.} \end{cases}$$

The value $f(v)$ can be computed in polynomial time by removing all vertices w from P_π with $\omega(w) < \omega(v)$ and looking for a cycle with v on it. Then condition (2) can be checked by looking for a path from v to a vertex w with $f(w) = \top$. ∎

Definition 3.52 (Winning Region) Let P be a parity game.
W_i The *winning region* $W_i(P) \subseteq V$ of player i is the set of vertices from which the player has a positional winning strategy. ⊣

We usually write W_i instead of $W_i(P)$ if the game is obvious from the context. We observe that it is impossible for both players to have a winning strategy for the same vertex.

Observation 3.53 *It holds that* $W_\Diamond \cap W_\Box = \emptyset$.

The fat arrows in Figure 3.4b on page 49 are an example for a positional strategy for player $\Diamond$. This strategy is in fact a winning strategy, as the reader may easily verify. That we can find a positional winning strategy is no coincidence. The following result is well-known (see e.g., [Zie98]).

Theorem 3.54 *For every parity game P and vertex v there exists a positional winning strategy π for (P, v) for one of the players.*

A different, but equivalent, way of stating this theorem is the following.

Theorem 3.55 (Positional Determinacy) *For every parity game $P = (V, E, V_\Diamond, \omega)$ it holds that $V = W_\Diamond \cup W_\Box$.*

In Chapter 7, we will prove this theorem formally with the interactive theorem prover Isabelle/HOL.

3.2.3 Attractor Sets

Another common notion on parity games is the idea of *attractor sets*. The idea is that there are sets from which player $\Diamond$ cannot necessarily win the game, but from which player $\Diamond$ can force every play to behave in a certain way. Specifically, she can force every play to visit a given set of vertices without player $\Box$ being able to prevent this. We are going to use these sets extensively in Chapter 6.

Definition 3.56 We define the *one-step attractor* of $A \subseteq V$ for player i with basis $W \subseteq V$ as

one-step attractor

$$\begin{aligned}\text{direct-attr}_{i,W}(A) := W \cup A \\ \cup \{v \in V_i \mid \text{there is a } w \text{ with } v \to w \text{ and } w \in A\} \\ \cup \{v \in V_{\bar{i}} \mid \text{for all } w \text{ with } v \to w \text{ it holds that } w \in A \\ \text{and there is at least one such } w\}.\end{aligned}$$ ⊣

Observation 3.57 $\text{direct-attr}_{i,W} : 2^V \to 2^V$ *is monotone.*

With Theorem 3.8 on page 21, it follows that $\text{direct-attr}_{i,W}$ has a least and a greatest fixpoint. However, the greatest fixpoint is uninteresting because it is V. The least fixpoint is

$$\mu\, \text{direct-attr}_{i,W} = \bigcap \{A \subseteq V \mid \text{direct-attr}_{i,W}(A) \subseteq A\}.$$

We call the least fixpoint the *attractor set*.

Definition 3.58 Let $A \subseteq V$. We denote by $\text{attr}_i(A)$ the *i-attractor set* of A, defined as the least fixpoint of $\text{direct-attr}_{i,A}$. That is,

$\text{attr}_i(A)$

$$\text{attr}_i(A) := \mu\, \text{direct-attr}_{i,A}\,.$$ ⊣

Observation 3.59 *For all $A \subseteq V$, we have*

$$A \subseteq \operatorname{attr}_i(A) \subseteq V.$$

Similar to the semantics of the least fixpoint formulas $\mu X.\varphi$, we can also view an attractor set via its approximations.

$$\begin{aligned} \operatorname{attr}_i^0(A) &:= \emptyset \\ \operatorname{attr}_i^{\alpha+1}(A) &:= \text{direct-attr}_{i,A}(\operatorname{attr}_i^\alpha(A)) \\ \operatorname{attr}_i^\alpha(A) &:= \bigcup_{\beta<\alpha} \operatorname{attr}_i^\beta(A) \qquad \text{for limit ordinals } \alpha. \end{aligned}$$

Lemma 3.60 *For every parity game $P = (V, E, V_\Diamond, \omega)$, there exists an α such that for all $i \in \{\Diamond, \Box\}$ and for all $A \subseteq V$, we have* $\operatorname{attr}_i(A) = \operatorname{attr}_i^\alpha(A)$.

Proof. This is analogous to the proof of Lemma 3.11 on page 22.■

It is easy to see that for finite games P, choosing $\alpha = |V|$ is sufficient. This is because if we are not at a fixpoint yet, then every successor approximation step adds at least one vertex. So after at most $|V|$ steps, there are no more vertices left to add and we have a fixpoint.

Furthermore, $\text{direct-attr}_{i,W}(A)$ is computable in time $O(|V|+|E|)$ by iterating over all vertices in $V \setminus (W \cup A)$ and checking their outgoing arcs. It follows that $\operatorname{attr}_i(A)$ is computable in time $O(|V|^2 + |V||E|)$ by iterating $\text{direct-attr}_{i,A}$ for $|V|$ times. But there is a better algorithm that can compute $\operatorname{attr}_i(A)$ in time $O(|V| + |E|)$.

Algorithm 3.1: Compute the i-attractor set of A.

```
ATTR(P = (V, E, V_◇, ω), i, A)
    X ← A
    while X ≠ ∅ do
        A ← A ∪ X
        X' ← ∅
        foreach v ∈ X do
            foreach w with (w, v) ∈ E and w ∉ A ∪ X
             do
                if w ∈ V_i or (w, v) is the only outgoing arc
                 of w then
                    X' ← X' ∪ {w}
                else
                    Remove (w, v) from P
        X ← X'
    return A
```

Lemma 3.61 *Algorithm 3.1 on a parity game* $P = (V, E, V_\Diamond, \omega)$, $i \in \{\Diamond, \Box\}$, *and* $A \subseteq V$ *computes* $\mathrm{attr}_i(A)$ *and runs in time* $O(|V| + |E|)$.

Proof. For the running time, we observe that the inner most loop in total runs over each arc $(w, v) \in E$ at most once, and it either adds w to X' (and eventually, A) or it removes the arc. In both cases it never visits the arc (w, v) again. Furthermore, the sets X from different iterations of the outer **while** loop are disjoint, so the algorithm visits every vertex at most once. So the running time is bounded by $O(|V| + |E|)$.

Correctness follows from the observation that each iteration of the outer **while** loop computes $\text{direct-attr}_{i,A}(A)$, and $\text{direct-attr}_{i,W}(A) = \text{direct-attr}_{i,A}(A)$ for all $W \subseteq A \subseteq V$. ∎

Lemma 3.62 *For every* $A \subseteq V$, *player* i *has a positional strategy* π *such that every* π*-conforming play starting in* $\mathrm{attr}_i(A)$ *visits* A.

Proof. For finite games, it is easy to extract such a strategy from Algorithm 3.1. For infinite games, we need transfinite induction and a well-ordering of the set of all strategies. We will discuss this case in detail in Section 7.5.3 on page 235. ∎

attractor strategy

We call a strategy π as in the previous lemma an *attractor strategy* for $\mathrm{attr}_i(A)$.

Corollary 3.63 *For all* $i \in \{\Diamond, \Box\}$ *and all* $A \subseteq V$, *we have*

$$\mathrm{attr}_i(A) = \{v \in V \mid \textit{Player } i \textit{ has a strategy } \pi \textit{ such that every } \pi\textit{-conforming play starting in } v \textit{ visits } A\}.$$

Lemma 3.64 *For all $i \in \{\Diamond, \Box\}$, we have $\mathrm{attr}_i(W_i) = W_i$.*

Proof. If $v \in \mathrm{attr}_i(W_i) \setminus W_i$, then v would be winning for player i by her playing according to her attractor strategy and then switching to her winning strategy as soon as the play visits W_i. ∎

Computability in time linear in the number of arcs makes attractor sets algorithmically very nice. Together with lemmas 3.65 and 3.66, which we will prove next, this will be the basis of Chapter 6.

Lemma 3.65 *Let $P = (V, E, V_\Diamond, \omega)$ be a parity game, $i \in \{\Diamond, \Box\}$ and $A \subseteq V$. Then*

$$W_{\bar{i}}(P \setminus \mathrm{attr}_i(A)) \subseteq W_{\bar{i}}(P).$$

Proof. Let $P = (V, E, V_\Diamond, \omega)$ be a parity game, $i \in \{\Diamond, \Box\}$, $A \subseteq V$ and

$$v \in W_{\bar{i}}(P \setminus \mathrm{attr}_i(A)).$$

Then we have in particular

$$v \notin \mathrm{attr}_i(A).$$

If $v \notin W_{\bar{i}}(P)$, then player i has a winning strategy from v in P.

But player i cannot force the play into $\mathrm{attr}_i(A)$ or otherwise we would have $v \in \mathrm{attr}_i(A)$. So, if player $\bar{i}$ chooses so, then the play stays in $P \setminus \mathrm{attr}_i(P)$, where it was winning for player $\bar{i}$. So player $\bar{i}$ has a winning strategy from v in P, showing $v \in W_{\bar{i}}(P)$. ∎

Lemma 3.66 *Let $P = (V, E, V_\Diamond, \omega)$ be a parity game, $i \in \{\Diamond, \Box\}$ and $U \subseteq W_i(P)$. Then*

1. $W_i(P) = \mathrm{attr}_i(U) \cup W_i(P \setminus \mathrm{attr}_i(U))$ *and*
2. $W_{\bar{i}}(P) = W_{\bar{i}}(P \setminus \mathrm{attr}_i(U))$.

Proof.

1. Let $v \in W_i(P)$ and assume $v \notin \mathrm{attr}_i(U)$ and $v \notin W_i(P \setminus \mathrm{attr}_i(U))$. Because parity games are determined, we have $v \in W_{\bar{i}}(P \setminus \mathrm{attr}_i(U))$. By Lemma 3.65, this implies $v \in W_{\bar{i}}(P)$, which contradicts $v \in W_i(P)$.

2. Let $v \in W_{\bar{i}}(P)$. Then we have $v \notin \mathrm{attr}_i(U)$ because all of U is winning for player i. Suppose to the contrary that $v \notin W_{\bar{i}}(P \setminus \mathrm{attr}_i(U))$. Then player i can force the play into a dead end $v \in P \setminus \mathrm{attr}_i(U)$ that did not exist in P, the only difference between P and $P \setminus \mathrm{attr}_i(U)$. But a dead end in $P \setminus \mathrm{attr}_i(U)$ which is not a dead end in P has all its successors (at least one) in $\mathrm{attr}_i(U)$. This implies $v \in \mathrm{attr}_i(U)$, a contradiction.

   The other direction of the equality follows directly from Lemma 3.65. ∎

3.3 Relation to the Modal μ-Calculus

Parity games are relevant because they are the model-checking game for the modal μ-calculus. For every formula $\varphi \in L_\mu$ and every structure $\mathcal{M}$, there exists a parity game $\mathcal{M} \ltimes \varphi$ such that $\mathcal{M}, v \models \varphi$ holds if and only if player $\Diamond$ wins from

$(\varphi, v) \in \mathcal{M} \ltimes \varphi$. Furthermore, if $\mathcal{M}$ is finite, then this model-checking game can be computed from φ and $\mathcal{M}$ in polynomial time and satisfies $|V(\mathcal{M} \ltimes \varphi)| \in O(|V(\mathcal{M})||\varphi|)$.

3.3.1 Annotated Formulas

In the definition of the model-checking game, the most tricky part is to find the priorities, because they somehow need to capture the semantics of the fixpoint operators. For this, let us first go back to the modal μ-calculus and assign priorities to the fixpoint operators based solely on syntax. Recall that in consistent formulas, a fixpoint variable X_i is bound only by μ or only by ν, but not by both.

Definition 3.67 Let $\varphi \in L_\mu$ be consistent with $\overline{X} = (X_1, \ldots, X_n)$. We call $(p_1, \ldots, p_n) \in \mathbb{N}^n$ a *priority sequence* for $\overline{X}$ if it is strictly increasing and for all $1 \leq i \leq n$, p_i is odd if and only if X_i is bound in μ-subformulas. ⊣

priority sequence

The intuition is that we assign the priority p_i to every subformula that binds X_i. In order to illustrate this concept, we write the number p_i on top of the fixpoint operators binding X_i, as in $\overset{p_i}{\mu} X_i.\psi$. We call such a formula an *annotated formula*.

annotated formula

For example, a formula

$$\nu Y.\Diamond(\mu X.\nu Y.\Diamond X \vee \Diamond Y) \vee \Box Y$$

consistent with (X, Y) under the priority sequence $(1, 2)$ would be labeled as

$$\overset{2}{\nu} Y.\Diamond(\overset{1}{\mu} X.\overset{2}{\nu} Y.\Diamond X \vee \Diamond Y) \vee \Box Y.$$

Note that it cannot be labeled

$$\overset{2}{\nu}Y.\Diamond(\overset{3}{\mu}X.\overset{4}{\nu}Y.\Diamond X \vee \Diamond Y) \vee \Box Y,$$

even though these priorities would work in the model-checking game (that we are going to define in Definition 3.68). However, they violate the sequence (X, Y) and the priority sequence $(1, 2)$.

The first annotated formula is an element of $\mathrm{CL}(\overset{1}{\mu}X.\overset{2}{\nu}Y.\Diamond X \vee \Diamond Y)$. Due to Lemma 3.36 on page 40, it holds true in general that for an annotated formula φ and for some $\psi \in \mathrm{CL}(\varphi)$, we can use the annotation of φ for ψ as well.

3.3.2 Model-Checking Game

model-checking game

Definition 3.68 For a σ-structure $\mathcal{M}$ and an annotated formula $\varphi \in L_\mu[\sigma]$ in negation normal form, let $\mathcal{M} \ltimes \varphi = (V, V_\Diamond, E, \omega)$ be the *model-checking game* defined as follows.

$\mathcal{M} \ltimes \varphi$

$$V(\mathcal{M} \ltimes \varphi) := \mathcal{M} \times \mathrm{CL}(\varphi)$$

There is a an arc from (v, ψ) to (w, χ) if and only if

- $v = w$ and $\psi \in \{\chi \wedge \chi', \chi \vee \chi', \chi' \wedge \chi, \chi' \vee \chi\}$ for some χ' or
- $v = w$, $\psi \in \{\mu X.\chi', \nu X.\chi'\}$ and $\chi = \mathrm{closure}_\varphi(\chi')$ or
- $(v, w) \in E(\mathcal{M})$ and $\psi \in \{\Diamond\psi, \Box\psi\}$.

A vertex (v, ψ) is a $\Box$-node if and only if

- $\psi = \top$ or

- $\psi = P$ for some $P \in \sigma$ and $\mathcal{M}, v \models P$ or
- $\psi \in \{\chi \wedge \chi', \Box\chi\}$ for some $\chi, \chi' \in L_\mu[\sigma]$.

A vertex (v, ψ) has the priority

$$\omega(v,\psi) := \begin{cases} p & \text{if } \psi = \overset{p}{\mu} X.\chi \text{ or } \psi = \overset{p}{\nu} X.\chi \text{ for some } \chi \\ p' & \text{otherwise,} \end{cases}$$

where p' is the maximum priority. ⊣

See Figure 3.5 on the following page for an example. Note how most vertices in the example have priority 2, the maximum priority.

It is easy to show that this definition gives a well-defined model-checking game (see e.g., [Zap01]). However, the usual proofs from the literature do not apply directly to our definition of the model-checking game because our definition is slightly different. Usually, the set of vertices is not $\mathcal{M} \times \mathrm{CL}(\varphi)$, but $\mathcal{M} \times \mathrm{sub}'(\varphi)$, where $\mathrm{sub}'(\varphi)$ is the set of all non-indexed subformulas[1] of φ, with the additional assumption that in φ every fixpoint variable is quantified at most once. Then the game has additional arcs from (v, X) to $(v, \mu X.\psi)$, where $\mu X.\psi$ is the subformula which binds X. Furthermore, the non-trivial priorities are on the vertices of the form (v, X). These vertices do not exist in our game, due to taking the closure, so we push the priorities to the vertices of the form $(v, \mu X.\psi)$.

Let us prove that our game is also a correct model-checking game.

[1] Recall from Definition 3.23 on page 35 that $\mathrm{sub}(\varphi)$ excludes subformulas consisting of a single variable, and is indexed. This is why we use $\mathrm{sub}'(\varphi)$ here.

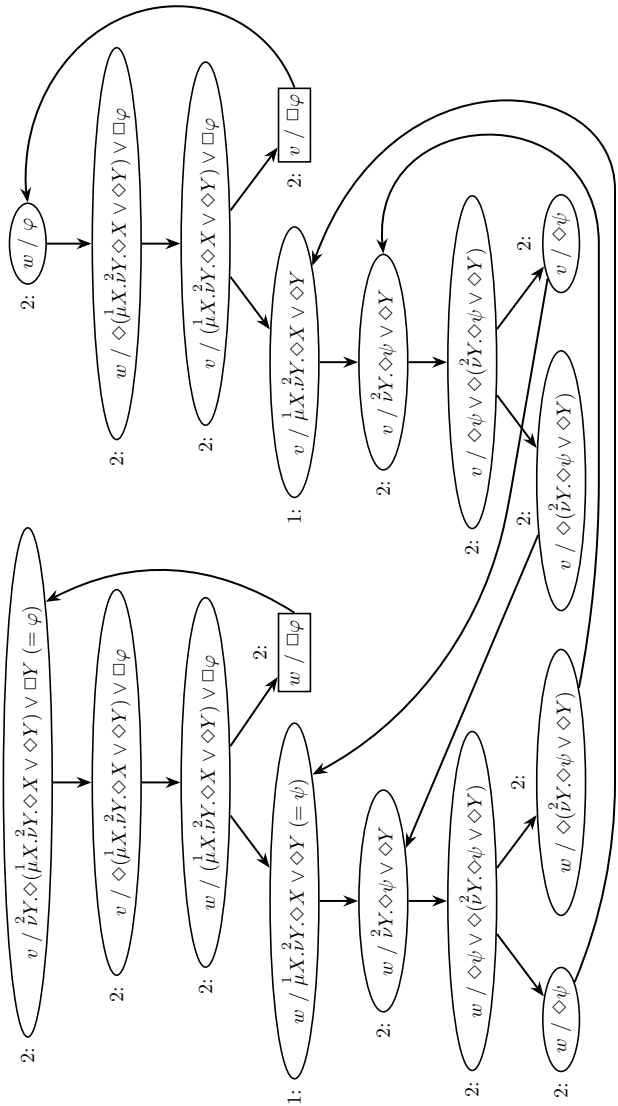


Figure 3.5: The model-checking game $\mathcal{M} \ltimes \varphi$ for $\varphi = \nu^2 Y.\diamond(\mu^1 X.\nu^2 Y.\diamond X \vee \diamond Y) \vee \Box Y$ and $\mathcal{M} = (\{v, w\}, \{(v, w), (w, v)\})$

Lemma 3.69 *Let $\varphi \in L_\mu[\sigma]$ be in negation normal form and let $\psi \in \mathrm{sub}(\varphi)$ such that ψ is not in the scope of a μ or ν fixpoint operator in φ. In particular, we have $\mathrm{closure}_\varphi(\psi) = \psi$ and hence $\psi \in \mathrm{CL}(\varphi)$.*

Then

$$(v, \psi) \in W_\Diamond(\mathcal{M} \ltimes \varphi) \iff (v, \psi) \in W_\Diamond(\mathcal{M} \ltimes \psi).$$

Proof. Let $\psi = (\psi, i)$ be the formula ψ together with its position in φ, and let $(\chi, j) \in \mathrm{sub}(\psi)$. Then we have $(\chi, j + i) \in \mathrm{sub}(\varphi)$. Furthermore, this satisfies

$$\pi_1(\mathrm{closure}_\varphi(\chi, j + i)) = \pi_1(\mathrm{closure}_\psi(\chi, j)),$$

where π_1 is the projection onto the first component (that is, π_1 forgets the index).

It follows that the subgraphs reachable from (v, ψ) in $\mathcal{M} \ltimes \psi$ and $\mathcal{M} \ltimes \varphi$ are isomorphic with the isomorphism

$$(v, \mathrm{closure}_\psi(\chi, j)) \mapsto (v, \mathrm{closure}_\varphi(\chi, j + i)). \qquad \blacksquare$$

Theorem 3.70 *For every $\varphi \in L_\mu[\sigma]$ in negation normal form and every σ-structure $\mathcal{M}, v$, it holds that*

$$\mathcal{M}, v \models \varphi \iff (v, \varphi) \in W_\Diamond(\mathcal{M} \ltimes \varphi).$$

Proof. We show this simultaneously for all σ by structural induction on φ.

($\varphi = P$, $P \in \sigma$) Let $v \in \mathcal{M}$ and $P \in \sigma$. By definition of the arcs, (v, P) has no successors. Furthermore, we have that $(v, P) \in V_\Box$ if and only if $\mathcal{M}, v \models P$. So (v, P) is winning for $\Diamond$ if and only if $\mathcal{M}, v \models P$, as required.

($\varphi = \neg P$, $P \in \sigma$) The argument is analogous for negated propositions.

($\varphi = \psi \vee \chi$) Let $v \in \mathcal{M}$. Then (v, φ) is a player $\Diamond$-vertex with the two successors (v, ψ) and (v, χ). Assume that $\mathcal{M}, v \models \varphi$ holds. Then $\mathcal{M}, v \models \psi$ or $\mathcal{M}, v \models \chi$, so by the induction hypothesis, we have $(v, \psi) \in W_\Diamond(\mathcal{M} \ltimes \psi)$ or $(v, \chi) \in W_\Diamond(\mathcal{M} \ltimes \chi)$. Using Lemma 3.69 and because (v, φ) is a player $\Diamond$-vertex, this implies $(v, \varphi) \in W_\Diamond(\mathcal{M} \ltimes \varphi)$.

On the other hand, if $(v, \varphi) \in W_\Diamond(\mathcal{M} \ltimes \varphi)$, then one of the successors must be winning for player $\Diamond$, too. Then, again by Lemma 3.69 and by the induction hypothesis, we have $\mathcal{M}, v \models \psi$ or $\mathcal{M}, v \models \chi$, so $\mathcal{M}, v \models \varphi$.

($\varphi = \psi \wedge \chi$) The case of conjunction is fully analogous to the case of disjunction.

($\varphi = \Diamond\psi$) Let $v \in \mathcal{M}$. Then (v, φ) is a $\Diamond$-vertex and has $(v_1, \psi), \ldots, (v_k, \psi)$ as its successors, where $v_1, \ldots, v_k$ are the successors of v in $\mathcal{M}$.

If $\mathcal{M}, v \models \Diamond\psi$, then there exists a successor v_i of v with $\mathcal{M}, v_i \models \psi$ and we have $(v_i, \psi) \in W_\Diamond(\mathcal{M} \ltimes \psi)$ by the induction hypothesis. This implies $(v, \varphi) \in W_\Diamond(\mathcal{M} \ltimes \varphi)$ because player $\Diamond$ can choose (v_i, ψ) as the successor of (v, φ), and the games $\mathcal{M} \ltimes \varphi$ and $\mathcal{M} \ltimes \psi$ are isomorphic if restricted to the vertices reachable from (v_i, ψ) due to Lemma 3.69.

If $v \in W_\Diamond(\mathcal{M} \ltimes \varphi)$, then player $\Diamond$ has a positional winning strategy. Using this positional strategy she chooses a successor (v_i, ψ) of $(v, \Diamond\psi)$ and wins, so we have $(v_i, \psi) \in W_\Diamond(\mathcal{M} \ltimes \varphi)$. Again, the games $\mathcal{M} \ltimes \varphi$ and $\mathcal{M} \ltimes \psi$ restricted to the vertices reachable from (v_i, ψ) are isomorphic by Lemma 3.69, so this implies $v_i \in W_\Diamond(\mathcal{M} \ltimes \psi)$. By the induction hypothesis, this gives us $\mathcal{M}, v_i \models \psi$, which implies $\mathcal{M}, v \models \varphi$ by the semantics of the $\Diamond$-operator.

($\varphi = \Box\psi$) This case follows analogously to the previous case.
($\varphi = \mu X.\psi$) Assume that $\mathcal{M}, v \models \varphi$. Let

$$F : V(\mathcal{M}) \to V(\mathcal{M}), \quad S \mapsto [\![\psi]\!]^{\mathcal{M}[X/S]}.$$

By the induction hypothesis, this is

$$F(S) = \{v \in \mathcal{M} \mid (v, \psi) \in W_\Diamond(\mathcal{M}[X/S] \ltimes \psi)\}.$$

We need to show that

$$\mu F = \{v \in \mathcal{M} \mid (v, \varphi) \in W_\Diamond(\mathcal{M} \ltimes \varphi)\}.$$

($\subseteq$): Let S be an abbreviation for the right-hand side of the equation. It suffices to show $F(S) \subseteq S$. Let $v_0 \in \mathcal{M}$ with $(v_0, \psi) \in W_\Diamond(\mathcal{M}[X/S] \ltimes \psi)$.

Let π be a winning $\Diamond$-strategy in $\mathcal{M}[X/S] \ltimes \psi$ for the winning region of player $\Diamond$. We want to apply this strategy to $\mathcal{M} \ltimes \mu X.\psi$. However, all the vertices of the form $(v, X) \in \mathcal{M}[X/S] \ltimes \psi$ do not exist in $\mathcal{M} \ltimes \mu X.\psi$. In their place, $\mathcal{M} \ltimes \mu X.\psi$ has vertices of the form $(v, \mu X.\psi)$ which did not exist in $\mathcal{M}[X/S] \ltimes \psi$. See Figure 3.6 for an illustration. Note that in the figure, the vertex (v, X) is a $\Diamond$-vertex if and only if $v \notin S$ by the definition of $\mathcal{M}[X/S] \ltimes \psi$.

As is evident from the figure, we can simply apply π directly to $\mathcal{M} \ltimes \mu X.\psi$ because there are no additional choices to be made by player $\Diamond$; every vertex $(v, \mu X.\psi)$ with $v \in \mathcal{M}$ has a single outgoing arc pointing to (v, ψ). We claim that this is a winning strategy on $\mathcal{M} \ltimes \mu X.\psi$ from $(v_0, \mu X.\psi)$. Suppose to the contrary that is not winning.

Then there exists a π-conforming play in $\mathcal{M} \ltimes \mu X.\psi$ starting from $(v_0, \mu X.\psi)$ winning for player $\Box$. The strategy π is

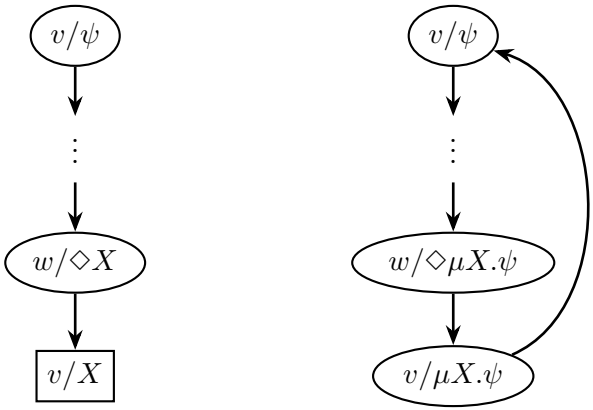


Figure 3.6: Comparing $\mathcal{M}[X/S] \ltimes \psi$ (left) to $\mathcal{M} \ltimes \mu X.\psi$ (right)

winning in $\mathcal{M}[X/S] \ltimes \psi$ from (v_0, ψ), but by assumption losing in $\mathcal{M} \ltimes \mu X.\psi$ from $(v_0, \mu X.\psi)$. The only way this can happen is that (after leaving the starting vertex) the play in $\mathcal{M} \ltimes \mu X.\psi$ eventually visits a vertex $(v, \mu X.\psi)$ with $S \in v$, because then (v, X) would be winning $\mathcal{M}[X/S] \ltimes \psi$ while $(v, \mu X.\psi)$ could be losing in $\mathcal{M} \ltimes \mu X.\psi$.

But $v \in S$ implies by the definition of S that player $\Diamond$ wins from $(v, \mu X.\psi)$ in $\mathcal{M} \ltimes \mu X.\psi$, which is a contradiction.

($\supseteq$): Let π be a winning $\Diamond$-strategy in $\mathcal{M} \ltimes \mu X.\psi$ for the winning region of player $\Diamond$ and let $v_0 \in S$. We apply the same strategy to $\mathcal{M}[X/\mu F] \ltimes \psi$ and claim that it is a winning strategy from (v_0, ψ).

Because μF is a fixpoint of F, we have

$$\mu F = F(\mu F) = \{v \in \mathcal{M} \mid (v, \psi) \in W_\Diamond(\mathcal{M}[X/\mu F] \ltimes \psi)\}$$

and it is sufficient to show $(v_0, \psi) \in W_\Diamond(\mathcal{M}[X/\mu F] \ltimes \psi)$.

The game $\mathcal{M} \ltimes \mu X.\psi$ differs from $\mathcal{M}[X/\mu F] \ltimes \psi$ only by having arcs from vertices of the form (v, X) (which we now call $(v, \mu X.\psi)$ due to taking the closure) back to (v, ψ). We claim that the same strategy π is winning for player $\Diamond$ in $\mathcal{M}[X/\mu F] \ltimes \psi$. Suppose to the contrary that it is not winning.

The only difference in the structures is that in the game $\mathcal{M}[X/\mu F] \ltimes \psi$ the outgoing arcs from vertices of the form (v, X) are missing. Only those (v, X) with $v \notin \mu F$ are winning for player $\Box$, so there exists a π-conforming path in $\mathcal{M}[X/\mu F] \ltimes \psi$ starting from (v_0, ψ) ending in some (v, X) with $v \notin \mu F$ or π would be winning.

So there exists a π-conforming path in $\mathcal{M} \ltimes \mu X.\psi$ starting $(v_0, \mu X.\psi)$ and ending in $(v, \mu X.\psi)$. It follows that $(v, \mu X.\psi)$ is winning for player $\Diamond$ in $\mathcal{M} \ltimes \mu X.\psi$ or π would not have been a winning strategy. But $\mathcal{M}, v \not\models \mu X.\psi$ implies $\mathcal{M}[X/\mu F], v \not\models \psi$, which means that (v, ψ) is winning for player $\Box$ in the game $\mathcal{M}[X/\mu F] \ltimes \psi$. So we can repeat this argument and find an infinite path in $\mathcal{M} \ltimes \mu X.\psi$ visiting an infinite number of vertices of the form $(v, \mu X.\psi)$. These all have odd priority, and this priority is the minimal priority in the game, so this play is losing for player $\Diamond$. This is a contradiction, so π is winning for player $\Diamond$ in $\mathcal{M}[X/\mu F] \ltimes \psi$ from (v, ψ). ∎

As described above the proof, our version of the model-checking game is slightly different from the usual definition of the model-checking game in the literature. We will now present the reason why we believe our definition is more useful for our purposes: The game $\mathcal{M} \ltimes \varphi$ is good not only for model-checking φ, but the same game also works for model-checking *every* $\psi \in \mathrm{CL}(\varphi)$.

To prove this, we need to strengthen Lemma 3.69.

Lemma 3.71 *Let $\varphi \in L_\mu[\sigma]$ be in negation normal form and let $\psi \in \mathrm{CL}(\varphi)$.*

Then

$$(v, \psi) \in W_\Diamond(\mathcal{M} \ltimes \varphi) \iff (v, \psi) \in W_\Diamond(\mathcal{M} \ltimes \psi).$$

Proof. Fix $\varphi \in L_\mu[\sigma]$ and $\psi \in \mathrm{CL}(\varphi)$. Let us revisit Definition 3.68, the definition of the model-checking game $\mathcal{M} \ltimes \varphi$, and observe that it works just as well if we use

$$V(\mathcal{M} \ltimes \varphi) := \mathcal{M} \times \mathrm{FLC}(\varphi)$$

instead of $\mathcal{M} \times \mathrm{CL}(\varphi)$. According to Lemma 3.29, $\mathrm{FLC}(\varphi)$ is $\mathrm{CL}(\varphi)$ without the indices, that is,

$$\mathrm{FLC}(\varphi) = \{\psi \mid (\psi, i) \in \mathrm{CL}(\varphi)\}.$$

Lemma 3.69 does not use the indices in an essential way (the isomorphism used in the proof then becomes the identity) and Theorem 3.70 never uses the indices, so both statements are also true under the modified model-checking games.

The modified games $\mathcal{M} \ltimes \varphi$ and $\mathcal{M} \ltimes \psi$ are identical when restricted to the vertices reachable from (v, ψ), which together with Theorem 3.70 proves the lemma. ∎

It follows that the game $\mathcal{M} \ltimes \varphi$ is the model-checking game for every $\psi \in \mathrm{CL}(\varphi)$.

Theorem 3.72 *For every $\varphi \in L_\mu[\sigma]$, every $\psi \in \mathrm{CL}(\varphi)$ and every σ-structure $\mathcal{M}, v$, it holds that*

$$\mathcal{M}, v \models \psi \iff (v, \psi) \in W_\Diamond(\mathcal{M} \ltimes \varphi).$$

Proof. By Theorem 3.70 and Lemma 3.71, we have

$$\begin{aligned} \mathcal{M}, v \models \psi &\iff (v, \psi) \in W_\Diamond(\mathcal{M} \ltimes \psi) \\ &\iff (v, \psi) \in W_\Diamond(\mathcal{M} \ltimes \varphi). \end{aligned}$$

■

4 Computational Complexity

In the previous chapter, we defined the modal μ-calculus and parity games. In this chapter we want to discuss the computational complexity of the problem of determining the winner in a parity game and of the model-checking problem of the modal μ-calculus. We will see that these problems are intimately related.

First, we will give an overview over existing algorithms for solving parity games, followed by the relation to the model-checking problem of the modal μ-calculus. Finally we will consider both problems on restricted classes of graphs and show that this is usually as hard as the general case.

Sections 4.1, 4.2 and 4.3.1 are a survey over known results. Sections 4.3.2 to 4.3.4 are new results.

4.1 The Complexity of Parity Games

Let us consider the complexity of the problem of determining the winner of a vertex in a given parity game. We saw in Theorem 3.51 on page 51 that it is possible to check in polynomial time whether a positional strategy is a winning strategy. It follows immediately that the problem of solving parity games is in NP because a nondeterministic machine can simply guess the winning strategy and then verify that the guess was correct.

Conversely, if player $\diamond$ does not have a winning strategy from a given vertex, then by Theorem 3.55, player $\square$ has a winning strategy. It follows that the problem is in $\mathrm{NP} \cap \mathrm{coNP}$.

In 1998, Marcin Jurdziński improved this and showed that the problem is in $\mathrm{UP} \cap \mathrm{coUP}$ [Jur98]. The class UP is defined as the class of NP-problems that have a nondeterministic Turing machine with a unique accepting path. That is, if the answer to an instance of a UP-problem is "yes", then the nondeterministic Turing machine has exactly one accepting path, and if the answer is "no", then it has no accepting path. Note that for an instance of an NP-problem, the non-deterministic polynomial-time machine may have more than one accepting path if the answer is "yes". Directly from this definition we see that $\mathrm{UP} \cap \mathrm{coUP} \subseteq \mathrm{NP} \cap \mathrm{coNP}$. It is an open question whether this inclusion is strict.

UP

Jurdziński proves membership in $\mathrm{UP} \cap \mathrm{coUP}$ via a polynomial-time reduction of parity games to *mean payoff games* followed by a polynomial-time reduction to *discounted mean payoff games*, and then proving that solving the latter games is in $\mathrm{UP} \cap \mathrm{coUP}$. A discounted mean payoff game is defined as follows.

disc. mean payoff game

Definition 4.1 A *discounted mean payoff game* $A = (V, E, V_\diamond, v, d, \lambda, \omega)$ is a directed graph (V, E) without dead ends and with $V_\diamond \subseteq V$, numbers $v, d \in \mathbb{N}$, $\lambda \in \mathbb{Q}$ and a function $\omega : E \to \{-d, \ldots, -1, 0, 1, \ldots, d\}$ assigning weights to the arcs.

Strategies and plays are defined as for parity games. A play $(v_1, v_2, \ldots)$ is winning for player $\diamond$ if and only if

$$(1 - \lambda) \sum_{i=1}^{\infty} \lambda^i \omega(v_{i-1}, v_i) \geq v.$$

⊣

Theorem 4.2 ([Jur98, Theorem 8]) *The problem of deciding the winner in a discounted mean payoff game, and thus the problem of deciding the winner in a mean payoff game and in a parity game, is in* $\mathrm{UP} \cap \mathrm{coUP}$.

Another famous problem contained in the class UP is integer factorization. This is the problem, given $n, k \in \mathbb{N}$, of deciding whether there exists a factor c of n with $1 < c < k$. Due to the fundamental theorem of algebra, every integer has a unique prime factorization if we fix the order of the factors. So it is easy to construct a non-deterministic Turing machine with at most one accepting path by guessing the unique prime factorization with factors in ascending order and then checking whether one of the primes is smaller than k.

Membership to $\mathrm{UP} \cap \mathrm{coUP}$ is the best result currently known for the problem of solving parity games. Let us remark that there is also a logical point of view under which we can consider the complexity of parity games, called the *descriptive complexity*. The question is, in what logics is it possible to define the winner of a parity game? Formally the task is to find a formula φ in some logic that is true at a vertex v in a parity game P (with the parity game presented suitably for the logic) if and only if player $\Diamond$ wins from (P, v) or prove that no such formula exists.

descriptive complexity

Similar to the computational complexity, very little is known about the descriptive complexity of parity games in general. By a result by Igor Walukiewicz which we will revisit in detail in Section 4.2, for every number $d \in \mathbb{N}$, there exists a formula $\varphi_d \in L_\mu$ that defines the winner of a parity game with at most d distinct priorities [Wal96].

In 2008, Anuj Dawar and Erich Grädel showed that with

an unbounded number of priorities, guarded second order logic (GSO) can define the winner, but least fixed point logic (LFP) is unable to do so [DG08]. Furthermore, they showed that on finite games, LFP can define the winner if and only if the winner is computable in polynomial time, relating the descriptive complexity to the computational complexity.

Dawar and Grädel also conjectured that monadic second order logic without quantifications over sets of arcs (called MSO_1) cannot define the winner of a parity game. This remains an open question.

4.1.1 General Algorithms

Although we only know that solving parity games belongs to $\mathrm{UP} \cap \mathrm{coUP}$, there are still numerous algorithms, not all of them with exponential running time. One of the first algorithms was invented by Robert McNaughton, who introduced more general infinite games and presented an exponential-time algorithm for solving these [McN93]. Wiesław Zielonka modified this algorithm, analyzing the memory requirements in detail and proving positional determinacy of parity games [Zie98]. Both algorithms have running time $O(n^d)$, where n is the number of vertices and d the number of distinct priorities in the parity game.

Another recursive algorithm is the algorithm by Perdita Stevens and Colin Stirling [SS98]. Oliver Friedmann showed that this algorithm also has exponential worst-case running time [Fri10].

In 1994, David E. Long, Anca Browne, Edmund M. Clarke, Somesh Jha, and Wilfredo R. Marrero significantly improved the running time to about $O(n^{d/2})$ [Lon+94]. Helmut Seidl im-

proved their result further by finding a far simpler proof [Sei96].

In 2000, Marcin Jurdziński presented an algorithm called *small progress measures* that runs in time

small progress measures

$$O\Big(m \cdot (n/\lfloor d/2 \rfloor)^{\lfloor d/2 \rfloor}\Big),$$

where m is the number of arcs [Jur00]. His algorithm is not based on the algorithms by Long et al. or by Seidl but matches their worst-case running time. The improvement in Jurdziński's algorithm is the space complexity, which is $O(dn)$.

Sven Schewe improved the running time of the small progress measure algorithm to roughly $O(mn^{d/3})$ [Sch07].

Marcin Jurdziński and Jens Vöge showed a different algorithm based on *strategy improvement*, improved later by Sven Schewe [VJ00; Sch08]. The idea is to start with some easily computable strategy (for example, an arbitrary strategy) and then improve it until no further improvements are possible. Provided that each improvement step can be done in polynomial time and that there is at most a polynomial number of improvement steps, this would give a polynomial-time algorithm.

strategy improvement

Luckily, the first part works: Each improvement step can be done in polynomial time. However, it was open for a long time whether the number of improvement steps is polynomial for any of these algorithms.

A few years later, Oliver Friedmann finally settled the question of the number of improvement steps negatively [Fri09; Fri11]. He showed that there exists a sequence of parity games on which the strategy improvement algorithm by Jurdziński and Vöge and the improved algorithm by Schewe both require

an exponential number of improvement steps, concluding that these algorithms are not polynomial time.

randomized

In [BSV03], Henrik Björklund, Sven Sandberg, and Sergei Vorobyov presented a *randomized subexponential* algorithm with running time roughly

$$\min\left(O(n^3 \cdot (1+n/d)^d), 2^{O(\sqrt{n \log n})}\right).$$

Their algorithm is based on work by Walter Ludwig on simple stochastic games and by Gil Kalai on linear programming [Lud95; Kal92].

subexponential

In contrast to the linear programming approach, Marcin Jurdziński, Mike Paterson, and Uri Zwick found a *deterministic subexponential* algorithm that is modification of McNaughton's algorithm [JPZ06]. The runtime of this subexponential algorithm is

$$n^{O(\sqrt{n})}.$$

If the outdegree of every vertex is bounded by 2, then the running time improves to

$$n^{O(\sqrt{n/\log n})}.$$

SAT

Another technique applied by Keijo Heljanko, Misa Keinänen, Martin Lange, and Ilkka Niemelä is to reduce the problem of solving parity games to a SAT problem [Hel+12]. Because solving parity games is in $\mathrm{UP} \cap \mathrm{coUP} \subseteq \mathrm{NP}$ and SAT is NP-complete, this is trivially possible by the definition of NP-completeness. However, the general reduction of NP-problems to SAT, although polynomial, is very inefficient. Heljanko et. al found that with a reduction specifically designed for

parity games, modern SAT solvers are surprisingly efficient at solving the instances.

In practice, there are also a few tricks that we can always use if we want to solve a parity game. All these tricks simplify common cases. The most well-known simplification is the observation that we can work on *strongly connected components* separately. If a game has more than one component, then we can solve the leaf components first and then work our way backwards. This works because there is no way that non-leaf components can affect the winning regions of leaf components (otherwise they would not be leaf components). For a discussion of this and other reductions, we refer the reader to the survey by Oliver Friedmann and to the documentation of *PGSolver*, a software for manipulating and solving parity games [FL09; Pgs]. PGSolver implements many of the algorithms described above, in particular the exponential recursive algorithm, the small progress measures algorithm, several strategy improvement algorithms, reductions to SAT, and the deterministic subexponential algorithm.

connected components

PGSolver

4.1.2 Algorithms on Restricted Classes

Because parity games proved to be very difficult in general, researchers have analyzed the problem on restricted classes of games, hopefully finding polynomial-time algorithms or better.

The first of many such projects was done by Jan Obdržálek who considered classes of parity games whose underlying undirected graph has bounded treewidth [Obd03]. He showed that for every such class, a polynomial-time algorithm exists.

Later, Dietmar Berwanger and Erich Grädel showed the same for classes of bounded *entanglement* [BG04]. Other results

in this setting are bounded clique-width [Obd07], bounded DAG-width [Ber+06], and (again) bounded treewidth [FS12] by John Fearnley and Sven Schewe, and (again) bounded treewidth and bounded clique-width [Gan15] by Moses Ganardi.

Fearnley and Schewe's result is an improvement over the result by Obdržálek in both the running time and in that they proved membership to the complexity class NC^2, a class of problems which can be efficiently parallelized.

Ganardi showed that parity games of bounded treewidth or bounded clique-width are in the complexity class LogCFL, the class containing all problems log-space reducible to a context-free language. LogCFL is a subclass of NC^2.

A reasonable conjecture could also be that parity games on undirected graphs are easier. However, it turns out that this is not the case. Dietmar Berwanger and Olivier Serre showed that undirected parity games are as hard as general parity games [BS12].

4.1.3 Fixed-parameter Tractability

Mostly for Chapter 5, but also for some earlier sections, we will need the notion of fixed-parameter tractability. For a good introduction and survey of parameterized complexity theory, we recommend the book by Jörg Flum and Martin Grohe [FG06]. We will use their definitions, which are standard.

alphabet
encoding procedure

The definitions are based on Turing machines, which require an *alphabet* and an *encoding procedure* for the problem instances. As is usual in complexity theory, the complexity classes are irrelevant of the choice of the alphabet and of the encoding procedure as long as they are chosen not to be exceedingly inefficient.

An example of an exceedingly inefficient encoding would be a unary encoding. Take for example $n \mapsto 01^n0$ for $n \in \mathbb{N}$, where 1^n is the letter 1 repeated n times. As you can see, we need to work hard to produce exceedingly inefficient encodings.

For this reason we will not concern ourselves with the choice of alphabets and encodings for the rest of this thesis. Nonetheless, here we will give the general definitions because it is unusual to define fixed-parameter tractability without referring to a specific alphabet and a specific encoding.

problem

Definition 4.3 Let Σ be a finite alphabet. A *problem* is a set $L \subseteq \Sigma^*$. ⊣

We will not repeat the definition of Turing machines because it is well-known, quite verbose, and irrelevant to our topic. We note that also the previously mentioned book by Flum and Grohe never bothers to define "Turing machines" and instead assumes from the beginning that the reader is familiar with the concepts of "algorithm" and "running times". This is a reasonable assumption because as with the choice of the alphabet and the encoding, the complexity classes are robust with respect to the exact formalization of Turing machines. We refer the reader to the book by Sanjeev Arora and Boaz Barak for a formal and excellent modern treatment of Turing machines [AB09].

Let us now define the fundamentals of parameterized complexity theory.

parameterization

Definition 4.4 A *parameterization* is a polynomial-time computable function $\Sigma^* \to \mathbb{N}$. A *parameterized problem* is a problem $L \subseteq \Sigma^*$ with a parameterization κ. ⊣

fixed-parameter tractable

Definition 4.5 A parameterized problem is *fixed-parameter tractable* (FPT) if there exists a computable function f, a polynomial p and an algorithm deciding the problem for all $x \in \Sigma^*$ in time $O(f(\kappa(x)) \cdot p(|x|))$, where $|x|$ is the length of x. ⊣

FPT-reducible

Definition 4.6 A parameterized problem $L \subseteq \Sigma^*$ with parameter κ is *FPT-reducible* to a parameterized problem $L' \subseteq (\Sigma')^*$ with parameter κ' if there exists a function $R : \Sigma^* \to (\Sigma')^*$ such that

1. For all $x \in \Sigma^*$ we have $x \in L \iff R(x) \in L'$.
2. R is computable by an FPT-algorithm with respect to κ. That is, there exists a computable function f, a polynomial p and an algorithm computing $R(x)$ in time $O(f(\kappa(x)) \cdot p(|x|))$ for all $x \in \Sigma^*$.
3. There is a computable function $g : \mathbb{N} \to \mathbb{N}$ with $\kappa'(R(x)) \leq g(\kappa(x))$ for all $x \in \Sigma^*$. ⊣

There are very few fixed parameter tractability results for parity games. While all algorithms mentioned in Section 4.1.2 are polynomial-time, the degree of the polynomial depends on the number of priorities. Gajarský et al. [Gaj+15] parameterized the problem by the size of a feedback vertex set, by the distance to a tournament, or by modular width and found that it is fixed-parameter tractable in these cases. In another recent paper, Matthias Mnich, Heiko Röglin, and Clemens Rösner showed fixed-parameter tractability for the parameter $|V_\diamond|$ [MRR16]. We will consider parity games on tournaments in detail in Chapter 6.

4.2 The L_μ Model-Checking Problem

Our prime example for a parameterized problem is the L_μ model-checking problem with parameter $|\varphi|$, the length of the formula. The motivation is that if this problem is fixed-parameter tractable, then we have an efficient model-checking algorithm for every fixed choice of φ. By the definition of FPT this would mean that the running time of such an algorithm is a polynomial in $\mathcal{M}$ with degree independent of φ. It is well-known that this problem is closely connected to parity games parameterized by the number of distinct priorities.

Theorem 4.7 *The L_μ model-checking problem parameterized by $|\varphi|$ is FPT equivalent to the problem of solving parity games parameterized by the number of distinct priorities d.*

Moreover, in the reduction from games to σ-structures we have $|\varphi| \in O(d)$, and the graph of the structure produced in the reduction is isomorphic to the graph of the input parity game.

Proof. The first direction is easy: The model-checking game $\mathcal{M} \ltimes \varphi$ is clearly computable in polynomial time, and Theorem 3.70 says that player $\Diamond$ wins $\mathcal{M} \ltimes \varphi$ from (v, φ) if and only if $\mathcal{M}, v \models \varphi$. So solving $\mathcal{M} \ltimes \varphi$ answers the model-checking problem.

Furthermore, this is an FPT reduction: The number of distinct priorities in $\mathcal{M} \ltimes \varphi$ only depends φ, and $\mathcal{M} \ltimes \varphi$ is computable in time $O(|\varphi|) \cdot |\mathcal{M}|^2$. The squaring is necessary because their could be a quadratic number of arcs in $\mathcal{M}$.

For the other direction we need to construct an FPT-reduction of the problem of solving parity games to the model-checking problem. Every parity game $P = (V, E, V_\Diamond, \omega)$ can be considered as a σ-structure with $\sigma = \{V_\Diamond, P_0, P_1, \ldots, P_d\}$,

where d is the maximum priority. We can assume that d is the parameter because without loss of generality, the number of distinct priorities cannot be significantly lower than d.

For the proposition symbols, $P, v \models P_i$ holds if and only if $\omega(v) = i$. Then player $\Diamond$ wins from $v \in V$ if and only if

$$P, v \models \varphi_d$$

with

$$\varphi_d := \nu X_0 \mu X_1 \nu X_2 \dots \lambda X_d \bigvee_{i=0}^{d} \big((V_\Diamond \wedge P_i \wedge \Diamond X_i) \vee (\neg V_\Diamond \wedge P_i \wedge \Box X_i)\big),$$

where $\lambda = \mu$ if d is odd and $\lambda = \nu$ if d is even. The idea is that if $v \in V_\Diamond$, then it should be player $\Diamond$'s turn in the model-checking game $P \ltimes \varphi_d$, so we use a $\Diamond$ modality. Additionally, if $\omega(v) = i$, then we need to pass through priority i in $P \ltimes \varphi_d$. This is guaranteed by passing through X_i because the formula can be annotated as follows.

$$\overset{0}{\nu} X_0 \overset{1}{\mu} X_1 \overset{2}{\nu} X_2 \dots \overset{d}{\lambda} X_d \bigvee_{i=0}^{d} \big((V_\Diamond \wedge P_i \wedge \Diamond X_i) \vee (\neg V_\Diamond \wedge P_i \wedge \Box X_i)\big),$$

For a detailed proof, we refer the reader to Section 3.3.6 of the book on finite model theory by Grädel et. al [Grä+07]. The original proof, although written for monadic second order logic, is by Igor Walukiewicz [Wal96].

This is an FPT-reduction because $d \leq |V|$, so the σ-structure

P is computable in time polynomial in $|V|$, and φ_d depends only on d. Furthermore, we have $|\varphi| \in O(d)$ as claimed. ∎

Without parameters, we get the classical version of the equivalence.

Corollary 4.8 *The L_μ model-checking problem is polynomial-time equivalent to solving parity games.*

As of 2016, it is unknown for the L_μ model-checking problem and for the problem of solving parity games if they are fixed-parameter tractable in general for the usual parameters.

On the other hand, at the end of Section 4.1.3 we saw FPT results for other parameters such as feedback vertex set. In the same spirit of choosing a different parameter, we will show in Section 5.4 on page 147 that the problem is fixed-parameter tractable with respect to the parameter $|\varphi| + k$, where k is the width of certain decompositions (Kelly decomposition or DAG decomposition), assuming the decomposition is part of the input.

4.3 Difficulty of Restricted Classes

The L_μ model-checking problem is the problem of determining $\mathcal{M} \models \varphi$, given a σ-structure $\mathcal{M}$ and $\varphi \in L_\mu[\sigma]$. Parity games are polynomial-time solvable on many restricted classes of graphs, as we saw in Section 4.1.2. However, these results do not transfer to the L_μ model-checking problem. The reason is that the reduction in Theorem 4.7 from the model-checking problem to parity games blows up most width measures such as treewidth.

In this sense, solving the L_μ model-checking problem on restricted classes of graphs is a worthy question independent of the problem of solving parity games. We will study this problem in depth in Chapter 5. Let us here discuss some reductions.

4.3.1 Hardness

In 2006, Nicolas Markey and Philippe Schnoebelen in [MS06] showed that L_μ model checking is reducible to L_μ model checking on cycles.

Theorem 4.9 ([MS06, Theorem 3.1]) *L_μ model checking logspace reduces to L_μ model checking of cycles.*

Nearly all measures of graph complexity (treewidth, DAG-width, etc.) are small on cycles. We get the following corollary.

Corollary 4.10 *Let $\mathcal{C}$ be a class of structures containing all cycles for all signatures (for example, structures of bounded treewidth, or of bounded DAG-width, etc.).*

If there exists a polynomial-time algorithm solving the L_μ model-checking problem on $\mathcal{C}$, then the general L_μ model-checking problem is polynomial-time solvable.

This means that short of solving the general L_μ model-checking problem in polynomial time, it is impossible to solve restricted classes in polynomial time. However, this is not as bad as it sounds. Fixed-parameter tractability (FPT) algorithms for restricted classes are still possible and interesting ways of attacking the problem, because Markey's and Schnoebelen's reduction is not FPT.

Furthermore, this obstruction does not apply to parity games. Markey's and Schnoebelen's reduction blows up the cycle, the signature and the formula linearly with the size of $\mathcal{M}$, which in term blows up the model-checking game. While parity games can be reduced to some restricted classes such as classes of locally bounded treewidth (as we will show in Section 4.3.4), no reductions to classes such as planar parity games or games of bounded treewidth or bounded clique-width are known. Of course, a polynomial-time reduction to classes of bounded treewidth or bounded clique-width would imply a polynomial-time algorithm for solving parity games.

A polynomial-time reduction to planar parity games seems more reasonable to expect, as planar parity games are not known to be easier than general parity games. However, we will see in the following that a simple reduction by replacing arc crossings with some gadget does not work.

4.3.2 Planar Graphs

In contrast to parity games, where the complexity of solving planar games is unknown, the L_μ model-checking problem is as hard on planar graphs as on general graphs. We prove this by constructing a polynomial-time reduction of the general problem to the planar problem.

Theorem 4.11 *There exist polynomial-time functions f, g, h such that for all finite σ-structures $\mathcal{M}$, all $v \in \mathcal{M}$ and all $\varphi \in L_\mu[\sigma]$, it holds that*

1. *$\mathcal{M}, v \models \varphi$ if and only if $f(\mathcal{M}), v \models g(\varphi)$*
2. *$f(\mathcal{M})$ is a planar $h(\sigma)$-structure.*

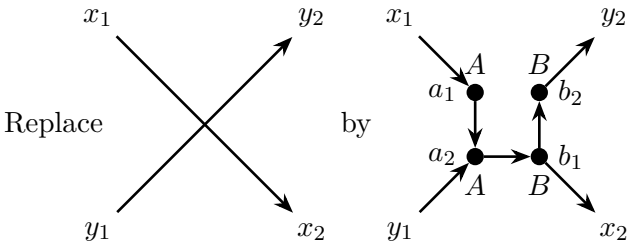


Figure 4.1: Arc crossing gadget

3. $g(\varphi) \in L_\mu[h(\sigma)]$.

4. $\sigma \subseteq h(\sigma)$.

Proof. Let $A, B \notin \sigma$ be fresh proposition symbols and define $h(\sigma) := \sigma \cup \{A, B\}$. The idea for defining $f(\mathcal{M})$ is to find an arbitrary embedding of $\mathcal{M}$ into the plane, possibly with arc crossings (but only with at most one crossing per pair of arcs and at most two arcs crossing in the same point), and then replace each arc crossing with a small planar gadget, depicted in Figure 4.1. Thus each arc crossing introduces four new vertices, two vertices where only A holds and two where only B holds. A and B are false everywhere else. Clearly, the resulting $h(\sigma)$-structure $f(\mathcal{M})$ is planar because the arc crossing gadget is planar.

It remains to define $g(\varphi)$. Every $\Box$ and $\Diamond$ may now need to skip over an unbounded number of arc crossing gadgets. This is possible by replacing every subformula $\Diamond\psi$ of φ (and analogously, $\Box\varphi$), with a suitable fixpoint formula.

Let Z be a fixpoint variable not occurring in φ. We define

$g(\varphi) := \widetilde{\varphi}$ inductively as follows.

$$\widetilde{\top} := \top \qquad \widetilde{\bot} := \bot$$
$$\widetilde{P} := P \quad \text{for all } P \in \sigma \qquad \widetilde{X} := X \quad \text{for all } X \in \mathrm{Var}$$
$$\widetilde{\varphi \wedge \psi} := \widetilde{\varphi} \wedge \widetilde{\psi} \qquad \widetilde{\varphi \vee \psi} := \widetilde{\varphi} \vee \widetilde{\psi}$$
$$\widetilde{\neg\varphi} := \neg\widetilde{\varphi}$$
$$\widetilde{\mu X.\varphi} := \mu X.\widetilde{\varphi} \qquad \widetilde{\nu X.\varphi} := \nu X.\widetilde{\varphi} \quad \text{for all } X \in \mathrm{Var}$$

So far, g has no effect. For the modal operators, this is different.

$$\widetilde{\Diamond\varphi} := \Diamond\mu Z.(\neg A \wedge \widetilde{\varphi}) \vee \Big(A \wedge \Diamond\Diamond\Diamond(\neg B \wedge Z)\Big)$$
$$\widetilde{\Box\varphi} := \Box\mu Z.(\neg A \wedge \widetilde{\varphi}) \vee \Big(A \wedge \Diamond\Diamond\Diamond(\neg B \wedge Z)\Big).$$

Clearly, $\widetilde{\varphi}$ is polynomial-time computable from φ.

We claim that $f(\mathcal{M}), v \models \widetilde{\varphi}$ if and only if $\mathcal{M}, v \models \varphi$. Let $x_1 \to z$ be an arc in $\mathcal{M}$. In $f(\mathcal{M})$, this arc may have an unbounded, but finite, number of arc crossing gadgets.

Let us consider the model-checking games $P := \mathcal{M} \ltimes \varphi$ and $P' := \mathcal{M} \ltimes \widetilde{\varphi}$. We are going to translate strategies for player $\Diamond$ between these two games. By the definition of $\widetilde{\varphi}$, the games only differ in the vertices corresponding to $\Diamond$ and $\Box$ subformulas of φ.

Let τ be a $\Diamond$-strategy on P. Let us consider a vertex of the form $(v, \Diamond\psi) \in V(P)$, and choose $w \in V(\mathcal{M})$ such that $\tau((v, \Diamond\psi)) = (w, \psi)$ (in particular, $(v, w) \in E(\mathcal{M})$). Let us introduce the abbreviation

$$\chi := \mu Z.(\neg A \wedge \widetilde{\psi}) \vee \Big(A \wedge \Diamond\Diamond\Diamond(\neg B \wedge Z)\Big),$$

such that $\Diamond\chi = \widetilde{\Diamond\psi}$.

We claim that player $\Diamond$ then has a strategy to end up in $(w, \widetilde{\psi}) \in P'$, starting from $(v, \Diamond\chi) \in P'$. The strategy is simple: Player $\Diamond$ has a strategy to pass over an arc crossing gadget in the $x_1 \to x_2$ direction as follows.

$$
\begin{aligned}
&(x_1, \Diamond\chi),\\
&(a_1, \chi),\\
&\Big(a_1, (\neg A \wedge \widetilde{\psi}) \vee \big(A \wedge \Diamond\Diamond\Diamond(\neg B \wedge \chi)\big)\Big),\\
&\big(a_1, A \wedge \Diamond\Diamond\Diamond(\neg B \wedge \chi)\big).
\end{aligned}
$$

Here it is player $\Box$'s turn, but his choice is forced if he does not want to lose immediately, because we have $\mathcal{M}, a_1 \models A$.

$$
\begin{aligned}
&\big(a_1, \Diamond\Diamond\Diamond(\neg B \wedge \chi)\big),\\
&\big(a_2, \Diamond\Diamond(\neg B \wedge \chi)\big),\\
&\big(b_1, \Diamond(\neg B \wedge \chi)\big),\\
&(x_2, \neg B \wedge \chi).
\end{aligned}
$$

Again, it is player $\Box$'s turn, but $\mathcal{M}, x_2 \not\models B$, so the choice is forced, and we end in

$$(x_2, \chi).$$

Similarly, player $\Diamond$ has a strategy to pass over a gadget in the $y_1 \to y_2$ direction. Furthermore, the strategies for the two directions are compatible, because they apply to disjoint sets

of vertices.

So player $\Diamond$ can pass over a single gadget, and thus over any finite number of gadgets, in the $x_1 \to x_2$ or in the $y_1 \to y_2$ direction. Eventually, player $\Diamond$ has passed over all gadgets on the arc $v \to w$ and arrives at (w, χ). From (w, χ), she moves to

$$\Big(w,\, (\neg A \wedge \widetilde{\psi}) \vee \big(A \wedge \Diamond\Diamond\Diamond(\neg B \wedge Z)\big)\Big),$$

$$\Big(w,\, \neg A \wedge \widetilde{\psi}\Big),$$

where we have $\mathcal{M}, w \not\models A$, so player $\Box$ will move to:

$$\Big(w,\, \widetilde{\psi}\Big),$$

which shows that player $\Diamond$ has the claimed strategy.

Is is easy to see that the above strategies are the only strategies on these gadgets for player $\Diamond$ and for player $\Box$ where they do not immediately lose. This implies the other direction: Every $\Diamond$-strategy on P' can be translated to a $\Diamond$-strategy in P, and the same for $\Box$-strategies.

Furthermore, our translation of strategies preserves dead ends and priorities that are visited along strategy-conforming paths because the priorities in the gadgets are irrelevant. Thus the winning regions (restricted to the parts outside the gadgets) are the same in P and in P'. So we have $\mathcal{M}, v \models \varphi$ if and only if $f(\mathcal{M}), v \models \widetilde{\varphi}$. ∎

Corollary 4.12 *The L_μ model-checking problem is polynomial-time and FPT-reducible to the L_μ model-checking problem on planar graphs, with both problems parameterized by $|\varphi|$.*

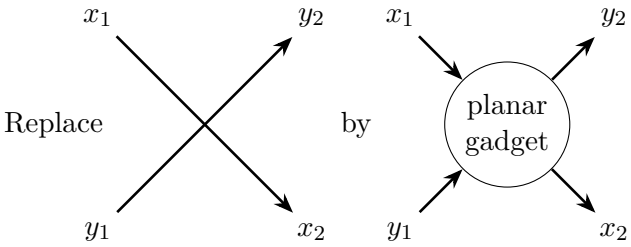


Figure 4.2: An arc crossing gadget in a parity game (impossible)

Proof. FPT-reducibility follows from the observation that $\widetilde{\varphi}$ in the proof of the previous theorem does not depend on $\mathcal{M}$, and $f(\mathcal{M})$ does not depend on φ. ∎

Let us point out the difference between Corollary 4.12 and Markey and Schnoebelen's construction (Theorem 4.9). In their reduction, the translated formula grows with $\mathcal{M}$. In contrast to this, in our reduction to planar graphs the formula $\widetilde{\varphi}$ does not depend on the structure. This is the reason our reduction is FPT, and also why our reduction does not follow from Theorem 4.9.

Let us conclude our remarks by showing that for parity games, the situation is not this easy. An arc crossing gadget as depicted in Figure 4.2 turns out to be impossible for parity games.

Theorem 4.13 *There exists in general no planar gadget to replace an arc crossing in a parity game.*

Proof. Suppose to the contrary that there exists such a gadget as depicted in Figure 4.2. Fix positional strategies for players

$\Diamond$ and $\Box$, winning on their winning regions, such that entering the planar gadget at x_1 leads to leaving the gadget at x_2, and entering at y_1 leads to leaving it at y_2.

Because we have fixed positional strategies for both players, there exists exactly one strategy-conforming path from x_1 to x_2 through the gadget and exactly one strategy-conforming path from y_1 to y_2. Because the gadget is planar, these paths intersect in at least one vertex v. But we have positional strategies, so after v the path cannot go to both x_2 and y_2, which contradicts our assumption. ∎

4.3.3 Treewidth

Another way to restrict classes is to bound their *treewidth*. We already mentioned in Section 4.1.2 the existence of polynomial-time algorithms for classes of parity games with bounded treewidth. However, for μ-calculus model checking, no such algorithms are known. Here we will present a reason why the polynomial time algorithms for solving parity games of bounded treewidth cannot be applied to the model-checking problem of the modal μ-calculus.

Let us first define treewidth as defined by Neil Robertson and Paul D. Seymour, as well as the closely related concept of a graph minor [RS86].

Definition 4.14 A *tree decomposition* of an undirected graph $G = (V, E)$ is a tree $\mathcal{T}$ with a map $\beta : V(\mathcal{T}) \to 2^V$ such that

tree decomposition

1. $\bigcup_{t \in V(\mathcal{T})} \beta(t) = V$.
2. For all $(v, w) \in E$ there exists a $t \in V(\mathcal{T})$ with $v, w \in \beta(t)$.

3. For all $s, t, u \in V(\mathcal{T})$ such that t is on the unique path from s to u, it holds that $\beta(s) \cap \beta(u) \subseteq \beta(t)$.

treewidth The *width* of $(\mathcal{T}, \beta)$ is $\max_{t \in V(\mathcal{T})} |\beta(t)| - 1$. The *treewidth* $\mathrm{tw}(G)$ of G is the minimal width of any of its tree decompositions. ⊣

Definition 4.15 For an undirected graph $G = (V, E)$, we call minor H a *minor* of G if H can be constructed from G by deleting edges or vertices and by contracting edges. ⊣

It is well-known and easy to check that treewidth is closed under minors.

Lemma 4.16 ([RS86]) *If H is a minor of G, then* $\mathrm{tw}(H) \leq \mathrm{tw}(G)$.

Consider a σ-structure $\mathcal{M}$ of small treewidth and a formula $\varphi \in L_\mu[\sigma]$. It may be reasonable to expect that the treewidth of the underlying undirected graph of the parity game $\mathcal{M} \ltimes \varphi$ could maybe be a little larger than the treewidth of $\mathcal{M}$, but not by much. Unfortunately, it turns out that the treewidth of $\mathcal{M} \ltimes \varphi$ has no relationship with the treewidth of $\mathcal{M}$.

Theorem 4.17 *Let σ be a signature. Then there exists a sequence $\varphi_1, \varphi_2, \ldots$ of L_μ formulas with $|\varphi_n| \in O(n^2)$ such that for every σ-structure $\mathcal{M}$ and $n > 0$ we have* $\mathrm{tw}(\mathcal{M} \ltimes \varphi_n) \geq n$.

Proof. We define

$$\varphi_n := \mu X_1 . \mu X_2 . \cdots \mu X_n . \bigwedge_{i=1}^{n} \bigvee_{i=1}^{n} X_i$$

Let $v \in \mathcal{M}$ and assume without loss of generality that $V(\mathcal{M}) = \{v\}$. The model-checking game $\mathcal{M} \ltimes \varphi_i$ then looks

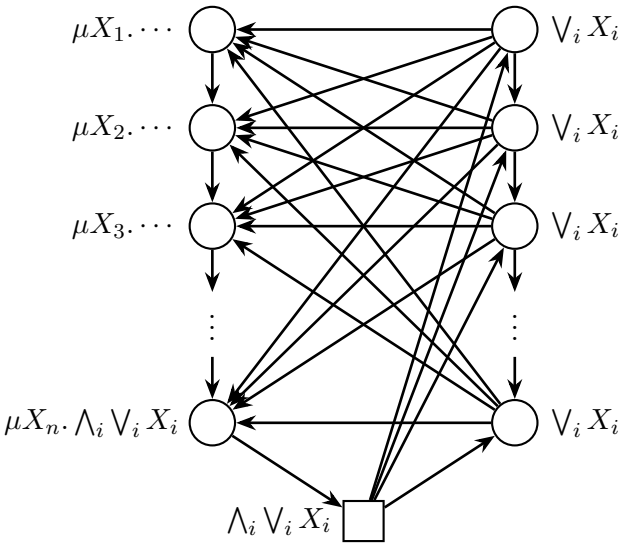


Figure 4.3: A model-checking game with unbounded treewidth

similar to the game shown in Figure 4.3. We simplified this game a little by writing X_i when in reality X_i is replaced by its binding formula. We also merged long chains of conjunctions into one conjunction, and the same for chains of disjunctions. It is easy to see that the graph in Figure 4.3 can be built from $\mathcal{M} \ltimes \varphi_i$ by arc contractions.

As the figure indicates, the underlying undirected graph contains a complete bipartite graph $K_{n,n}$ as a subgraph. Because all we did in order to get to the graph in Figure 4.3 was to contract edges, this means that the underlying undirected graph of $\mathcal{M} \ltimes \varphi$ has a complete bipartite graph $K_{n,n}$ as a minor. It is well-known that $\mathrm{tw}(K_{n,n}) = n$ [Klo94, Lemma 2.2.1]. So it follows with Lemma 4.16 that $\mathrm{tw}(\mathcal{M} \ltimes \varphi) \geq n$. ∎

We conclude that the reduction from structures to parity games in Theorem 4.7 is not an FPT reduction if we include treewidth in the parameter. This means that efficient algorithms on classes of parity games of bounded treewidth cannot be transferred to the model-checking problem of the modal μ-calculus without adding φ to the parameter.

4.3.4 Locally Bounded Treewidth

We saw in Section 4.3.2 that solving the model-checking problem of the modal μ-calculus on planar structures is as hard as solving it on general structures. We also saw that for parity games it is unknown whether solving planar games is as hard as general games. One approach to solving planar parity games could be to work with their property of having locally bounded treewidth.

Intuitively a class of graphs has locally bounded treewidth

if the treewidth of a neighborhood around a vertex depends only on the size of the neighborhood and not on the graph. As it turns out, locally bounded treewidth is not enough to make parity games easier. In this section we will show that if we can solve all classes of locally bounded treewidth in polynomial time, then we can solve parity games in general in polynomial time.

For an undirected graph $G = (V, E)$ and $v, w \in V$, let $d(v, w)$ denote the *distance* between v, w. That is, $d(v, w) = n$ if and only if $v = v_0, v_1, \ldots, v_n = w$ is a shortest path from v to w. If v and w are in different components, we write $d(v, w) = \infty$ with the property that $n < \infty$ for all $n \in \mathbb{N}$.

distance

Definition 4.18 For an undirected graph $G = (V, E)$ and $v \in V$, $r \in \mathbb{N}$, let

$$N_r^G(v) := \{w \in V \mid d(v, w) \leq r\}$$

be the *r-neighborhood of v*, the vertices of distance at most r from v. For directed graphs, we consider the underlying undirected graph. As usual, we omit the index G if it is clear from the context. ⊣

r-neighborhood of v

Definition 4.19 A class $\mathcal{C}$ of graphs has *locally bounded treewidth* if there exists a function $f : \mathbb{N} \to \mathbb{N}$ such that for all $r \in \mathbb{N}$, $G \in \mathcal{C}$, $v \in V(G)$, it holds that

$$\mathrm{tw}(N_r(v)) < f(r). \qquad \dashv$$

locally bounded treewidth

Example 4.20 The class of planar graphs has locally bounded treewidth with $f(r) = 3r + 1$ ([RS84, Section 2]). More generally, every class of graphs excluding a fixed apex graph as a

minor has treewidth locally bounded by $f(r) = cr + d$ for some constants $c, d \in \mathbb{N}$ depending on the class, as Erik Demaine and Mohammad Hajiaghayi show in [DH04]. A graph is an *apex graph* if it has a vertex whose removal makes the graph planar.

Theorem 4.21 *If for all classes $\mathcal{C}$ of parity games of locally bounded treewidth there exists a polynomial-time algorithm for solving $\mathcal{C}$, then there exists a polynomial-time algorithm for solving parity games in general.*

Proof. Let $\mathcal{C}$ be the class of all parity games. We transform $\mathcal{C}$ into a class $\mathcal{C}'$ of locally bounded treewidth as follows. For every parity game $P \in \mathcal{C}$, subdivide every arc of P into n segments, where n is the number of vertices in P, and assign the new vertices an irrelevant priority (that is, the maximum priority). Call the subdivided game P' and add it to $\mathcal{C}'$. Clearly, $V(P') \leq n^3$ because $E(P) \leq n^2$. Furthermore, solving P' solves P because subdividing arcs does not change the winner if we use irrelevant priorities for the additional vertices.

It remains to show that $\mathcal{C}'$ has locally bounded treewidth. Let $r \in \mathbb{N}$ be arbitrary and let $P \in \mathcal{C}$, $n = |V(P)|$, $P' \in \mathcal{C}'$ be as above. Let $v \in V(P')$. If $r < n$, then $\operatorname{tw}\big(N_r^{P'}(v)\big) = 1$ because $N_r^{P'}(v)$ is a tree. If $r \geq n$, then

$$\operatorname{tw}\big(N_r^{P'}(v)\big) \leq |V(P')| \leq n^3 \leq r^3.$$

Together we have $\operatorname{tw}\big(N_r^{P'}(v)\big) \leq r^3$ for all $r \in \mathbb{N}$, which proves that $\mathcal{C}'$ has locally bounded treewidth. ∎

This theorem shows that solving parity games of locally bounded treewidth is just as hard as solving general parity

games. In particular, hopes for a polynomial time algorithm for planar parity games cannot be based on exploiting only the locally bounded treewidth of planar graphs, because this would solve all parity games.

5 L_μ Types

> We shall investigate the relation between the elementary properties possessed by the product algebraic system and those possessed by its factors.
>
> *(Solomon Feferman and Robert L. Vaught, [FV59])*

5.1 A Feferman-Vaught Theorem

A famous result by Solomon Feferman and Robert L. Vaught known as the *Feferman-Vaught Theorem* shows how to evaluate a first-order formula on a generalized product of structures by evaluating sets of formulas on the individual factors and then combining the results [FV59]. This theorem is widely used in model theory. For applications and more background on this classical result, we refer the reader to the survey by Johann A. Makowsky [Mak04]. Our goal in this chapter is to obtain a similar result for the modal μ-calculus, where no such theorem was known.

One issue preventing a theorem similar to the Feferman-Vaught Theorem for the modal μ-calculus is that in L_μ we cannot identify individual vertices due to invariance under bisimilarity. So by evaluating formulas on a vertex in a small part of a structure we cannot say much about how this vertex behaves in a larger structure.

Our approach is to give every vertex, where necessary, a

unique label. We can do this with only a small number of labels, and thus get the following theorem. Here the *type* of a vertex is a set of formulas true at that vertex.

Theorem 5.1 *Let $\mathcal{M} = \mathcal{M}_1 \cup \mathcal{M}_2$ be a structure such that every vertex in $\mathcal{M}_1 \cap \mathcal{M}_2$ has a unique label and there are no arcs from $\mathcal{M}_2$ to $\mathcal{M}_1$ except via the intersection (see Figure 5.1 on page 102).*

Then the type of every vertex in $\mathcal{M}_1$ can be computed from $\mathcal{M}_1$ and the types of the vertices in $\mathcal{M}_2$.

In this section we will make these notions precise, and in particular accurately define the type of a vertex. See Theorem 5.14 for the precise statement of this theorem.

Decomposition theorems such as the above are often used with dynamic programming. In Section 5.4, these techniques will allow us to construct FPT algorithms for the L_μ model-checking problem on classes of structures of bounded DAG-width or bounded Kelly-width. As discussed in Section 4.1.3, there are very few FPT algorithms for parity games and for the modal μ-calculus. Our result is significant because it provides FPT algorithms for the modal μ-calculus on restricted classes of structures and by Theorem 4.7 on page 81, these immediately give FPT algorithms for parity games on these classes.

As discussed before in Section 4.3.1 on page 84, FPT is the best we might hope for because Nicolas Markey and Philippe Schnoebelen showed in [MS06] that the general model-checking problem is reducible to model checking on cycles, where all the width measures we consider are bounded by a small constant.

The results of this chapter have been published in [BDK14].

5.1.1 Directed Separations

The desired decomposition theorem talks about the union of two structures with a small intersection and some additional arcs all going in the same direction. To formalize this, let us introduce *directed separations*.

Definition 5.2 Let $\mathcal{M}$ be a σ-structure. A pair $(\mathcal{M}_1, \mathcal{M}_2)$ of induced substructures is a *directed σ-separation of $\mathcal{M}$ with interface $\overline{X} = (x_1, \ldots, x_k)$* directed separation interface

- $V(\mathcal{M}) = V(\mathcal{M}_1) \cup V(\mathcal{M}_2)$,
- $X = \{x_1, \ldots, x_k\} = V(\mathcal{M}_1) \cap V(\mathcal{M}_2)$,
- and there are no arcs from $\mathcal{M}_2 \setminus X$ to $\mathcal{M}_1 \setminus X$. ⊣

Abusing notation, we write $\mathcal{M} = (\mathcal{M}_1, \mathcal{M}_2)$ to denote that $(\mathcal{M}_1, \mathcal{M}_2)$ is a directed separation of $\mathcal{M}$, and notationally we consider $(\mathcal{M}_1, \mathcal{M}_2)$ to be interchangeable with $\mathcal{M}$. We call $\mathcal{M}_1$ the *left side* and $\mathcal{M}_2$ the *right side* of the separation.

See Figure 5.1 for an example of a directed separations with an interface of size 3. Note that there are no arcs going from $\mathcal{M}_2 \setminus X$ to $\mathcal{M}_1 \setminus X$, but all other arcs are possible.

The idea here is that a directed separation is pointing towards its right side, and that it is difficult to go from the right side to the left side. Difficult here means that in order to go from the right side to the left side, one has to pass through the interface. On the other hand, going from the left side to the right side is always possible without restrictions.

So a path in a directed separation has only very few ways of coming back to the left side if it ever happens to visit the right side. In fact, we do not care about how exactly a path

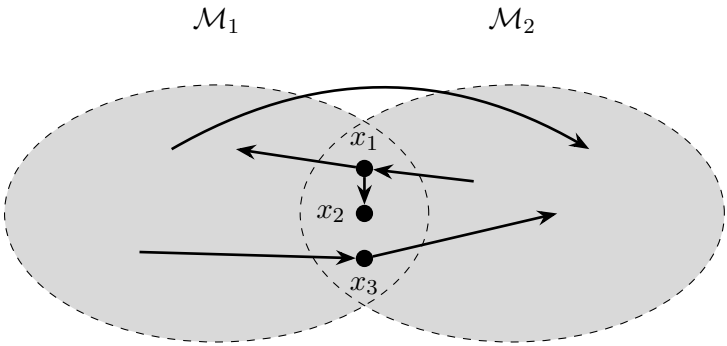


Figure 5.1: A directed separation with interface $\overline{X} = (x_1, x_2, x_3)$

manages to come back to the left side, we only care about two things: Which interface vertex does it use and what was the worst priority visited in $\mathcal{M}_2 \ltimes \varphi$? Clearly, this amount of information is independent of the size of $\mathcal{M}_2$ and only depends on the size of the interface and on the number of distinct priorities, which only depends on φ.

We want to be able to specify all this in L_μ, so we need some way to identify the interface vertices. This motivates the following definition.

$\overline{P}$
$\partial_{\overline{P}}(\mathcal{M}, \overline{X})$

Definition 5.3 For some k, let $\overline{P} = (P_1, \ldots, P_k)$ be a sequence of fresh proposition symbols. For a σ-structure $\mathcal{M}$ and a k-tuple $\overline{X} = (x_1, \ldots, x_k) \in V(\mathcal{M})^k$, we define $\partial_{\overline{P}}(\mathcal{M}, \overline{X})$ to be the $\sigma \cup P$-structure based on $\mathcal{M}$ such that P_i is true only at the vertex x_i. If the sequence $\overline{P} = (P_1, \ldots, P_k)$ is longer than $\overline{X} = (x_1, \ldots, x_l)$, then $\partial_{\overline{P}}(\mathcal{M}, \overline{X})$ is defined the same except that P_i is always false for $i > l$. ⊣

So $\partial_{\overline{P}}(\mathcal{M}, \overline{X})$ differs from $\mathcal{M}$ only in that each interface vertex receives a unique color, in order to be able to be identified by L_μ-formulas. Now let us define what it formally means that the μ-calculus is unable to distinguish the right sides of two separations.

Recall that for a set $L \subseteq L_\mu$, the L-type of a vertex v in a structure $\mathcal{M}$ is the set $\{\varphi \in L \mid \mathcal{M}, v \models \varphi\}$ (Definition 3.40 on page 44).

Definition 5.4 Let $(\mathcal{M}_1, \mathcal{M}_2)$, $(\mathcal{M}_1, \mathcal{M}_2')$ be two directed separations with the same interface $\overline{X}$ and the same left side $\mathcal{M}_1$. Let $\overline{P}$ be a set of $|X|$ many fresh proposition symbols and let $L \subseteq L_\mu[\sigma \cup P]$.

We call $(\mathcal{M}_1, \mathcal{M}_2)$, $(\mathcal{M}_1, \mathcal{M}_2')$ *L-equivalent* if L-equivalent

- for every vertex in X, its L-type is the same in $\partial_{\overline{P}}(\mathcal{M}_2, \overline{X})$ and in $\partial_{\overline{P}}(\mathcal{M}_2', \overline{X})$, and
- for every arc (v, w) in $(\mathcal{M}_1, \mathcal{M}_2)$ with $v \in \mathcal{M}_1$, $w \in \mathcal{M}_2$ there is an arc (v, w') in $(\mathcal{M}_1, \mathcal{M}_2')$ with $w' \in \mathcal{M}_2'$ such that w and w' have the same L-types in $\partial_{\overline{P}}(\mathcal{M}_2, \overline{X})$ and in $\partial_{\overline{P}}(\mathcal{M}_2', \overline{X})$, and
- for every arc (v, w') in $(\mathcal{M}_1, \mathcal{M}_2')$ with $v \in \mathcal{M}_1$, $w \in \mathcal{M}_2'$ there is an arc (v, w) in $(\mathcal{M}_1, \mathcal{M}_2)$ with $w \in \mathcal{M}_2$ such that w' and w have the same L-types in $\partial_{\overline{P}}(\mathcal{M}_2', \overline{X})$ and in $\partial_{\overline{P}}(\mathcal{M}_2, \overline{X})$. ⊣

If δ is a μ-depth (Definition 3.40), we say that two directed separations are *δ-equivalent* if they are L-equivalent with L being all formulas consistent with δ. Let us state our main theorem. δ-equivalent

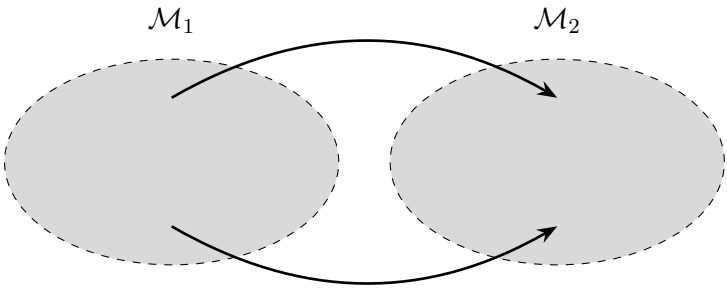


Figure 5.2: A disjoint separation

Theorem 5.5 *Let δ be a μ-depth and let $\mathcal{M} = (\mathcal{M}_1, \mathcal{M}_2)$, $\mathcal{M}' = (\mathcal{M}_1, \mathcal{M}'_2)$ be δ-equivalent directed separations. Then for every vertex in $\mathcal{M}_1$, its δ-type is the same in $\mathcal{M}$ and $\mathcal{M}'$.*

In fact, we will prove a more general version of Theorem 5.5, without limiting us to μ-depth. It turns out that there exists a suitable closure operator $\mathrm{CL}_P : 2^{L_\mu} \to 2^{L_\mu}$ that maps finite sets to finite sets such that the main theorem holds for $\mathrm{CL}_P(L)$-equivalent directed separations. In particular, we can choose $L = \{\varphi\}$ if we are only interested in the model-checking problem for a fixed formula φ consistent with δ. Then $\mathrm{CL}_P(\{\varphi\})$ will consist of δ-consistent formulas, but will also be significantly smaller than the set of all δ-consistent formulas.

5.2 Proof of the Main Theorem

5.2.1 An Easy Case

Let us first consider a much easier case, the case of a directed separation $\mathcal{M} = (\mathcal{M}_1, \mathcal{M}_2)$ with $\mathcal{M}_1 \cap \mathcal{M}_2 = \emptyset$ (see Figure 5.2).

We call such separations *disjoint*.

disjoint separation

Because the interface of disjoint separations is empty, there are no arcs from $\mathcal{M}_2$ to $\mathcal{M}_1$. Note that there are no restrictions on the arcs that are fully inside one of the substructures.

With a formula φ we can construct the model-checking game $\mathcal{M} \ltimes \varphi$ and observe that its set of vertices is the disjoint union of $\mathcal{M}_1 \ltimes \varphi$ and $\mathcal{M}_2 \ltimes \varphi$, with no arcs from the latter to the former. So this parity game has at least two strongly connected components.

How to solve parity games comprising multiple strongly connected components is well-known in the literature (see e.g., [FL09]); the solution is to solve the leaves first and then work backward along the topologically sorted components.

In the case of only two components A, B with arcs going from A to B, this means we solve B first and annotate every vertex in B with its winning player. Then all of B can be replaced by just two representative vertices, one winning vertex per player. All arcs pointing from A to B are redirected to the appropriate vertex. Finally we can solve A together with these two vertices, thus saving a considerate amount of work compared to solving $A \cup B$ directly.

The same argument applies to the model-checking game $\mathcal{M} \ltimes \varphi$, because, as we saw, it too consists of multiple components. In particular, the player winning from a vertex in $\mathcal{M}_1 \ltimes \varphi$ does not actually depend on the whole structure of $\mathcal{M}_2$. The only information we need to compute the winner is the winning player for each vertex $(v, \psi) \in \mathcal{M}_2 \ltimes \varphi$.

However, the winning player of this vertex is already determined by the truth value of a single formula on v in $\mathcal{M}_2$. The correctness of the model-checking game (Theorem 3.72 on page 68) says exactly what we need here; it gives us that

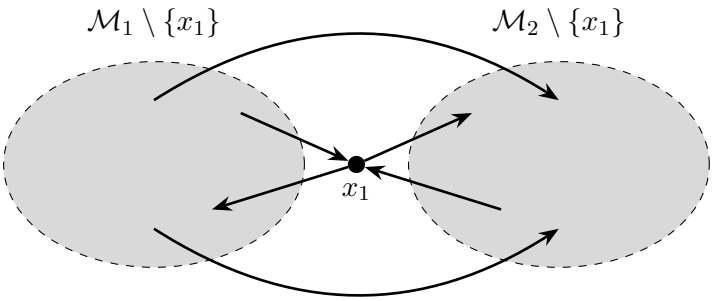


Figure 5.3: A separation with one vertex in the interface

$\mathcal{M}_2, v \models \psi$ if and only if player $\Diamond$ wins from (v, ψ).

So if we know for each vertex $v \in \mathcal{M}_2$ the truth value of every formula in $\mathrm{CL}(\varphi)$, then we already know the winning player of every vertex in $\mathcal{M}_2 \ltimes \varphi$ in the model-checking game.

So it seems that $\{\psi \in \mathrm{CL}(\varphi) \mid \mathcal{M}, v \models \psi\}$ could be a suitable definition for the *type* of a vertex. For disjoint separations, this is true. For general directed separations, however, we are going to need something more sophisticated.

5.2.2 A Slightly More General Case

Let us consider a slightly more general case. Fix an arbitrary formula φ and let $\mathcal{M} = (\mathcal{M}_1, \mathcal{M}_2)$ be a directed separation such that $\mathcal{M}_1 \cap \mathcal{M}_2 = \{x_1\}$ (see Figure 5.3). Now the model-checking game $\mathcal{M} \ltimes \varphi$ cannot be as neatly divided into components as we did when we had a disjoint separation. However, not all is lost. Every arc from $\mathcal{M}_2 \ltimes \varphi$ to $\mathcal{M}_1 \setminus \{x_1\} \ltimes \varphi$ has its tail in $\{x_1\} \ltimes \varphi$ (see Figure 5.4).

Again, we would like to solve $\mathcal{M}_2 \ltimes \varphi$ separately from the

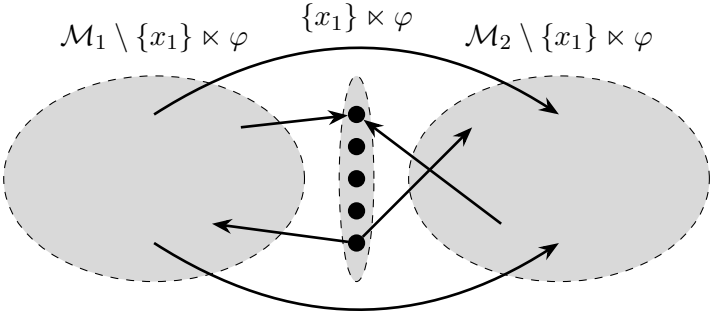


Figure 5.4: The model-checking game of Figure 5.3

rest and replace it by something simpler. In the case of disjoint separations, we were able to simplify this subgame down to two vertices, because who is winning is the only relevant information if we can never leave the subgame. Now we can leave the subgame, but only by going through $\{x_1\} \ltimes \varphi$. The idea is to mark x_1 with a fresh predicate so that we have a way of identifying x_1 in formula, hopefully becoming able to solve $\mathcal{M}_2 \ltimes \varphi$ separately from the rest.

Let P be a fresh predicate and assume that P in $\mathcal{M}$ is true only at x_1; note the we are slowly moving towards the definition of $\partial_{(P)}(\mathcal{M}_2, (x_1))$ we presented on page 102 (where (P) and (x_1) are 1-element tuples). We now want to find the minimum information that we need from each vertex of $\mathcal{M}_2 \ltimes \varphi$ to determine the winning regions of $\mathcal{M}_1 \ltimes \varphi$ in $\mathcal{M} \ltimes \varphi$. In the case of disjoint separations, this was simply the winning player.

Let us fix a vertex $(v, \psi) \in \mathcal{M}_2 \ltimes \varphi$. Suppose that this vertex is losing for player $\Diamond$ in the subgame restricted to $\mathcal{M}_2 \ltimes \varphi$. This certainly looks like useful information. However, it could

be that (v, ψ) is winning in the full game $\mathcal{M} \ltimes \varphi$. Fortunately, this is possible only if player $\Diamond$ can force his way from (v, ψ) back to $\mathcal{M}_1 \setminus \{x_1\} \ltimes \varphi$, because otherwise the play would stay in $\mathcal{M}_2 \ltimes \varphi$ where we assumed player $\Diamond$ is losing. This can happen only through $\{x_1\} \ltimes \varphi$ because $(\mathcal{M}_1, \mathcal{M}_2)$ is a directed separation with interface $\{x_1\}$.

In fact, we can be even more precise: By the construction of the model-checking game, the only vertices in $\{x_1\} \ltimes \varphi$ that could possibly allow player $\Diamond$ to move back to

$$\mathcal{M}_1 \setminus \{x_1\} \ltimes \varphi$$

are (x_1, χ) for some $\chi \in \mathrm{CL}(\varphi)$ where χ starts with a $\Box$ or $\Diamond$. All vertices (x_1, χ) where χ does not start with $\Box$ or $\Diamond$ have outgoing arcs only to vertices of the form (x_1, χ') for some χ'; the first component of the vertices in the model-checking game only changes when going through a $\Box$ or $\Diamond$-formula.

On further observation we notice that also $\Box$-vertices cannot be responsible for allowing player $\Diamond$ to leave the subgame because we assumed that (v, ψ) was losing for player $\Diamond$ in $\mathcal{M}_2 \ltimes \varphi$. So every $\Box$-vertex must offer a choice for player $\Box$ to stay in $\mathcal{M}_2 \ltimes \varphi$.

We conclude that

$$X_1 := \{x_1\} \times \Psi$$

with

$$\Psi := \{\psi \in \mathrm{CL}(\varphi) \mid \psi \text{ starts with a } \Diamond\}$$

is the set of interest. Note that the size of this set is independent of $\mathcal{M}_1$ and $\mathcal{M}_2$.

So it seems that we need to know which of the vertices in X_1

are reachable from (v, ψ). Moreover, after having left $\mathcal{M}_2 \ltimes \varphi$, the game is far from over. The players might end up going back to $\mathcal{M}_2 \ltimes \varphi$. So we need to collect the reachability information not only for (v, ψ) but for all vertices in $\mathcal{M}_2 \ltimes \varphi$.

On top of this, we have a parity game, so reachability is a little more involved than just graph reachability. We also need to keep track of the worst priority that we visited on the way. Here we use an idea similar to the idea of a "border" from Jan Obdržálek [Obd03].

What we do is that for each subset $\Psi' \subseteq \Psi$, we assume that all vertices in $\{x_1\} \times \Psi'$ are immediately winning for player $\diamond$. Under this assumption, we check if (v, ψ) is winning for player $\diamond$ in $\mathcal{M}_2 \ltimes \varphi$. We get a function mapping subsets $\Psi' \subseteq \Psi$ to players $\{\diamond, \Box\}$.

Let us compare this one more time with the easy case of disjoint separations. There the information that we collected for each $(v, \psi) \in \mathcal{M}_2 \ltimes \varphi$ was simply the winning player. This time, we collect a function mapping subsets $\Psi' \subseteq \Psi$ to the winning player.

We claim two things.

1. We can define this information by evaluating a set of formulas on $\mathcal{M}_2, v$.

2. Collecting this information for all $(v, \psi) \in \mathcal{M}_2 \ltimes \varphi$ is enough to solve the remainder of the game.

Let us look at the first claim. We immediately see that $\mathrm{CL}(\varphi)$ will not be sufficient to define this information, except if $\Psi' = \emptyset$. We need to take the subset $\Psi' \subseteq \Psi$ into account.

First, let us assume for simplicity that $\Psi' = \Psi$. This means that we now work under the assumption that all vertices in

X_1 are immediately winning for player $\Diamond$. This is the point where the proposition P comes into play. The proposition P is true only at x_1, so it provides a way to say in a formula "we are currently at x_1". So we can easily express "a vertex $(x_1, \chi) \in X_1$ is winning for player $\Diamond$" by replacing χ with $P \vee \chi$ everywhere.

We do this for every $\psi \in \mathrm{CL}(\varphi)$. More precisely, we construct a formula ψ' by replacing in ψ every subformula of the form $\Diamond\chi$ with $P \vee \Diamond\chi$. Then $\mathcal{M}_2, v \models \psi'$ iff (v, ψ) is winning for player $\Diamond$.

The idea is to generalize this to arbitrary subsets Ψ'. But this is easy: We simply replace only the subformulas $\Diamond\chi \in \Psi'$ with $P \vee \Diamond\chi$. We conclude that there is a way of defining this information with formulas, although we still need to make everything precise.

The other claim is that this information suffices to solve $\mathcal{M}_1 \ltimes \varphi$. Here the idea is that we replace $\mathcal{M}_2 \ltimes \varphi$ by a simpler parity game, similar to replacing it with two vertices like we did when we had a disjoint separation.

Now, the replacement game will be a lot more involved and will not be a dead-end but instead have arcs pointing back to $\mathcal{M}_1 \ltimes \varphi$. We need to represent all the ways how subsets of X_1 are reachable. We also need to care about the priorities with which each reachable vertex in X_1 is reachable.

The crucial observation here is that we can fix the priority of every μ/ν-subformula in φ once and then use the same priorities for every $\psi \in \mathrm{CL}(\varphi)$[1]. From this it follows that although there may be more than one path reaching a vertex

[1] This is expressed formally by Theorem 3.72 on page 68, in contrast to the weaker Theorem 3.70.

(x_1, ψ) in $\mathcal{M}_2 \ltimes \varphi$, they will all visit the same minimum priority. So it is indeed enough to consider subsets $\Psi' \subseteq \Psi$ and ignore the priorities, because they are determined.

This means that we can replace $\mathcal{M}_2 \ltimes \varphi$ by a parity game whose size only depends on φ and whose structure is completely determined by the types of the vertices in $\mathcal{M}_2 \ltimes \varphi$, because the types describe exactly how the players can win, lose, or leave this subgame.

Unfortunately, making all these notions precise is a fair bit of work. We devote the remainder of Section 5.2 to this.

5.2.3 Priority Tracking

As a motivating example, consider the formula $\varphi := \Diamond\psi \wedge \mu X.\Diamond X \vee \Diamond\psi$ for some $\psi \in L_\mu$. Recalling the definition of $\mathrm{CL}(\varphi)$ from Section 3.1.7 on page 34, we observe that the subformula $\Diamond\psi$ occurs twice in φ, so $\mathrm{CL}(\varphi)$ contains the two elements $(\Diamond\psi, 2)$ and $(\Diamond\psi, i)$ for some $i > 2$ in order to distinguish both occurrences.

As described above, we would like to replace $\Diamond\psi$ by $P_1 \vee \Diamond\psi$, where P_1 is a proposition that is true at only one vertex. In order to handle more than one vertex in the interface of a directed separation, we also need to be able to replace $\Diamond\psi$ with $P_2 \vee \Diamond\psi$, or with $P_1 \vee P_2 \vee \Diamond\psi$, or any other combination of proposition symbols that identify the vertices in the interface. We also want to be able to replace the two occurrences of $\Diamond\psi$ with different combinations of disjunctions. This motivates the following definition.

Let $P = \{P_1, \ldots, P_k\}$ be a set of proposition symbols disjoint from σ.

Definition 5.6 For a formula $\varphi \in L_\mu[\sigma \cup P]$ we define the set of *priority tracking variants of* φ, denoted as $\mathrm{PT}_P(\varphi)$, inductively as follows.

$\mathrm{PT}_P(\varphi)$

$$
\begin{aligned}
\mathrm{PT}_P(\varphi) &= \{\varphi\} \quad \text{for atomic } \varphi \\
\mathrm{PT}_P(\neg\varphi) &= \{\neg\varphi' \mid \varphi' \in \mathrm{PT}_P(\varphi)\} \\
\mathrm{PT}_P(\varphi \vee \psi) &= \\
&\qquad \{\varphi' \vee \psi' \mid \varphi' \in \mathrm{PT}_P(\varphi), \psi' \in \mathrm{PT}_P(\psi)\} \\
\mathrm{PT}_P(\varphi \wedge \psi) &= \\
&\qquad \{\varphi' \wedge \psi' \mid \varphi' \in \mathrm{PT}_P(\varphi), \psi' \in \mathrm{PT}_P(\psi)\} \\
\mathrm{PT}_P(\mu X.\varphi) &= \{\mu X.\varphi' \mid \varphi' \in \mathrm{PT}_P(\varphi)\} \\
\mathrm{PT}_P(\nu X.\varphi) &= \{\nu X.\varphi' \mid \varphi' \in \mathrm{PT}_P(\varphi)\}
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{PT}_P\Big(\Big(\bigvee_{R\in S} R\Big) \vee \Diamond\varphi\Big) = \\
&\qquad \Big\{\Big(\bigvee_{R\in Q\cup S} R\Big) \vee \Diamond\varphi' \;\Big|\; Q \subseteq P, \varphi' \in \mathrm{PT}_P(\varphi)\Big\} \\
&\mathrm{PT}_P\Big(\Big(\bigwedge_{R\in S} R\Big) \vee \Box\varphi\Big) = \\
&\qquad \Big\{\Big(\bigwedge_{R\in Q\cup S} \neg R\Big) \vee \Box\varphi' \;\Big|\; Q \subseteq P, \varphi' \in \mathrm{PT}_P(\varphi)\Big\}. \quad \dashv
\end{aligned}
$$

Note that $\mathrm{PT}_P(\varphi)$ is finite because P is finite. The complicated way of writing the last two lines is necessary in order to get idempotency. Note that the set S in the last two lines of the definition may be empty.

Observation 5.7 $\mathrm{PT}_P(\mathrm{PT}_P(\varphi)) = \mathrm{PT}_P(\varphi)$ *for all* $\varphi \in L_\mu[\sigma \cup P]$.

Observation 5.8 *If $\varphi \in L_\mu[\sigma \cup P]$ has $\overline{X}$-depth at most d, then $\mathrm{PT}_P(\varphi)$ has $\overline{X}$-depth at most d.*

Definition 5.9 Similar to the closure operator CL, we define $\mathrm{PT}_P(L)$ for sets of formulas $L \subseteq L_\mu[\sigma \cup P]$ as

$$\mathrm{PT}_P(L) := \bigcup_{\varphi \in L} \mathrm{PT}_P(\varphi). \qquad \mathrm{PT}_P(L)$$

We introduce the abbreviation $\mathrm{CL}_P(L) := \mathrm{PT}_P(\mathrm{CL}(L))$. ⊣ $\mathrm{CL}_P(L)$

With Corollary 3.39 on page 44 and Observation 5.8, we immediately get that $\overline{X}$-depth is preserved under CL_P.

Lemma 5.10 *If $\varphi \in L_\mu[\sigma \cup P]$ has $\overline{X}$-depth at most d, then every $\psi \in \mathrm{CL}_P(\{\varphi\})$ has $\overline{X}$-depth at most d.*

We also find that CL_P is idempotent.

Lemma 5.11 $\mathrm{CL}_P(\mathrm{CL}_P(L)) = \mathrm{CL}_P(L)$ *for all* $L \subseteq L_\mu[\sigma \cup P]$.

Proof. By definition, $\mathrm{CL}_P(L)$ is closed under PT_P. Hence, it is enough to show $\mathrm{CL}(\mathrm{CL}_P(L)) = \mathrm{CL}_P(L)$. Let $\mu X.\varphi \in \mathrm{CL}_P(L)$. We want to show that $\varphi[X/\mu X.\varphi] \in \mathrm{CL}_P(L)$, where $\varphi[X/\mu X.\varphi]$ is φ with all free occurrences of X replaced by $\mu X.\varphi$. By definition of $\mathrm{PT}_P(\mathrm{CL}(L))$, there is a $\mu X.\varphi' \in \mathrm{CL}(L)$ with $\varphi \in \mathrm{PT}_P(\varphi')$. Because $\mathrm{CL}(L)$ is essentially equal to the Fischer-Ladner closure of L, we have $\varphi'[X/\mu X.\varphi'] \in \mathrm{CL}(L)$. Since $\varphi \in \mathrm{PT}_P(\varphi')$, we have $\varphi[X/\mu X.\varphi] \in \mathrm{PT}_P(\varphi'[X/\mu X.\varphi'])$, hence $\varphi[X/\mu X.\varphi] \in \mathrm{PT}_P(\mathrm{CL}(L))$.

The other cases are similar. ∎

Definition 5.12 For a structure $\mathcal{M}, v$, a k-tuple $\overline{X} \in V(\mathcal{M})^k$ and a set $L \subseteq L_\mu[\sigma \cup P]$ where $\overline{P}$ is a sequence of at least k many proposition symbols, we define the $(L, \overline{P})$-*type of* v *in* $\mathcal{M}, \overline{X}$ as

$(L,\overline{P})$-type of v in $\mathcal{M},\overline{X}$

$$\mathrm{tp}_{L,\overline{P}}(\mathcal{M}, v, \overline{X}) := \{\varphi \in \mathrm{CL}_P(L) \mid \partial_{\overline{P}}(\mathcal{M}, \overline{X}), v \models \varphi\}. \qquad \mathrm{tp}_{L,\overline{P}}$$

We also define the set of $(L, \overline{P})$-types realized in a structure,

$$\mathcal{T}_{L,\overline{P}}(\mathcal{M}, \overline{X}) := \{\mathrm{tp}_{L,\overline{P}}(\mathcal{M}, v, \overline{X}) \mid v \in V(\mathcal{M})\}. \qquad \mathcal{T}_{L,\overline{P}}(\mathcal{M},\overline{X})$$

Finally, let

$$\mathcal{T}_L(\overline{P}) := 2^{\mathrm{CL}_P(L)} \qquad \mathcal{T}_L(\overline{P})$$

be the set of all candidates for $(L, \overline{P})$-types. ⊣

5.2.4 The General Case

First, let us restate Theorem 5.5 in a more general form.

Theorem 5.13 *Let* $\overline{P}$ *be a sequence of proposition symbols disjoint from* σ*,* $L \subseteq L_\mu[\sigma \cup P]$ *and let* $(\mathcal{M}_1, \mathcal{M}_2)$*,* $(\mathcal{M}_1, \mathcal{M}_2')$ *be* $\mathrm{CL}_P(L)$*-equivalent directed* σ*-separations with interface* $\overline{X}$*.*

Then for all $v \in \mathcal{M}_1$*, we have*

$$\mathrm{tp}_{L,\overline{P}}((\mathcal{M}_1, \mathcal{M}_2), v, \overline{X}) = \mathrm{tp}_{L,\overline{P}}((\mathcal{M}_1, \mathcal{M}_2'), v, \overline{X}).$$

We have already seen that if all formulas in $L \subseteq L_\mu$ are consistent with a μ-depth δ, then the same is true for $\mathrm{CL}_P(L)$; this is Lemma 5.10. Therefore, δ-equivalence implies $\mathrm{CL}_P(L)$-equivalence, and thus Theorem 5.5 follows from Theorem 5.13. We will also use a different and slightly stronger way of stating Theorem 5.13, stated below.

Theorem 5.14 *Let $\overline{P}$, $\overline{Q}$ be sequences of proposition symbols such that $\sigma \cap P = \sigma \cap Q = P \cap Q = \emptyset$.*

Let $L \subseteq L_\mu[\sigma \cup P]$ and $\mathcal{M}$ be a structure with a directed σ-separation $(\mathcal{M}_1, \mathcal{M}_2)$ with interface $\overline{X}$. Let $\overline{Y} \in V(\mathcal{M}_1)^{|Q|}$ be a tuple.

For all $v \in \mathcal{M}_1$, the set $\mathrm{tp}_{L,\overline{Q}}(\mathcal{M}, v, \overline{Y})$ depends only on

- *$\mathcal{M}_1$ and $\overline{Q}$ and*
- *$\left\{(x_i, \mathrm{tp}_{L,\overline{P}}(\mathcal{M}_2, x_i, \overline{X})) \;\middle|\; x_i \in X\right\}$ and*
- *$\left\{(v, \mathrm{tp}_{L,\overline{P}}(\mathcal{M}_2, w, \overline{X})) \mid (v, w) \in E(\mathcal{M}) \cap (\mathcal{M}_1 \times \mathcal{M}_2)\right\}$.*

Provided L is finite, $\mathrm{tp}_{L,\overline{Q}}(\mathcal{M}, v, \overline{Y})$ can be computed from these sets.

Furthermore, for every $w \in \mathcal{M}_2$, the set $\mathrm{tp}_{L,\overline{Q}}(\mathcal{M}, w, \overline{Y})$ depends only on the above sets and on $\mathrm{tp}_{L,\overline{P}}(\mathcal{M}_2, w, \overline{X})$ and can be computed from these sets if L is finite.

5.2.5 Parity Games

To prove the decomposition theorems, we want to use the model-checking game of the modal μ-calculus. Instead of replacing a substructure by a different substructure preserving the types in the whole structure, we replace a subgame by a different subgame preserving the winner in the whole game.

The winner of a parity game from a given vertex is always determined (see Chapter 7). However, in order to replace subgames by different subgames preserving the winner in the whole game, we need a more subtle analysis of the subgame than just its winner.

We call the intersection between a subgame and the rest of the game its *interface*. For the more subtle analysis, we look at *partial* strategies, which may be undefined on some vertices of the interface. If a partial strategy is undefined on some vertex, the player indicates that she would like to leave the subgame. We then partially order these strategies by their *profiles*, that is, the set of interface vertices that are possibly reachable by player $\Box$, together with the worst priority that player $\Box$ can enforce.

All this culminates in a proof that the feasibility of profiles of strategies is in fact definable in L_μ. The formulas that define profiles in a partial model-checking game of φ will all be in $\mathrm{CL}_P(\{\varphi\})$, so this proves that $\mathrm{tp}_{\{\varphi\},\overline{P}}(\mathcal{M}, v, \overline{X})$ determines the set of possible profiles, which we will use to define a specific parity game.

From now on, let us fix a sequence $\overline{Z}$ of fixpoint variables and a priority sequence $(p_1, \ldots, p_n) \in \mathbb{N}^n$ (see Definition 3.67). All formulas in the rest of Section 5.2 should be consistent with $\overline{Z}$ and annotated with the p_i, even if we do not mention this explicitly.

5.2.6 Profiles and Types

We already know parity games, (positional) strategies and the model-checking game from page 47ff. We now generalize these definitions to partial games and partial strategies. This is necessary so we can analyze the effect of replacing a subgame by a different, but in some sense, similar subgame.

partial game
interface of a partial game

Definition 5.15 A *partial parity game* is a parity game P with a subset $U \subseteq V(P)$ called the *interface*. ⊣

The game is played the same way as a parity game, except that upon reaching an interface $\Diamond$-vertex, player $\Diamond$ may choose to end the play and win immediately. Therefore, a *partial strategy* for player $\Diamond$ is defined the same way as in a non-partial parity game, except that the partial strategy may be undefined on plays that end in an interface $\Diamond$-vertex.

Definition 5.16 Let P be a partial parity game. A *partial strategy* for player $\Diamond$ for a game (P, v_1) is a partial function $\pi : V(P)^+ \to V(P)$ with the following conditions. partial strategy

1. For every $(v_1, \ldots, v_n) \in \mathrm{dom}(\pi)$, the sequence $(v_1, \ldots, v_n, \pi(v_1, \ldots, v_n))$ is a π-conforming path in P with $v_n \in V_\Diamond(P)$.

2. For every π-conforming path $(v_1, \ldots, v_n)$, if $v_n \in V_\Diamond(P)$ and $v_n \notin U$, then $(v_1, \ldots, v_n) \in \mathrm{dom}(\pi)$. $\dashv$

A partial strategy π is called *winning* if for every strategy of the opponent, the resulting play either visits an interface vertex where π is undefined or satisfies the parity condition. Formally, we define this as follows.

Definition 5.17 Let (P, v_1) be a partial parity game with interface U and π be a partial strategy. Let P' be the game constructed from P by adding a $\Box$-vertex called $\top$ and an arc from every vertex in $V_\Diamond \cap U$ to $\top$. Then define π' as an extension of π such that on all π-conforming paths $(v_1, \ldots, v_n)$ with $v_n \in V_\Diamond \cap U$, if $(v_1, \ldots, v_n) \notin \mathrm{dom}(\pi)$, then $\pi'((v_1, \ldots, v_n)) = \top$. Then π' is a strategy on (P', v_1).

We say that π is a *partial winning strategy* from vertex v_1 iff π' wins from vertex v_1 in the game P'. $\dashv$ partial winning strategy

If we have a structure together with some subset of its vertices, we consider the corresponding model-checking games to be partial with respect to these vertices.

$\mathcal{M} \ltimes_X \varphi$

Definition 5.18 Let $\varphi \in L_\mu[\sigma]$, $\mathcal{M}$ be a σ-structure and $X \subseteq V(\mathcal{M})$. The game $\mathcal{M} \ltimes_X \varphi$ is the partial parity game defined as $\mathcal{M} \ltimes \varphi$ with interface

$$\{(v, \psi) \in X \times \mathrm{CL}(\varphi) \mid \psi \text{ starts with } \Diamond \text{ or } \Box\}.$$

We will usually write $\mathcal{M} \ltimes \varphi$ for this game if X is clear from the context. ⊣

Definition 5.19 Let P be a partial parity game with interface U. We define

strategy-targets(P)

$$\begin{aligned}&\text{strategy-targets}(P) :=\\ &\quad \{(u, p) \mid u \in U,\ p \text{ a priority of } P\}\end{aligned}$$

profiles(P)

$$\begin{aligned}&\text{profiles}(P) :=\\ &\quad \{y \subseteq \text{strategy-targets}(P) \mid \text{for all } u\\ &\qquad \text{there is at most one } p \text{ with } (u, p) \in y\}.\end{aligned}$$

⊣

$p \sqsubseteq p'$

Definition 5.20 Let $\sqsubseteq$ be the *reward ordering* on priorities. That is, $p \sqsubseteq p'$ if p is better for player $\Diamond$ than p'. Formally, $p \sqsubseteq p'$ is true if and only if

- p is even and p' is odd or
- both p and p' are even and $p \leq p'$ or
- both p and p' are odd and $p \geq p'$. ⊣

Definition 5.21 Let P be a partial parity game with interface U, $v_1 \in V(P)$ and let π be a partial winning strategy for (P, v_1). We define

$$\begin{aligned}
&\mathrm{preprofile}(\pi, v_1) := \\
&\quad \{(v_n, \min_{1 \le i \le n} \omega(v_i)) \mid \\
&\qquad n > 1,\ (v_1, \ldots, v_n) \text{ is a } \pi\text{-conforming path with} \\
&\qquad v_n \in U \text{ and } (v_1, \ldots, v_n) \notin \mathrm{dom}(\pi)\} \\
&\mathrm{profile}(\pi, v_1) := \\
&\quad \{(u, p) \mid p \text{ is } \sqsubseteq\text{-maximal such that} \\
&\qquad (u, p) \in \mathrm{preprofile}(\pi, v_1)\}.
\end{aligned}$$

preprofile(π, v_1)

profile(π, v_1)

The min is taken with respect to the usual ordering $\leq$.

We say that a profile $y \in \mathrm{profiles}(P)$ is *possible* on (P, v_1) if there exists a π such that $y = \mathrm{profile}(\pi, v_1)$. ⊣

possible profile

Definition 5.22 Let $y, y' \in \mathrm{profiles}(P)$. We say that y *is at least as good as* y' iff for every $(u, p) \in y$, there is a $(u, p') \in y'$ with $p \sqsubseteq p'$. We denote this as $y \sqsubseteq y'$. ⊣

$y \sqsubseteq y'$

As an example, consider the two parity games given in Figure 5.5 with interface vertices ○, ◎. For simplicity, we assume that all vertices in these parity games have priority 0. Then the profile $\{(○, 0)\}$ is possible on (P_1, v) but not on (P_2, v). On the other hand, the profile $\{(○, 0), (◎, 0)\}$ is possible on both (P_1, v) and (P_2, v). Note that on (P_1, v), the last profile is only possible with a non-positional strategy. However, the need for a non-positional strategy here is of course somewhat artificial because player $\diamond$ must deliberately avoid a decision where she could simply make one.

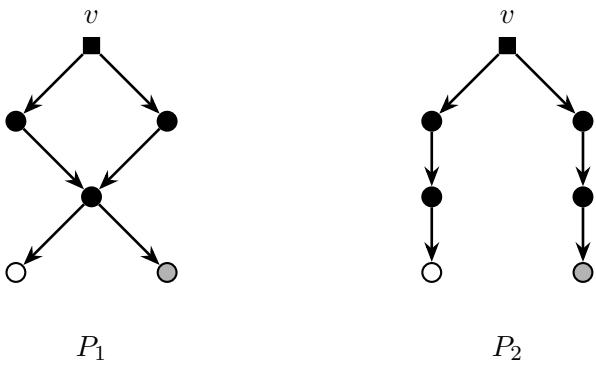


Figure 5.5: Two parity games with similar possible profiles

As one might expect, every partial strategy can be converted into a positional partial strategy at least as good as the original strategy.

Lemma 5.23 *Let $P = (V, V_\Diamond, E, \omega)$ be a partial parity game with interface U, $v \in V$ and π be a partial strategy for (P, v). Then there exists a positional partial strategy ρ such that*

$$\text{profile}(\rho, v) \sqsubseteq \text{profile}(\pi, v).$$

Proof. The proof is a reduction to the positional determinacy of (non-partial) parity games.

We define a game $P' = (V', V'_\Diamond, E', \omega')$ based on P and use Theorem 3.54. Let

$$\begin{aligned} V' &:= V \cup \{v_p \mid p \text{ is a priority of } P\} \cup \{v_\bot\} \\ V'_\Diamond &:= V_\Diamond \cup \{v_\bot\} \\ E' &:= E \cup \{(v_p, v) \mid p \text{ is a priority of } P\} \end{aligned}$$

$$
\begin{aligned}
&\cup \{(u, v_p) \mid p \text{ is odd} \\
&\qquad\qquad \text{and } (u, p-1) \in \text{profile}(\pi, v)\} \\
&\cup \{(u, v_p) \mid p \text{ is even} \\
&\qquad\qquad \text{and } (u, p+1) \in \text{profile}(\pi, v)\} \\
&\cup \{(u, v_\bot) \mid (u, p) \notin \text{profile}(\pi, v) \text{ for all } p\}
\end{aligned}
$$

$$
\omega'(w) := \begin{cases} \omega(w) & \text{if } w \in V(P) \\ 0 & \text{if } w = v_\bot \\ p & \text{if } w = v_p. \end{cases}
$$

Note that for each $u \in U$, there is exactly one v_p such that $(u, v_p) \in E'$. So we can extend π to a strategy π' on P' by defining $\pi'(v_1, \ldots, u) = v_p$ if $\pi(v_1, \ldots, u)$ is undefined.

We claim that π' is a winning strategy. Let $v = v_1, v_2, \ldots$ be an infinite π'-conforming path. Clearly the path is winning if it has a π-conforming suffix.

So assume that it visits some $u \in U$ an infinite number of times followed by v_p. If p is odd, then $(u, p-1) \in \text{profile}(\pi, v)$ guarantees that the worst priority on all path segments that go from v to u is $p-1$. By the pigeon principle there is at least one priority $p' \sqsubseteq p-1$ that we visit infinitely often on the path. Furthermore, $p' \leq p-1$ because $p-1$ is even. This means the priority p of v_p is irrelevant because $p' < p$.

If p is even, then $(u, p+1) \in \text{profile}(\pi, v)$ guarantees that the worst priority on all path segments that go from v to u is $p+1$. So there must be a minimum priority $p' \sqsubseteq p+1$ that occurs infinitely often on these path segments. If $p' \geq p$, then p' becomes irrelevant because we visit v_p an infinite number of times. If $p' < p$, then p becomes irrelevant. However, $p' \sqsubseteq p+1$ then implies that p' is even.

We repeat this argument for all pairs (u, v_p) that occur infinitely often in the path. We see that in all cases the minimum priority that occurs infinitely often is even, so π' is a winning strategy.

By Theorem 3.54 on page 52, there exists a positional winning strategy ρ' on (P', v). Let ρ be the restriction of ρ' to P. We claim that $\text{profile}(\rho, v) \sqsubseteq \text{profile}(\pi, v)$.

Clearly $(u, p) \notin \text{profile}(\pi, v)$ implies $(u, p) \notin \text{profile}(\rho, v)$ because otherwise we would visit the vertex $v_\bot$ and immediately lose. Let $(u, p) \in \text{profile}(\rho, v)$ and $(u, p') \in \text{profile}(\pi, v)$. We have to show $p \sqsubseteq p'$. If $p \sqsupset p'$, then there is a ρ-conforming path from v to u with a priority no better than p. In P' this gives us a ρ'-conforming path by going back from u to v. However, the only new vertex we visit is $v_{p''}$ and p'' is not enough to offset p, so this path loses, contradicting the fact that ρ' was a winning strategy. ∎

Definition 5.24 The type of a vertex $v \in V(P)$ is the set of optimal profiles.

$\text{ptype}_P(v)$

$$\begin{aligned}\text{ptype}_P(v) := \{&\text{profile}(\pi, v) \mid \\ &\pi \text{ is a partial winning strategy for } (P, v) \text{ and} \\ &\text{there is no partial winning strategy } \pi' \text{ such that} \\ &\quad \text{profile}(\pi', v) \sqsubset \text{profile}(\pi, v)\}.\end{aligned}$$ ⊣

By Lemma 5.23, the strategies occurring in the above definition can be chosen to be positional.

Next, we define the notion of a parity game *simulating* another parity game. A game simulates another game if it behaves in the same way when viewed from the outside. For every vertex in the old game there must be a vertex in the new

game that has the same type. Internally the games could be quite different, and in fact the new game could have a very different number of vertices than the old game.

Our goal is to find small games that simulate large games.

Definition 5.25 Let P, P' be partial parity games with the same interface U.

The game P' *simulates* P if there is a map $f : V(P) \to V(P')$ such that $f(u) = u$ for all $u \in U$ and for every vertex $v \in V(P)$, $\mathrm{ptype}_P(v) = \mathrm{ptype}_{P'}(f(v))$. ⊣

simulation

Whenever we have a game P with an induced subgame Q with no arcs going from Q to the rest of P except via the interface of Q, then we can replace Q in P by one of its simulations without the rest of P noticing.

Lemma 5.26 (Simulation Lemma) *Let P, Q be parity games such that Q is an induced subgame of P with interface U and with no arcs from $Q \setminus U$ to $P \setminus Q$. Let Q' be a partial parity game with interface U which simulates Q via the function $f : V(Q) \to V(Q')$. Extend f to $V(P)$ by letting $f(v) = v$ for all $v \in V(P) \setminus V(Q)$.*

Define P' as the parity game where the induced subgame Q has been replaced by Q' and arcs from $P \setminus Q$ pointing to vertices $v \in V(Q)$ now point to $f(v) \in V(Q')$.

Then for all $v \in V(P)$, player $\Diamond$ wins (P, v) iff player $\Diamond$ wins $(P', f(v))$.

Proof. Let P, Q, f be as described. Let π be a positional winning strategy for player $\Diamond$ on $W_\Diamond(P)$.

Let $v \in V(P)$ and $w = f(v)$. We claim that player $\Diamond$ has a winning strategy from (P', w). We will define this strategy

as we go. While defining it, we will at nearly all times keep track of a partial winning strategy ϱ on Q' and overwrite ϱ throughout the play at specific points. Let us postpone the choice of ϱ for now.

Whenever it is player $\Diamond$'s turn, we play according to π or according to ϱ, depending on whether we are in $P' \setminus Q'$ or in Q'. Formally, if we are on $w \in P' \setminus Q'$, then player $\Diamond$ chooses $f(\pi(w))$. Note that in this case we also have $w \in P \setminus Q$ by the definition of P'. If the play is on $w \in Q'$ and if $\varrho(w)$ is defined, then player $\Diamond$ chooses $\varrho(w) \in Q'$. If the play is on $w \in U$ and $\varrho(w)$ is undefined, then player $\Diamond$ chooses $f(\pi(w))$.

We claim that a play following the above strategy is winning for player $\Diamond$. We uphold the invariant that if the play visits a vertex $w \in P'$ such that there exists a $v \in P$ with $f(v) = w$, then $v \in W_\Diamond(P)$. For the first vertex, this is true.

But first, let us define the updates of ϱ. Initially, ϱ is undefined. Let $(w_1, \ldots, w_n)$ be the play so far. We update ϱ whenever both of the following two conditions are true.

1. $w_n \in Q'$.
2. $n = 1$ or $w_{n-1} \notin Q'$.

By the definition of P' and the choice of the initial vertex, if both these conditions are true, then there exists v such that $f(v) = w_n$. We then update ϱ by choosing a partial winning strategy such that

$$\operatorname{profile}(\varrho, w_n) \sqsubseteq \operatorname{profile}(\pi\restriction_Q, v)$$

and

$$\operatorname{profile}(\varrho, w_n) \in \operatorname{ptype}_{P'}(w_n).$$

Such a strategy exists because $\pi\restriction_Q$ is a partial winning strategy for (Q, v), which implies that there exists some ϱ' with

$$\text{profile}(\varrho', v) \sqsubseteq \text{profile}(\pi\restriction_Q, v)$$

such that

$$\text{profile}(\varrho', v) \in \text{ptype}_P(v).$$

Because we have

$$\text{ptype}_P(v) = \text{ptype}_{P'}(w_n),$$

there exists a partial winning strategy ϱ for (Q', w_n) with

$$\text{profile}(\varrho, w_n) \in \text{ptype}_{P'}(w_n)$$

such that

$$\text{profile}(\varrho, w_n) = \text{profile}(\varrho', v).$$

Because we always update ϱ when possible, this makes our strategy well-defined, that is, we never use ϱ before defining it.

Let us prove that the play is winning for player $\diamond$. If the play stays in $P' \setminus Q'$, then clearly it is winning for player $\diamond$ because we follow the winning strategy π. So if it is losing, then at some point the play must go to Q'. We can only enter Q' via a vertex $f(v) \in Q'$. When this happens, we choose a partial winning strategy ϱ on $(Q', f(v))$ with an optimal profile that agrees with the profile that π gives us on (Q, v). Furthermore, we follow this ϱ until we leave Q'.

If the play stays in Q', then it is winning for player $\diamond$ because π was winning from (Q, v). If the play leaves Q', then it can only do so via a vertex in U. When leaving Q' via U, the worst priority visited in Q' cannot be worse than the worst priority

that we would have visited in Q following π. Furthermore, we can only leave Q' via a vertex w that we could also use to leave Q in P, so we have that $w \in W_\Diamond(P)$.

In all cases, player $\Box$ cannot force an immediate win but he also cannot force a win by going over bad priorities. In total this means that player $\Diamond$ wins the play.

The other direction follows analogously. ∎

5.2.7 Definable Profiles

In the next step, we would like to encode a profile in a formula. Given a profile y in a model-checking game and a starting point $x = (x', \psi)$, we would like to define a formula ψ^y with the property that ψ^y is true on the vertex x' in the structure if and only if the profile y is possible on (P, x). However, we do not know how to do this.

Hence we weaken the restriction and want ψ^y to be true iff a profile $y' \sqsubseteq y$ is possible. This is enough for our purposes because the type of x cares only about $\sqsubseteq$-minimal profiles. This formula turns out to be definable. Using a suitable definition of ψ^y, we get the following theorem.

Theorem 5.27 *Let $\overline{P}$ be a sequence of proposition symbols disjoint from σ. Let $\varphi \in L_\mu[\sigma \cup P]$, $\mathcal{M}, v$ be a σ-structure and $\overline{X}$ be a sequence of vertices of $\mathcal{M}$. For $\psi \in \mathrm{CL}(\varphi)$, $y \in \mathrm{profiles}(\mathcal{M} \ltimes \varphi)$, it holds that $\mathcal{M}, v \models \psi^y$ iff there is a positional partial winning strategy π for $(\mathcal{M} \ltimes \varphi, (v, \psi))$ such that $\mathrm{profile}(\pi, (v, \psi)) \sqsubseteq y$.*

Corollary 5.28 *Let $\overline{P}$ be a sequence of proposition symbols disjoint from σ. Let $\varphi \in L_\mu[\sigma \cup P]$, $\mathcal{M}, v$ be a σ-structure,*

$\overline{X} \in V(\mathcal{M})^{|P|}$ *and* $\psi \in \mathrm{CL}(\varphi)$. *Then*

$$\mathrm{ptype}_{\mathcal{M}\ltimes\varphi}((v,\psi)) = \Big\{ y \in \mathrm{profiles}(\mathcal{M}\ltimes\varphi) \;\Big|\; \mathcal{M}, v \models \psi^y \text{ and there is no } y' \sqsubset y \text{ with } \mathcal{M}, v \models \psi^{y'} \Big\}.$$

That is, $\mathrm{tp}_{\{\varphi\},\overline{P}}(\mathcal{M}, v, \overline{X})$ *determines* $\mathrm{ptype}_{\mathcal{M}\ltimes\varphi}((v,\psi))$.

Before we can explain ψ^y, we need one more definition.

Definition 5.29 For an annotated $\varphi \in L_\mu[\sigma]$, $\psi \in \mathrm{CL}(\varphi)$ and $\chi \in \mathrm{sub}(\psi)$, let

$$\mathrm{prio}_\varphi(\psi \rightsquigarrow \chi)$$

$\mathrm{prio}_\varphi(\psi \rightsquigarrow \chi)$

be the minimum priority of all fixpoint operators that enclose χ in ψ. ⊣

Definition 5.30 Let $\overline{P} = (P_1, \ldots, P_k)$ be a sequence of proposition symbols disjoint from σ. Let $\varphi \in L_\mu[\sigma \cup P]$ be a formula, $\mathcal{M}$ be a σ-structure and $X = (x_1, \ldots, x_k) \in V(\mathcal{M})^k$. Let $\psi \in \mathrm{CL}(\varphi)$ and $y \in \mathrm{profiles}(\mathcal{M} \ltimes \varphi)$. For every $\psi' \in \mathrm{sub}^+(\psi)$, there is a formula $\varphi' \in \mathrm{CL}(\varphi)$ corresponding to ψ'. We inductively define an operation $\cdot^y$ over the structure of ψ'.

ψ'^y

$$\begin{aligned}
V^y &:= V, \quad (\neg V)^y := \neg V && \text{for prop. or var. } V \\
(\chi * \chi')^y &:= (\chi^y) * (\chi'^y) && \text{for } * \in \{\vee, \wedge\} \\
(\alpha X.\chi)^y &:= \alpha X.(\chi^y) && \text{for } \alpha \in \{\mu, \nu\} \\
(\Diamond\chi)^y &:= \Big(\Big(\bigvee_{i\in N} P_i \Big) \vee \Diamond(\chi^y) \Big) \\
(\Box\chi)^y &:= \Big(\Big(\bigwedge_{i\in N} \neg P_i \Big) \wedge \Box(\chi^y) \Big)
\end{aligned}$$

In the case $\Diamond\chi$, we use

$$N := \{1 \leq i \leq k \mid \\ ((x_i, \varphi'), p') \in y \text{ for some } p' \sqsupseteq \mathrm{prio}_\varphi(\psi \rightsquigarrow \Diamond\chi)\}.$$

In the case $\Box\chi$, we use

$$N := \{1 \leq i \leq k \mid \\ ((x_i, \varphi'), p') \notin y \text{ for all } p' \sqsubset \mathrm{prio}_\varphi(\psi \rightsquigarrow \Box\chi)\}.$$

In both cases, $\varphi' \in \mathrm{CL}(\varphi)$ is the formula corresponding to $\Diamond\chi$ or $\Box\chi$, respectively. ⊣

The motivation behind this seemingly quite arbitrary definition is that if a profile says we can reach $(x_i, \Diamond\chi)$ with the worst priority p', and the actual priority we have is at least as good as p', then we are allowed to take the shortcut and leave the game. That is why we add P_i to the disjunction in this case. Of course, we need to pay close attention to the games that are involved, because $(x_i, \Diamond\chi)$ is not a vertex in $\mathcal{M} \ltimes \varphi$ and y is not a profile of $\mathcal{M} \ltimes \psi$. However, this is not a problem because every $\Diamond\chi$ corresponds to a unique $\varphi' \in \mathrm{CL}(\varphi)$, and the game $\mathcal{M} \ltimes \psi$ is a partial unfolding of the game $\mathcal{M} \ltimes \varphi$. This means that every strategy on one of these games is also a strategy on the other game, although not necessarily positional.

Dually, in the case $\Box\chi$, if the actual priority is worse than what the profile wants, then we must make sure that $(x_i, \Box\chi)$ is not reached, so we add $\neg P_i$ to the conjunction.

A formal statement of this explanation is Theorem 5.27. Before we can prove this, however, we need a technical lemma about $\mathrm{prio}_\varphi(\psi \rightsquigarrow \chi)$.

Lemma 5.31 *Let $\mathcal{M}$ be a structure, $v, x \in V(\mathcal{M})$ and $\varphi \in L_\mu$, $\psi \in \mathrm{CL}(\varphi)$ and $\chi \in \mathrm{sub}(\psi)$. Then every path from (v, ψ) to (x, χ) in $\mathcal{M} \ltimes \psi$ (with priorities according to φ) has $\mathrm{prio}_\varphi(\psi \rightsquigarrow \chi)$ as its minimum priority.*

Proof. Let p be the minimum priority of a path from (v, ψ) to (x, χ). Clearly $p \leq \mathrm{prio}_\varphi(\psi \rightsquigarrow \chi)$ because χ is a subformula of ψ, so every fixpoint operator enclosing χ must have been visited at some point on the path.

Assume to the contrary that $p < \mathrm{prio}_\varphi(\psi \rightsquigarrow \chi)$. This means that there is a vertex $(v', \overset{p}{\alpha} X.\psi')$ with $\alpha \in \{\mu, \nu\}$ on the path. Assume this is the first vertex of priority p on the path. The priorities increase with respect to a fixed sequence of variables $\overline{Z}$, so ψ' cannot contain a free variable Y for any Y that is quantified earlier, or p would have to be larger. But this means that $\overset{p}{\alpha} X.\psi'$ is a closed formula. So in order to reach (x, χ), the formula χ must be a subformula of $\overset{p}{\alpha} X.\psi'$, and we have that $\overset{p}{\alpha} X$ encloses χ, a contradiction to $p < \mathrm{prio}_\varphi(\psi \rightsquigarrow \chi)$. ∎

We split the proof of Theorem 5.27 into two directions. Lemma 5.32 shows the first direction and Lemma 5.33 the other.

Lemma 5.32 *Let $\overline{P}$ be a sequence of k proposition symbols disjoint from σ. Let $\varphi \in L_\mu[\sigma \cup P]$, $\mathcal{M}, v$ be a σ-structure, $\overline{X} \in V(\mathcal{M})^k$, $\psi \in \mathrm{CL}(\varphi)$, $y \in \mathrm{profiles}(\mathcal{M} \ltimes \varphi)$. Let π be a partial winning strategy for $(\mathcal{M} \ltimes \psi, (v, \psi))$ and π_φ be the corresponding strategy on $\mathcal{M} \ltimes \varphi$. If $\mathrm{profile}(\pi_\varphi, (v, \psi)) \sqsubseteq y$, then there exists a winning strategy π' for $(\mathcal{M} \ltimes \psi^y, (v, \psi^y))$.*

Proof. Let π be as required. Without loss of generality we are going to assume that π_φ is a positional strategy. According to

Lemma 5.23, this is always possible. Then π can be chosen to be positional, too.

There is an obvious mapping from $\mathrm{sub}(\psi)$ to $\mathrm{sub}(\psi^y)$ because ψ^y is only a slightly modified version of ψ.

Define π' positionally on $\mathcal{M} \ltimes \psi^y$ so that it follows π wherever possible using the mapping we just described. The only points where π' is undefined are the vertices of the form $\overline{w} := (w, \bigvee_{i \in N} P_i \vee \Diamond\chi)$. On these vertices, if $w = x_i \in X$ for some $i \in N$, define $\pi'(\overline{w}) = (w, P_i)$. Otherwise, define $\pi'(\overline{w}) = (w, \Diamond\chi)$. We claim that π' is a strategy on $(\mathcal{M} \ltimes \psi^y, (v, \psi^y))$.

Let $(x_i, \Diamond\chi) \in \mathcal{M} \ltimes \psi^y$ be such that $(x_i, \Diamond\chi)$ is reachable in $\mathcal{M} \ltimes \psi^{y\pi'}$ from (v, ψ^y) but $(x_i, \Diamond\chi) \notin \mathrm{dom}(\pi')$. Let $(v_1, \ldots, v_n)$ be a π'-conforming path with $v_1 = (v, \psi^y)$ and $v_n = (x_i, \Diamond\chi)$ with minimum priority p. This path corresponds to a π-conforming path in $\mathcal{M} \ltimes \psi$ starting from (v, ψ) with the same minimum priority, and hence a π_φ-conforming path in $\mathcal{M} \ltimes \varphi$ with the same minimum priority. By Lemma 5.31, we have $p = \mathrm{prio}_\varphi(\psi \rightsquigarrow \Diamond\chi)$.

Let $\Diamond\chi' \in \mathrm{CL}(\varphi)$ be the unique subformula of φ corresponding to $\Diamond\chi$. Then we have

$$((x_i, \Diamond\chi'), p') \in \mathrm{profile}(\pi_\varphi, (v, \psi))$$

for some $p' \sqsupseteq p$ and hence $((x_i, \Diamond\chi'), p'') \in y$ for some $p'' \sqsupseteq p'$. By the construction of ψ^y, the vertex $(x_i, \Diamond\chi)$ in $\mathcal{M} \ltimes \psi^y$ must have a unique predecessor $(x_i, \bigvee_{i \in N} P_i \vee \Diamond\chi)$ for some set N. Recall the definition of N,

$$N := \{1 \leq i \leq k \mid (x_i, \Diamond\chi', q) \in y \text{ for some } q \sqsupseteq \mathrm{prio}_\varphi(\psi \rightsquigarrow \Diamond\chi)\}.$$

We find that $i \in N$, so the path was not π'-conforming.

We need to show that π' is winning. Let $(v_1, \ldots, v_n)$ be a maximal π'-conforming path in $\mathcal{M} \ltimes \psi^y$ with $v_1 = (v, \psi^y)$. By definition of π', the last vertex cannot be (w, P_i) for $w \neq x_i$.

Assume $v_n \in V_\Diamond$, that is, the path is losing. The same path can be viewed as a maximal π-conforming path in $\mathcal{M} \ltimes \psi$. In $\mathcal{M} \ltimes \psi$, the last vertex is also in $V_\Diamond$ and has no successors, so we would have a π-conforming losing path in $\mathcal{M} \ltimes \psi$, which contradicts the assumption that π was a partial winning strategy.

Clearly all infinite paths starting from $(\mathcal{M} \ltimes \psi^y, (v, \psi^y))$ can never visit a vertex of the form (w, P_i), so they can be viewed as paths on $\mathcal{M} \ltimes \psi$. They visit exactly the same priorities. This implies that π' is a winning strategy. ∎

Lemma 5.33 *Let $\overline{P}$ be a sequence of k proposition symbols disjoint from σ. Let $\varphi \in L_\mu[\sigma \cup P]$, $\mathcal{M}, v$ be a σ-structure, $\overline{X} \in V(\mathcal{M})^k$, $\psi \in \mathrm{CL}(\varphi)$, $y \in \mathrm{profiles}(\mathcal{M} \ltimes \varphi)$. Let π' be a winning strategy for $(\mathcal{M} \ltimes \psi^y, (v, \psi^y))$.*

Then there exists a partial winning strategy π for $(\mathcal{M} \ltimes \psi, (v, \psi))$ such that for the corresponding partial strategy π_φ on $\mathcal{M} \ltimes \varphi$ it holds that $\mathrm{profile}(\pi_\varphi, (v, \psi)) \sqsubseteq y$.

Proof. Let π' be as required. Assume π' is a positional winning strategy. Define π (positionally) like π' where possible. If $\pi'((w, \bigvee_{i \in N} P_i \vee \Diamond\chi)) = (x_i, P_i)$ for some $\overline{w} \in V(\mathcal{M} \ltimes \psi^y)$, N and i, then leave $\pi'((x_i, \Diamond\chi))$ undefined.

Similar to the proof of the previous lemma one shows that π' is a partial winning strategy and $\mathrm{profile}(\pi_\varphi, (v, \psi)) \sqsubseteq y$. ∎

5.2.8 A Small Parity Game

With Theorem 5.27 at our hands, we can now define a partial parity game simulating the model-checking game that depends only on the types of some vertices in the original structure. The parity game consists of four layers of vertices.

1. One layer of $\Diamond$-vertices, one for each type, where player $\Diamond$ can choose a profile.
2. Then one layer of $\Box$-vertices, one for each profile, where player $\Box$ can choose one of the allowed paths.
3. Then a layer of vertices with out-degree 1 to ensure the priorities match the chosen path.
4. Finally a layer representing the interface.

The arcs point only from one layer to the next or from the last layer back to the first layer. Formally, let $\mathcal{M}$ be a structure and $X = \{x_1, \ldots, x_k\} \subseteq V(\mathcal{M})$. Let $\varphi \in L_\mu$. First, we define the layers described above.

$$V_1 := 2^{\text{profiles}(\mathcal{M} \ltimes_X \varphi)} \qquad V_3 := \text{strategy-targets}(\mathcal{M} \ltimes_X \varphi)$$
$$V_2 := \text{profiles}(\mathcal{M} \ltimes_X \varphi) \quad V_4 := X \times \text{CL}(\varphi).$$

P^φ Next, we define the game $P^\varphi = (V, V_\Diamond, E, \omega)$ with interface V_4 depending only on φ and the sets $\text{tp}_{\{\varphi\},\overline{P}}(\mathcal{M}, x_i, \overline{X})$, but not on $\mathcal{M}$.

$$V := V_1 \cup V_2 \cup V_3 \cup V_4 \qquad E := E_1 \cup E_2 \cup E_3 \cup E_4$$
$$V_\Diamond := V_1 \cup \{(x_i, \psi) \in V_4 \mid \psi \text{ starts with a } \Diamond\}$$

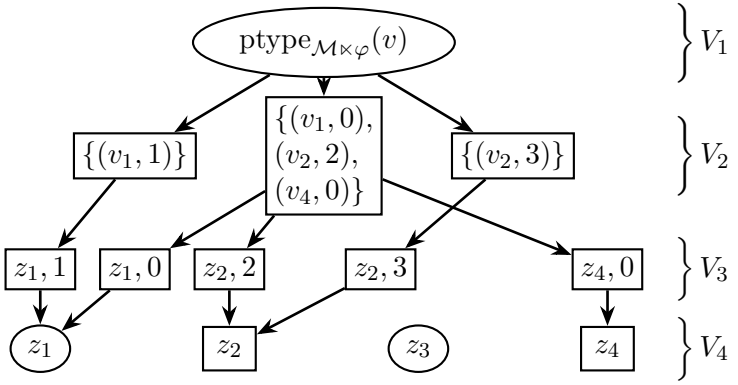


Figure 5.6: A part of P^{φ}

$$\omega(v) := \begin{cases} p & \text{for } v = (x_i, \psi, p) \in V_3 \\ p' & \text{otherwise,} \end{cases}$$

where p' is the maximum priority of φ.

For the set of arcs, we connect the vertices according to the subset relation and the vertices from V_4 back to their types.

$$\begin{aligned} E_1 &:= \{(x,y) \in V_1 \times V_2 \mid y \in x\} \\ E_2 &:= \{(x,y) \in V_2 \times V_3 \mid y \in x\} \\ E_3 &:= \{((x_i,\psi,p),(x_i',\psi')) \in V_3 \times V_4 \mid (x_i,\psi) = (x_i',\psi')\} \\ E_4 &:= \left\{(x,t) \in V_4 \times V_1 \;\middle|\; t = \text{ptype}_{\mathcal{M}\ltimes\varphi}(x)\right\}. \end{aligned}$$

Note that E_4 is determined by the sets $\text{tp}_{\{\varphi\},\overline{P}}(\mathcal{M}, x, \overline{X})$ by Corollary 5.28.

To illustrate this construction, assume that $\mathcal{M} \ltimes \varphi$ has the

interface $\{z_1, \ldots, z_4\} \subseteq X \times \mathrm{CL}(\varphi)$ and a vertex $v \in \mathcal{M} \ltimes \varphi$ with

$$\mathrm{ptype}_{\mathcal{M}\ltimes\varphi}(v) = \\ \{\{(z_1, 1)\}, \{(z_1, 0), (z_2, 2), (z_4, 0)\}, \{(z_2, 3)\}\}.$$

Figure 5.6 illustrates a part that could occur in the game P^φ. In the full game P^φ, we would also add the arcs

$$(z_i, \mathrm{ptype}_{\mathcal{M}\ltimes\varphi}(z_i)) \in V_4 \times V_1.$$

In the vertex $\mathrm{ptype}_{\mathcal{M}\ltimes\varphi}(v)$, player $\diamond$ can choose one of the possible profiles. This corresponds to player $\diamond$ fixing a strategy π. After fixing her strategy, player $\Box$ can choose a path through the game conforming to this strategy. The profile tells us exactly what the worst possible paths are, and the layer V_3 makes sure that the correct priority is visited.

The goal of this construction is to get a game such that the type of a vertex labeled $\mathrm{ptype}_{\mathcal{M}\ltimes\varphi}(v)$ is exactly $\mathrm{ptype}_{\mathcal{M}\ltimes\varphi}(v)$. This leads to the main theorem of this subsection.

Theorem 5.34 *For a formula $\varphi \in L_\mu$, a structure $\mathcal{M}$ and $X \subseteq V(\mathcal{M})$, the game P^φ simulates $\mathcal{M} \ltimes_X \varphi$.*

Proof. For every vertex $u \in X \times \mathrm{CL}(\varphi)$, define $f(u) = u$. For the remaining vertices $v \in V(\mathcal{M} \ltimes \varphi) \setminus (X \times \mathrm{CL}(\varphi))$, define $f(v) = \mathrm{ptype}_{\mathcal{M}\ltimes\varphi}(v) \in V_\diamond(P^\varphi)$.

All we have to do now is to show that $\mathrm{ptype}_{\mathcal{M}\ltimes\varphi}(v) = \mathrm{ptype}_{P^\varphi}(f(v))$ for all $v \in V(\mathcal{M} \ltimes \varphi)$. First we show $\subseteq$.

Let π be a positional partial winning strategy for $(\mathcal{M} \ltimes \varphi, (v, \psi))$. We want to construct a positional partial winning strategy π' for $(P^\varphi, f((v, \psi)))$ such that $\mathrm{profile}(\pi, (v, \psi)) =$

$\mathrm{profile}(\pi', f((v,\psi)))$.

For every vertex $(v,\psi) \in \mathcal{M} \ltimes \varphi$, define

$$\pi'(\mathrm{ptype}_{\mathcal{M}\ltimes\varphi}((v,\psi))) := \mathrm{profile}(\pi,(v,\psi)).$$

For $(x_i,\psi) \in V_\Diamond(P^\varphi)$, if $(x_i,\psi) \in \mathrm{dom}(\pi)$, then we define

$$\pi'((x_i,\psi)) := \mathrm{ptype}_{\mathcal{M}\ltimes\varphi}(x_i).$$

Otherwise, leave $\pi'((x_i,\psi))$ undefined.

We claim that π' is a partial winning strategy on $(P^\varphi,(v,\psi))$. By Theorem 5.27, for all $(x_i,\chi) \in X \times \mathrm{CL}(\varphi)$ it holds that $\mathcal{M}, x_i \models \chi^{\mathrm{profile}(\pi,(x_i,\chi))}$. So the unique arc leaving from (x_i,ψ) in P^φ goes to some vertex y with $\mathrm{profile}(\pi,(x_i,\psi)) \in y$.

Inductively it follows that every π'-conforming path in P^φ corresponds to a π-conforming path in $\mathcal{M} \ltimes \varphi$ and vice versa. So π' is a partial winning strategy with $\mathrm{profile}(\pi,(v,\psi)) = \mathrm{profile}(\pi', f((v,\psi)))$.

It remains to show the other direction $\mathrm{ptype}_{\mathcal{M}\ltimes\varphi}(v) \supseteq \mathrm{ptype}_{P^\varphi}(f(v))$.

Let π' be a $\sqsubseteq$-minimal positional partial winning strategy for $(P^\varphi, f((v,\psi)))$, that is, an element of $\mathrm{ptype}_{P^\varphi}(f(v))$. We want to construct a partial winning strategy π for $(\mathcal{M} \ltimes \varphi,(v,\psi))$ such that

$$\mathrm{profile}(\pi,(v,\psi)) = \mathrm{profile}(\pi', f((v,\psi))).$$

We define a non-positional partial winning strategy π with this property as we go, starting from (v,ψ). In addition to the token on $\mathcal{M} \ltimes \varphi$ that we use to play the game in the normal way, player $\Diamond$ places and moves a token on P^φ in order to keep track of her choices. Initially she places the token on

$\pi'(f(v,\psi))$. We uphold the invariant that the token on P^φ is always in $V_2(P^\varphi)$. Because V_2 contains only profiles, let us introduce the abbreviation

$$g(w) := \pi,$$

where π is some strategy such that $w = \text{profile}(\pi, x)$ for some $x \in X \times \text{CL}(\varphi)$.

Suppose that the current position is some $(v,\psi) \in \mathcal{M} \ltimes \varphi$ and the token on P^φ is on some vertex $w \in V_2(P^\varphi)$.

If $(v,\psi,p) \in w$ for some p, then player $\diamond$ moves the token on P^φ as follows.

$$\begin{aligned} w &\to (v,\psi,p) \\ &\to (v,\psi) \\ &\to \text{ptype}_{\mathcal{M}\ltimes\varphi}((v,\psi)) \\ &\to \pi'(\text{ptype}_{\mathcal{M}\ltimes\varphi}((v,\psi))) \in V_2(P^\varphi). \end{aligned}$$

If $(v,\psi,p) \notin w$ for all p, then player $\diamond$ does not touch the token on P^φ. If she moved the token, then the vertex

$$\pi'(\text{ptype}_{\mathcal{M}\ltimes\varphi}((v,\psi))) \in V_2(P^\varphi)$$

is the new w.

Then, if $(v,\psi) \in V_\diamond(\mathcal{M} \ltimes \varphi)$, player $\diamond$ moves in $\mathcal{M} \ltimes \varphi$ to the vertex $g(w)(v,\psi)$ (or stops moving if $g(w)(v,\psi)$ is undefined), and the process starts from the beginning.

We see that this defines a non-positional partial strategy π for player $\diamond$ on $\mathcal{M} \ltimes \varphi$ starting from (v,ψ). We also see that player $\diamond$ consistently follows one strategy until she moves the token in P^φ. She moves the token in P^φ only if she hits a vertex

(v, ψ) with $(v, \psi, p) \in w$ for some p. By the definition of profile, this means that the worst priority q that she encountered in $\mathcal{M} \ltimes \varphi$ since she started following $g(w)$, satisfies $q \sqsubseteq p$.

Player $\diamond$ then moves the token in P^φ according to her partial winning strategy π' and according to a specific move by player $\Box$ that she chooses. In particular, she moves the token in P^φ through a vertex of priority p. This means that every priority q visited in $\mathcal{M} \ltimes \varphi$ is trumped by a priority $p \sqsupseteq q$ in P^φ. Note also that the path the token takes in P^φ is π'-conforming. Because π' is a partial winning strategy in P^φ, it follows that π is a partial winning strategy in $\mathcal{M} \ltimes \varphi$.

The argument also implies

$$\text{profile}(\pi, (v, \psi)) \sqsubseteq \text{profile}(\pi', f((v, \psi))).$$

Furthermore, by Lemma 5.23, we can assume that π is a positional partial winning strategy with this property.

As we saw when proving the other direction, we can construct from π a partial winning strategy π'' for $(P^\varphi, f((v, \psi)))$ such that

$$\text{profile}(\pi, (v, \psi)) = \text{profile}(\pi'', f((v, \psi))).$$

From the definition of ptype() it follows that

$$\text{profile}(\pi, (v, \psi)) = \text{profile}(\pi', (v, \psi)).$$ ∎

5.2.9 Finishing the Proof

With Theorem 5.34, we finally have the necessary tool to conclude the proof of the decomposition theorems from pages 114 and 115.

Proof of Theorem 5.13. Fix some $\varphi \in \mathrm{CL}_P(L)$. Consider the model-checking game $\mathcal{M} \ltimes \varphi$ and the induced subgame $\mathcal{M}_2 \ltimes \varphi$ with interface U. We can assume that

$$V(\mathcal{M}_2 \ltimes \varphi) \cap V(\mathcal{M} \ltimes \varphi) = U$$

by duplicating some vertices as necessary.

The game $\mathcal{M}_2 \ltimes \varphi$ is simulated by P^φ, constructed as described in Theorem 5.34. By Lemma 5.26, we can replace $\mathcal{M}_2 \ltimes \varphi$ by P^φ (by properly adapting the arcs) without changing the winner on (v, φ). Since the construction of P^φ depends only on the types of the vertices in X, we will get the same game P^φ if we start the construction with $\mathcal{M}_2'$.

Let (v, w) be an arc from $\mathcal{M}_1 \setminus X$ to $\mathcal{M}_2 \setminus X$ and let $w' \in \mathcal{M}_2'$ be the vertex chosen as the replacement for w. Because $\mathrm{tp}_{\{\varphi\},\overline{P}}(\mathcal{M}_2, w, \overline{X})$ determines $\mathrm{ptype}_{\mathcal{M}_2 \times \varphi}((w, \varphi))$ by Corollary 5.28 and we have

$$\mathrm{tp}_{\{\varphi\},\overline{P}}(\mathcal{M}_2, w, \overline{X}) \subseteq \mathrm{tp}_{L,\overline{P}}(\mathcal{M}_2, w, \overline{X}),$$

it follows that

$$\mathrm{ptype}_{\mathcal{M}_2 \times \varphi}((w, \varphi)) = \mathrm{ptype}_{\mathcal{M}_2' \times \varphi}((w', \varphi)).$$

So in the simulation, the arc will point to the same vertex no matter if we started with $\mathcal{M}_2$ or $\mathcal{M}_2'$. ∎

Proof of Theorem 5.14. The first part is essentially a different way of stating Theorem 5.13 which follows immediately with the same argument as in the previous proof.

Note that we may assume without loss of generality that $X \cap Y = \emptyset$. If this is not the case, then we have $x_i = y_j$ for

some $x_i \in X, y_j \in Y$ and the propositional variables $X_i \in P$ and $Y_j \in Q$ will be interchangeable.

Set $L' := \mathrm{CL}_Q(L)$. Theorem 5.13 states that $\mathrm{tp}_{L',\emptyset}(\mathcal{M}, v, \emptyset)$ is invariant under $\mathrm{CL}_\emptyset(L')$-equivalent directed separations for all $v \in \mathcal{M}_1$. All we need to show is that the requirements listed in Theorem 5.14 specify the directed separation $(\mathcal{M}_1, \mathcal{M}_2)$ up to $\mathrm{CL}_\emptyset(L')$-equivalence.

For all vertices $w \in \mathcal{M}_2$, the set $\mathrm{tp}_{L',\overline{P}}(\mathcal{M}_2, w, \overline{X})$ can be computed from $\mathrm{tp}_{L,\overline{P}}(\mathcal{M}_2, w, \overline{X})$; a propositional variable $Y_i \in Q$ corresponding to a vertex $y_i \in Y$ is always false in $\mathcal{M}_2$. From this we can easily compute $\mathrm{tp}_{L',\emptyset}(\mathcal{M}_2, w, \emptyset)$ by forgetting about $\overline{P}$.

The computability in the above argument follows from the observation that all sets involved are finite in size and the model checking for L_μ is decidable.

For the second part, let $\varphi \in \mathrm{CL}_Q(L)$. We want to decide whether $\mathcal{M}, w \models \varphi$. By the first part, we already know the sets $\mathrm{tp}_{L,\overline{Q}}(\mathcal{M}, x_i, \overline{Y})$ for all $x_i \in X$. Consider the model-checking game $\mathcal{M} \ltimes \varphi$. In this game, the vertices of the form (v, Y_i) with $v \in \mathcal{M}_1$ are always losing because $Y_i \in Q$ is never true in $\mathcal{M}_2$. It follows that the subgame $\mathcal{M}_2 \ltimes \varphi$ is isomorphic to $\mathcal{M}_2 \ltimes \varphi'$, where φ' is constructed from φ by replacing all $Y_i \in Q$ by $\bot$. Note that $\varphi' \in \mathrm{CL}_P(L)$, so we know all optimal partial strategies for $(\mathcal{M}_2 \ltimes \varphi', (w, \varphi'))$ because we know $\mathrm{tp}_{L,\overline{P}}(\mathcal{M}_2, w, \overline{X})$. It follows that the winner is determined by the remaining sets given in the theorem. ∎

5.3 Running Time

5.3.1 Upper Bound

Theorem 5.14 claims that $\mathrm{tp}_{L,\overline{Q}}(\mathcal{M}, v, \overline{Y})$ can be computed from

- $\mathcal{M}_1$ and $\overline{Q}$ and
- $\left\{(x_i, \mathrm{tp}_{L,\overline{P}}(\mathcal{M}_2, x_i, \overline{X})) \;\middle|\; x_i \in X\right\}$ and
- $\Big\{(v, \mathrm{tp}_{L,\overline{P}}(\mathcal{M}_2, w, \overline{X})) \;\Big|\; (v,w) \in E(\mathcal{M}) \cap (\mathcal{M}_1 \times \mathcal{M}_2)\Big\}$.

Let us analyze the previous proof to find the running time for this computation.

First, let us prove an upper bound on the number of formulas in $\mathrm{PT}_P(\varphi)$ and on the length of formulas in the closure $\mathrm{CL}(\varphi)$.

Lemma 5.35 $|\mathrm{PT}_P(\varphi)| \in O(2^{|P|}|\varphi|)$.

Proof. The inductive definition of $\mathrm{PT}_P(\varphi)$ has depth at most $|\varphi|$ and at each step we collect at most 2^P many formulas, so the bound follows. ∎

Lemma 5.36 *For all $\varphi \in L_\mu$, $|\varphi| = n$ and $\psi \in \mathrm{CL}(\varphi)$ it holds that $|\psi| \leq n^{n-1}$.*

Proof. Fix a $\varphi \in L_\mu$ and let $|\varphi| = n$. We proceed via induction over the recursive definition of the Fischer-Ladner closure. This gives us a finite tree of unique formulas, every formula labeled by their depth in this tree. The root φ has depth 0, and the

formulas directly below φ have depth 1, etc., until the tree is complete.

Let X be fixpoint variable. We say that a specific occurrence of X in a formula $\psi \in \mathrm{CL}(\varphi)$ is *relevant* if the depth of the enclosing μX or νX formula is not smaller than the depth of ψ. In particular we need that if $\psi = \mu X.\chi$ has at most k relevant occurrences of X, then χ has at most k free occurrences of X.

The induction hypothesis has two parts. If ψ occurs at depth $i-1$ in the tree, then

1. $|\psi| \leq n^{i-1}$ and
2. for every variable X, the formula ψ contains at most $n-1$ relevant occurrences of X.

For $i = 1$ this is obviously true because φ itself has no free variables.

The only place where the length of a formula or the number of occurrences of variables could increase is the case $\mu X.\varphi$ (or $\nu X.\varphi$, which is identical). Let $\mu X.\varphi$ be at depth $i-1$ in the tree. By the induction hypothesis we have $|\mu X.\varphi| \leq n^{i-1}$ and φ contains at most $n-1$ free occurrences of X. It follows that

$$|\varphi[X/\mu X.\varphi]| \leq n^{i-1} + (n-1) \cdot n^{i-1} = n^i.$$

We also see that $\varphi[X/\mu X.\varphi]$ contains no relevant occurrences of X. Additionally, for every variable Z different from X, all new occurrences of Z introduced by the substitution are not relevant, so the second part of the induction hypothesis also holds.

The depth of $\psi \in \mathrm{CL}(\varphi)$ can never be larger than $n-1$ because the number of subformulas of φ is bounded by n. It follows that every formula in $\mathrm{CL}(\varphi)$ has length at most n^{n-1}.■

Corollary 5.37 *Let $\varphi \in L_\mu$ with $|\varphi| = n$. Then the following are true.*

1. *For all $\psi \in \mathrm{CL}_P(\varphi)$, it holds that $|\psi| \in O(|P|n^{n-1})$.*
2. *$|\mathrm{CL}_P(\varphi)| \in O(2^{|P|}n^n)$.*
3. *$|V(P^\varphi)| \in O\left(2^{2^{|P|n}}\right)$.*

Proof.

1. Let $\psi' \in \mathrm{CL}(\varphi)$ be such that $\psi \in \mathrm{PT}_P(\psi')$. By the previous lemma we have $|\psi'| \leq n^{n-1}$. In the definition of PT_P, we blow up every box and every diamond by a factor of at most $|P|$, so the bound follows.
2. We have $|\mathrm{CL}(\varphi)| \in O(n)$ and $|\mathrm{PT}_P(\psi)| \in O(2^{|P|}|\psi|)$. The bound then follows with the previous lemma.
3. We have $\mathrm{profiles}(\mathcal{M} \ltimes \varphi) \in O(2^{|P|n})$. In the construction of P^φ, the powerset of this set dominates the size of the game. ∎

Let us combine these results to get the time needed to compute $\mathrm{tp}_{L,\overline{Q}}(\mathcal{M}, v, \overline{Y})$ from $\mathcal{M}_1$ and the types of $\mathcal{M}_2$ in Theorem 5.14, where $(\mathcal{M}_1, \mathcal{M}_2)$ is a directed separation with interface X and $L = \{\varphi\}$. First let us observe the cost to compute the type from scratch.

Lemma 5.38 *Given a σ-structure $\mathcal{M}$ with $|V(\mathcal{M})| = n$, a vertex $v \in \mathcal{M}$, a sequence of proposition symbols $\overline{P}$, a set of vertices $\overline{X}$ and a formula $\varphi \in L_\mu[\sigma]$, the type $\mathrm{tp}_{\{\varphi\},\overline{P}}(\mathcal{M}, v, \overline{X})$ can be computed in time*

$$O\left(2^{|P|}\left(n|P||\varphi|^{|\varphi|}\right)^{3+|\varphi|}\right).$$

Proof. We recall the definition

$$\mathrm{tp}_{\{\varphi\},\overline{P}}(\mathcal{M}, v, \overline{X}) = \left\{\psi \in \mathrm{CL}_P(\varphi) \;\middle|\; \partial_{\overline{P}}(\mathcal{M}, \overline{X}), v \models \psi\right\}$$

and fix some $\psi \in \mathrm{CL}_P(\varphi)$.

The parity game $\partial_{\overline{P}}(\mathcal{M}, \overline{X}) \ltimes \psi$ has size $n \cdot |\psi|$. There are many algorithms for solving parity games, but so far they are all super-polynomial in the number of priorities, so for simplicity we use one of the earliest and easiest algorithms, the algorithm by McNaughton [McN93]. This algorithm has a running time of $O(n^{2+l})$ where n is the number of vertices in the game and l the number of different priorities.

Note that ψ has exactly the same priorities as φ because going through PT_P does not affect the fixpoint operators, leaving the priorities unaffected. So computing the winning regions of the parity game $\partial_{\overline{P}}(\mathcal{M}, \overline{X}) \ltimes \psi$ can be done in time $O((n|\psi|)^{2+|\varphi|})$. Together with the upper bounds

$$|\psi| \in O(|P||\varphi|^{|\varphi|-1})$$

and

$$|\mathrm{CL}_P(\varphi)| \in O(2^{|P|}|\varphi|^{|\varphi|}),$$

from Corollary 5.37, we get a total running time of

$$\begin{aligned} &O\left(2^{|P|}|\varphi|^{|\varphi|}\left(n|P||\varphi|^{|\varphi|-1}\right)^{2+|\varphi|}\right) \\ &\quad\subseteq O\left(2^{|P|}\left(n|P||\varphi|^{|\varphi|}\right)^{3+|\varphi|}\right). \end{aligned}$$

■

In the proof of Theorem 5.14, we solve a parity game consist-

ing of P^ψ and $\mathcal{M}_1 \ltimes \psi$ for $\psi \in \mathrm{CL}_Q(\varphi)$. By the construction, this parity game has $|\varphi|$ many priorities because even though ψ is longer than φ, we use the same priorities to label the fixpoints.

In total we need to solve $O(2^{|Q|}|\varphi|^{|\varphi|})$ many parity games, one for each formula in $\mathrm{CL}_Q(\varphi)$. Furthermore, each parity game has $O\Big(2^{2^{|Q||\psi|}}\Big)$ vertices.

With this algorithm we can solve one of the parity games in time

$$O\Big(2^{(2+|\varphi|)2^{|Q||\psi|}}\Big) \subseteq O\Big(2^{(2+|\varphi|)2^{|Q||\varphi|^{|\varphi|}}}\Big).$$

We need to solve $O(2^{|Q|}|\varphi|^{|\varphi|})$ many parity games, so in total we need time

$$O\Big(|\varphi|^{|\varphi|}2^{|Q|+(2+|\varphi|)2^{|Q||\varphi|^{|\varphi|}}}\Big). \qquad (*)$$

Because

$$O\Big(|\varphi|^{|\varphi|}2^{|Q|}\Big) \subseteq O\Big(2^{|Q||\varphi|^{|\varphi|}}\Big)$$

and

$$O\Big(|\varphi|2^{|Q||\varphi|^{|\varphi|}}\Big) \subseteq 2^{O\left(|Q||\varphi|^{|\varphi|}\right)},$$

we have that (*) is bounded by

$$O\Big(2^{(3+|\varphi|)2^{|Q||\varphi|^{|\varphi|}}}\Big) \subseteq 2^{2^{O\left(|Q||\varphi|^{|\varphi|}\right)}},$$

and this is an upper bound on the time needed to compute

$\mathrm{tp}_{L,\overline{Q}}(\mathcal{M}, v, \overline{Y})$. So we have the following theorem.

Theorem 5.39 *There is an algorithm that computes the type* $\mathrm{tp}_{L,\overline{Q}}(\mathcal{M}, v, \overline{Y})$ *as in Theorem 5.14 in time*

$$2^{2^{O\left(|Q||\varphi|^{|\varphi|}\right)}}.$$

5.3.2 Lower Bound

Let us now show a lower bound. We will show that formulas in $\mathrm{CL}(\varphi)$ can have exponentially more diamonds than φ, which implies that the types have an exponential number of elements. It follows that every algorithm working on the types will have a running time of at least exponential in $|\varphi|$.

Theorem 5.40 *For every* $c \in \mathbb{N}$ *with* $c > 1$ *there exists a sequence* $\varphi_1, \varphi_2, \ldots \in L_\mu$ *such that*

1. $|\varphi_n| \in O(c \cdot n)$ *and*
2. $\max\{|\psi|_\Diamond \mid \psi \in \mathrm{CL}(\varphi_n)\} \in \Omega(c^n)$.

where $|\psi|_\Diamond$ *is the number of* $\Diamond$ *modalities in* ψ.

Proof. We define

$$\varphi_n := \mu X_1.\mu X_2.\cdots\mu X_n.\bigvee_{i=1}^{n}\bigvee_{j=1}^{c}\Diamond X_i.$$

Clearly we have $|\varphi_n| \in O(c \cdot n)$. It remains to show that the length of the longest formula in $\mathrm{CL}(\varphi_n)$ grows exponentially.

For $i \in \{1, \ldots, n-1\}$ we define inductively

$$\psi_1 := \varphi_n$$

$$\psi_{i+1} := \chi[X_i/\mu X_i.\chi] \qquad \text{where } \psi_i = \mu X_i.\chi.$$

We have $\psi_i \in \mathrm{CL}(\varphi)$ because the definition above follows the definition of the Fischer-Ladner closure, and we already saw with Lemma 3.29 that this is essentially equivalent to CL.

We also observe that the formula with the most $\Diamond$ modalities in $\mathrm{CL}(\varphi_n)$ is among the ψ_i: All subformulas of ψ_n starting with a μ have already been considered, so we never go the μ rule of the Fischer-Ladner closure again and we never add new diamonds.

Let $i \in \{1, \ldots, n-1\}$. Then ψ_i is of the form $\psi_i = \mu X_i.\chi_i$. By induction it follows that χ_i has exactly c free occurrences of X_i. So the step from ψ_i to ψ_{i+1} replaces c occurrences of X_i by ψ_i. Every ψ_i contains at least c occurrences of X_n. Together this implies that ψ_n contains c^n occurrences of X_n.

By construction every occurrence of X_n is preceded by a diamond. This shows that $|\psi_n|_\Diamond \in \Omega(c^n)$, which concludes the proof. ∎

The formula used in the above proof is inspired by the construction in [BFL15, Theorem 3.1]. The theorem implies in particular that

$$|\mathrm{CL}_P(\varphi)| = |\mathrm{PT}_P(\mathrm{CL}(\varphi))|$$

can be exponential in $|P|$ because PT_P has $2^{|P|}$ different choices at each diamond.

Because we will use $\mathrm{CL}_P(\varphi)$ a lot when we construct algorithms in Section 5.4, this already shows that all types that we consider might have exponentially many formulas and all algorithms have at least an exponential worst case running time.

5.4 FPT Algorithms for L_μ Model Checking

In this section we derive two algorithmic applications of Theorem 5.14. More precisely, we show that L_μ-model-checking is fixed-parameter tractable on any class of structures of bounded Kelly-width or bounded DAG-width, provided a decomposition is given as part of the input.

Before proving our results, we develop some algorithmic concepts common to both proofs. We first need an algorithmic version of L-equivalence.

In the following, let σ be a signature, $\overline{P}$ be a sequence of propositional symbols of the appropriate length disjoint from σ and let $L \subseteq L_\mu[\sigma \cup P]$.

5.4.1 Weak Separations

Definition 5.41 Let $\mathcal{M}$ be a σ-structure. A pair $(\mathcal{M}_1, \mathcal{M}_2)$ of induced substructures is a *weak* directed σ-separation of $\mathcal{M}$ with interface $\overline{X} = (x_1, \ldots, x_k)$ if

weak directed sep.

- $V(\mathcal{M}) = V(\mathcal{M}_1) \cup V(\mathcal{M}_2)$,
- $X = \{x_1, \ldots, x_k\} \subseteq V(\mathcal{M}_1) \cap V(\mathcal{M}_2)$,
- there are no arcs from $\mathcal{M}_2 \setminus X$ to $\mathcal{M}_1 \setminus \mathcal{M}_2$. ⊣

Recalling Definition 5.2 on page 101, we see that every directed separation is a weak directed separation because the only difference is that in directed separations the interface X is exactly the intersection $V(\mathcal{M}_1) \cap V(\mathcal{M}_2)$, whereas in weak separations the intersection may be larger. However, vertices in the intersection but not in X cannot have outgoing arcs pointing to $\mathcal{M}_1 \setminus \mathcal{M}_2$. Figure 5.7 on the following page shows the possible

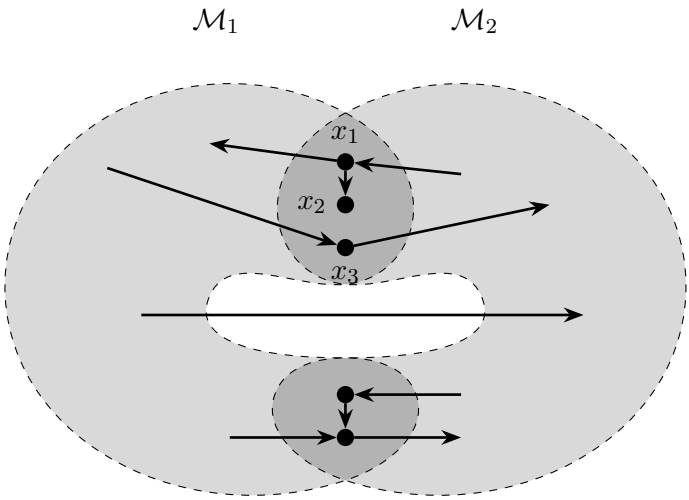


Figure 5.7: A weak directed separation with interface $\overline{X} = (x_1, x_2, x_3)$

arcs in a weak directed separation with an interface of size 3. As with directed separations, loops are always possible.

Weak separations can be transformed into proper separations by duplicating the vertices outside of the interface X. This is made precise by the following theorem.

Theorem 5.42 *Let $(\mathcal{M}_1, \mathcal{M}_2)$ be a weak directed separation of $\mathcal{M}$ with interface $\overline{X}$. Then there exists a structure $\mathcal{M}'$ and a directed separation $(\mathcal{M}'_1, \mathcal{M}'_2)$ of $\mathcal{M}'$ with the same interface $\overline{X}$ and isomorphisms $\pi_1 : \mathcal{M}_1 \to \mathcal{M}'_1$, $\pi_2 : \mathcal{M}_2 \to \mathcal{M}'_2$ which are the identity on $\overline{X}$ such that*

$$\mathrm{tp}_{L,\overline{P}}(\mathcal{M}, v, \overline{X}) = \mathrm{tp}_{L,\overline{P}}(\mathcal{M}', \pi_i(v), \overline{X})$$

for all $i \in \{1, 2\}$ and $v \in V(\mathcal{M}_i)$.

Proof. For $i \in \{1, 2\}$, define π_i and $\mathcal{M}'$ as

$$\begin{aligned} \pi_i(v) &:= \begin{cases} v & \text{if } v \in X \\ (i, v) & \text{if } v \notin X \end{cases} \\ V(\mathcal{M}') &:= \pi_1(V(\mathcal{M}_1)) \cup \pi_2(V(\mathcal{M}_2)) \\ E(\mathcal{M}') &:= E_1 \cup E_2 \cup E_3, \end{aligned}$$

where, for $i \in \{1, 2\}$,

$$\begin{aligned} E_i &:= \{(\pi_i(v), \pi_i(w)) \mid (v, w) \in E(\mathcal{M}_i)\}, \\ E_3 &:= \{(\pi_1(v), \pi_2(w)) \mid \\ &\qquad (v, w) \in E(\mathcal{M}) \cap (V(\mathcal{M}_1) \times V(\mathcal{M}_2))\}. \end{aligned}$$

The substructures $\mathcal{M}'_1$, $\mathcal{M}'_2$ of $\mathcal{M}'$ are induced by the sets

$$V(\mathcal{M}'_i) := \pi_i(V(\mathcal{M}_i)).$$

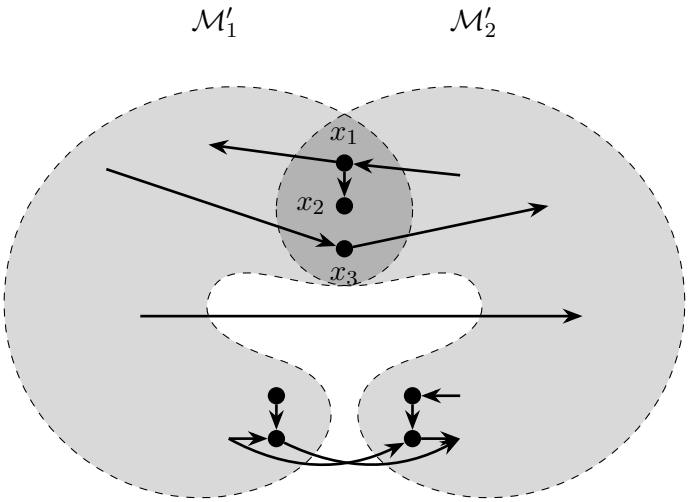


Figure 5.8: $\mathcal{M}'$ from the proof of Theorem 5.42, based on Figure 5.7

For example, applying this construction to the weak directed separation described in Figure 5.7, we get Figure 5.8 on the preceding page. We duplicate all vertices in the intersection of $\mathcal{M}_1$ and $\mathcal{M}_2$ that are not also in X, and introduce additional arcs.

Clearly, π_i is an isomorphism between $\mathcal{M}_i$ and $\mathcal{M}'_i$ and the identity on $\overline{X}$. We also have that $(\mathcal{M}'_1, \mathcal{M}'_2)$ is a directed separation of $\mathcal{M}'$.

It is easy to verify that the colored structures $\partial_{\overline{P}}(\mathcal{M}, \overline{X})$ and $\partial_{\overline{P}}(\mathcal{M}', \overline{X})$ are bisimilar. With Theorem 3.22 on page 33, bisimilarity of these structures implies

$$\mathrm{tp}_{L,\overline{P}}(\mathcal{M}, v, \overline{X}) = \mathrm{tp}_{L,\overline{P}}(\mathcal{M}', \pi_i(v), \overline{X})$$

for all $i \in \{1, 2\}$ and $v \in V(\mathcal{M}_i)$. ∎

Having isomorphisms means that

$$\mathrm{tp}_{L,\overline{P}}(\mathcal{M}_i, v, \overline{X}) = \mathrm{tp}_{L,\overline{P}}(\mathcal{M}'_i, \pi_i(v), \overline{X})$$

for all $v \in V(\mathcal{M}_i)$.

This and the previous theorem imply that Theorem 5.13 on page 114 and with appropriate wording also Theorem 5.14 hold for weak directed separations as well. Let us restate the last theorem in its more general form.

Theorem 5.43 (Corollary of theorems 5.14 and 5.42) *Let $\overline{P}$, $\overline{Q}$ be sequences of proposition symbols such that $\sigma \cap P = \sigma \cap Q = P \cap Q = \emptyset$.*

Let $L \subseteq L_\mu[\sigma \cup P]$ and $\mathcal{M}$ be a structure with a weak directed σ-separation $(\mathcal{M}_1, \mathcal{M}_2)$ with interface $\overline{X}$. Let $\overline{Y} \in ((V(\mathcal{M}_1) \setminus V(\mathcal{M}_2)) \cup X)^{|Q|}$ be a tuple.

For all $v \in \mathcal{M}_1$, the set $\operatorname{tp}_{L,\overline{Q}}(\mathcal{M}, v, \overline{Y})$ depends only on

- *$\mathcal{M}_1$ and $\overline{Q}$ and*
- *$\left\{(x_i, \operatorname{tp}_{L,\overline{P}}(\mathcal{M}_2, x_i, \overline{X})) \;\middle|\; x_i \in X\right\}$ and*
- *$\left\{(v, \operatorname{tp}_{L,\overline{P}}(\mathcal{M}_2, w, \overline{X})) \mid (v,w) \in E(\mathcal{M}) \cap (\mathcal{M}_1 \times \mathcal{M}_2)\right\}$.*

Provided L is finite, $\operatorname{tp}_{L,\overline{Q}}(\mathcal{M}, v, \overline{Y})$ can be computed from these sets.

Furthermore, for every $w \in \mathcal{M}_2$, the set $\operatorname{tp}_{L,\overline{Q}}(\mathcal{M}, w, \overline{Y})$ depends only on the above sets and on $\operatorname{tp}_{L,\overline{P}}(\mathcal{M}_2, w, \overline{X})$ and can be computed from these sets if L is finite.

The only difference of this statement to Theorem 5.14 is that we only require a weak separation and that the tuple $\overline{Y}$ should not contain a vertex $v \in V(\mathcal{M}_1) \cap V(\mathcal{M}_2)$ which is not part of the interface. This last requirement is necessary because otherwise we would have a color in $\mathcal{M}_2$ where there was none before, and the types of $\mathcal{M}_2$ with respect to $\overline{X}$ do not carry this information.

We observe that Theorem 5.39 also applies to Theorem 5.43.

5.4.2 Kelly-Width

First we consider Kelly-width. We follow the notation and definitions by Paul Hunter and Stephan Kreutzer [HK08]. For a directed acyclic graph (DAG), we write $\preceq$ for the reflexive and transitive closure of the arc relation.

$\preceq$

guard

Let G be a digraph. A set $W \subseteq V(G)$ *guards* $X \subseteq V(G)$ if $W \cap X = \emptyset$ and for all $(u,v) \in E(G)$ with $u \in X$, we have $v \in X \cup W$.

Definition 5.44 A *Kelly decomposition* of a digraph G is a triple $\mathcal{D} := (D, \beta, \gamma)$, where $\beta, \gamma : V(D) \to 2^{V(G)}$ such that Kelly decomposition

- D is a DAG and $(\beta(t))_{t \in V(D)}$ partitions $V(G)$,
- for all $t \in V(D)$, $\gamma(t)$ guards $\mathcal{B}_t^{\downarrow} := \bigcup_{t' \geq t} \beta(t')$ and $\mathcal{B}_t^{\downarrow}$
- for all $s \in V(D)$ there is a linear order $\leq_t$ on its children so that the children can be ordered as $t_1, \ldots, t_p$ such that for all $1 \leq i \leq p$,

$$\gamma(t_i) \subseteq \beta(s) \cup \gamma(s) \cup \bigcup_{j<i} \mathcal{B}_{t_j}^{\downarrow}.$$

  Similarly, there is a linear order on the roots such that

$$\gamma(r_i) \subseteq \bigcup_{j<i} \mathcal{B}_{r_j}^{\downarrow}.$$

The *width* of $\mathcal{D}$ is $\max\{|\beta(t) \cup \gamma(t)| \mid t \in V(D)\}$. The *Kelly-width* of G is the minimal width of any of its Kelly decompositions. Kelly-width ⊣

In order to distinguish $V(G)$ from $V(D)$, we call the elements of $V(D)$ *nodes*, in contrast to the vertices in $V(G)$. node

Theorem 5.45 *There exists a constant $c \in \mathbb{N}$ and an algorithm that solves the L_μ model-checking problem in time*

$$O\left(2^{2^{ck|\varphi|}|\varphi|} n^3\right),$$

where k is the Kelly-width and n the size of the input structure, provided a Kelly decomposition of width at most k is given as part of the input.

Let G be a structure of Kelly-width k with a given Kelly-decomposition of width k and let $v \in V(G)$. It is easily seen that we can construct a Kelly decomposition of G of width $\leq k+1$ which has only one root and this root contains v. We call such a Kelly decomposition *rooted at* v.

rooted at v

Proof of Theorem 5.45. Let G, v be a structure and $\overline{P}$ be a sequence of k fresh proposition symbols. We pick an arbitrary linear order of $V(G)$ in order to define interfaces consistently.

Let $\mathcal{D} = (D, \beta, \gamma)$ be a Kelly decomposition of width k of G rooted at v and $\varphi \in L_\mu$. We set $L := \{\varphi\}$.

Let us introduce the abbreviation

$$\mathcal{T}(A, B) := \left\{ (v, \mathrm{tp}_{L,\overline{P}}(A, v, B)) \mid v \in A \right\}.$$

We will inductively compute the types

$$\mathcal{T}(\mathcal{B}_t^\downarrow \cup \gamma(t), \gamma(t))$$

for all $t \in V(D)$. For the leaves, these sets can be computed by brute force. Let $t \in V(D)$ be a node with children $s_1, \ldots, s_l$ and assume that we already know the above types for all s_i.

Let

$$\delta_i := \bigcup_{j \leq i} (\gamma(s_j) \cap (\beta(t) \cup \gamma(t)))$$
$$\delta_i' := \delta_i \cup \bigcup_{j \leq i} \mathcal{B}_{s_j}^\downarrow.$$

We inductively compute the types $\mathcal{T}(\delta_i', \delta_i)$. For $i = 1$ we already know these types by assumption. Assume $i > 1$.

We want to construct weak directed separations. Note that

by assumption we know $\mathcal{T}(\mathcal{B}^{\downarrow}_{s_i} \cup \gamma(s_i), \gamma(s_i))$. We now first compute

$$\mathcal{T}(\mathcal{B}^{\downarrow}_{s_i} \cup \gamma(s_i) \cup \delta_{i-1}, \gamma(s_i) \cup \delta_{i-1}).$$

This is possible because $(\delta_{i-1}, \mathcal{B}^{\downarrow}_{s_i} \cup \gamma(s_i))$ is a directed separation with interface $\delta_{i-1} \cap \gamma(s_i)$.

Next, we observe that $(\mathcal{B}^{\downarrow}_{s_i} \cup \gamma(s_i) \cup \delta_{i-1}, \delta'_{i-1})$ is a weak directed separation with interface $\delta_{i-1} \cup (\gamma(s_i) \cap \delta_{i-1})$. Thus Theorem 5.43 allows us to compute $\mathcal{T}(\delta'_i, \delta_i)$.

After the last step we still need to compute $\mathcal{T}(\mathcal{B}^{\downarrow}_t \cup \gamma(t), \gamma(t))$ for the parent t. The pair $(\beta(t) \cup \gamma(t), \delta'_l)$ is a directed separation with interface δ_l, which is the final piece to the proof.

Let us analyze the running time. We compute $\mathcal{T}(\mathcal{B}^{\downarrow}_t \cup \gamma(t), \gamma(t))$ for each $t \in V(D)$ and we have $|V(D)| \leq n$. So if T is the time needed to compute $\mathcal{T}(\mathcal{B}^{\downarrow}_t \cup \gamma(t), \gamma(t))$, then the total running time will be $O(T \cdot n)$.

According to Lemma 5.38, computing $\mathcal{T}(\mathcal{B}^{\downarrow}_t \cup \gamma(t), \gamma(t))$ for a leaf takes time

$$O\left(2^k \left(k^2 |\varphi|^{|\varphi|}\right)^{3+|\varphi|}\right)$$

because $|\gamma(t)| \leq k$ and $|P| = k$. This term is bounded by

$$\begin{aligned} O\left(2^{k+k^2|\varphi|^{|\varphi|}(3+|\varphi|)}\right) &\subseteq O\left(2^{k+3k^2|\varphi|^{|\varphi|}+k^2|\varphi|^{|\varphi|+1}}\right) \\ &\subseteq O\left(2^{2^{k|\varphi|^{|\varphi|}}}\right). \end{aligned}$$

For a node $t \in V(D)$ with children $s_1, \ldots, s_l$, computing $\mathcal{T}(\mathcal{B}^{\downarrow}_t \cup \gamma(t), \gamma(t))$ consists of computing $\mathcal{T}(\delta'_i, \delta_i)$ for all $1 \leq i \leq l$. In each step we compute for every $v \in \delta'_i$ the

set $\mathrm{tp}_{L,\overline{P}}(\delta_i', v, \delta_i)$. Computing the latter is an application of Theorem 5.43 and by Theorem 5.39 this can be done in time

$$2^{2^{O\left(k|\varphi|^{|\varphi|}\right)}}.$$

So computing $\mathcal{T}(\delta_i', \delta_i)$ can be done in time

$$2^{2^{O\left(k|\varphi|^{|\varphi|}\right)}} \cdot n.$$

The node t can have at most n successors, so we have at most n such computations. In total, $\mathrm{tp}_{L,\overline{P}}(\mathcal{B}_t^{\downarrow} \cup \gamma(t), v, \gamma(t))$ can be computed in time

$$2^{2^{O\left(k|\varphi|^{|\varphi|}\right)}} \cdot n^2.$$

So our total running time to solve the model-checking problem on graphs of bounded Kelly-width is

$$O\left(2^{2^{ck|\varphi|^{|\varphi|}}} n^3\right)$$

for some constant $c \in \mathbb{N}$. ∎

By Theorem 4.7 on page 81, solving parity games with parameter d, where d is the number of distinct priorities, is FPT-reducible to the L_μ model-checking problem with parameter φ. Furthermore, the reduction yields a φ with $|\varphi| \in O(d)$ and a parity game whose graph is isomorphic to the structure $\mathcal{M}$.

So our FPT result for L_μ model-checking with parameter φ for structures of bounded Kelly-width also holds for parity games of bounded Kelly-width with the parameter d.

Corollary 5.46 *There exists a constant $c \in \mathbb{N}$ and an algorithm that, given a parity game $P = (V, E, V_\diamond, \omega)$ and $v \in V$, decides whether player $\diamond$ wins from (P, v) in time*

$$O\left(2^{2^{ckd^{cd}}}|V|^3\right),$$

where k is the Kelly-width and d is the number of distinct priorities of P, provided a Kelly decomposition of width at most k is given as part of the input.

5.4.3 DAG-width

Next we consider DAG-width. We use the definition and notation from [Ber+12]. Again, we write $\preceq$ for the reflexive and transitive closure of the arc relation of a DAG.

Definition 5.47 A *DAG decomposition* of a digraph G is a pair $\mathcal{D} := (D, (X_d)_{d \in D})$ such that DAG decomposition

- D is a DAG,
- $\bigcup_{d \in D} X_d = V(G)$,
- For all $d \preceq d' \preceq d''$, $X_d \cap X_{d''} \subseteq X_{d'}$,
- for all arcs $(d, d') \in E(D)$, $X_d \cap X_{d'}$ guards $X_{\succeq d'} \setminus X_d$, where

$$X_{\succeq d'} := \bigcup_{d'' \succeq d'} X_{d''}. \qquad X_{\succeq d}$$

The *width* of $\mathcal{D}$ is $\max\{|X_d| \mid d \in V(D)\}$. The *DAG-width* of G is the minimal width of any of its DAG decompositions. ⊣ DAG-width

Theorem 5.48 *There exists an algorithm that, given a structure $\mathcal{M}$ of size n, a vertex $v \in \mathcal{M}$, and a DAG-decomposition of $\mathcal{M}$ of width k and size k', answers $\mathcal{M}, v \models \varphi$ in time $O(f(k, |\varphi|) \cdot k' \cdot n)$ for some computable function f.*

Proof. Let G, v_0 be a structure and $(D, (X_d)_{d \in V(D)})$ be a nice DAG decomposition of G. That means (see [Ber+12])

1. D has a unique source.
2. Every $d \in V(D)$ has at most two successors.
3. For $d_0, d_1, d_2 \in V(D)$, if d_1, d_2 are two successors of d_0, then $X_{d_0} = X_{d_1} = X_{d_2}$.
4. For $d_0, d_1 \in V(D)$, if d_1 is the unique successor of d_0, then
$$|(X_{d_0} \setminus X_{d_1}) \cup (X_{d_1} \setminus X_{d_0})| = 1.$$

We set $L := \{\varphi\}$. As in the proof for bounded Kelly width, we fix an arbitrary linear order $<$ on $V(G)$ so that we can consistently map vertices to the proposition symbols P_i occurring in the types.

During the run of the algorithm, we fill a table $\mathcal{T}$ with indices from the set $\{(v, d) \in V(G) \times V(D) \mid v \in X_{\succeq d}\}$ and entries that are elements of $\mathcal{T}_L(\overline{P})$. We will write to every index in this table at most once during the run, and we will always make sure to write

$$\mathcal{T}(v, d) = \mathrm{tp}_{L,\overline{P}}(G[X_{\succeq d}], v, X_d).$$

If d is the root of D, then $\mathcal{T}(v_0, d)$ will answer the model-checking problem $G, v_0 \models \varphi$.

Clearly, we can fill in all values for the leaves d immediately by computing them directly.

Let $d \in V(D)$. If d has two successors d_0, d_1, then we have $X_d = X_{d_0} = X_{d_1}$. Then $(G[X_{\succeq d_0}], G[X_{\succeq d_1}])$ is a weak directed separation with interface X_d. Because we already know $\mathrm{tp}_{L,\overline{P}}(G[X_{\succeq d_i}], v, X_d)$ for all v and $i \in \{0,1\}$, Theorem 5.43 allows us to compute the types $\mathrm{tp}_{L,\overline{P}}(G[X_{\succeq d}], v, X_d)$.

The other case is that d has a unique successor d_0. Let $X_d = \{v_1, \ldots, v_k\}$ be ordered by the global linear order $<$. If $X_{d_0} \setminus X_d = \{v_i\}$, then for all $v \in X_{\succeq X_d}$ we set

$$\mathcal{T}(v,d) = \{\mathrm{shrink}_i(\psi) \mid \psi \in \mathcal{T}(v,d_0) \text{ and } P_i \text{ does not occur in } \psi\},$$

where $\mathrm{shrink}_i(\psi)$ is a function defined inductively over the structure of formulas with the base case

$$\mathrm{shrink}_i(P_j) := \begin{cases} P_j & \text{if } j < i \\ P_{j-1} & \text{if } j > i. \end{cases}$$

In other words, $\mathrm{shrink}_i(\psi)$ is the formula ψ with all P_j with $j > i$ replaced by P_{j-1} in order to not leave a hole. It is easy to check that we have

$$\{\mathrm{shrink}_i(\psi) \mid \psi \in \mathcal{T}(v,d_0) \text{ and } P_i \text{ does not occur in } \psi\} = \mathrm{tp}_{L,\overline{P}}(G[X_{\succeq d}], v, X_d).$$

The last case is $X_d \setminus X_{d_0} = \{v_i\}$. Because $X_d \cap X_{d_0}$ guards $X_{\succeq d_0} \setminus X_d$, all arcs $(v, v_i) \in G[X_{\succeq d}]$ satisfy $v \in X_d$.

This means we have a directed separation $(G[X_d], G[X_{\succeq d_0}])$ with interface X_{d_0}. We know $G[X_d]$ (its size is small), and we

know the types $\mathrm{tp}_{L,\overline{P}}(G[X_{\succeq d_0}], v, X_{d_0})$ for all $v \in X_{\succeq X_{d_0}}$.

By Theorem 5.14, this is all the information that we need to compute the type $\mathrm{tp}_{L,\overline{P}}(G[X_{\succeq d}], v, X_d)$ for all $v \in X_{\succeq X_d}$, which completes the algorithm.

For the running time we note that by a theorem by Dietmar Berwanger, Anuj Dawar, Paul Hunter, Stephan Kreutzer and Jan Obdržálek, a nice DAG-decomposition has width k and size at most $4kk'$ if we start with a DAG-decomposition of width k and size k' [Ber+12, Theorem 24]. Furthermore, a nice DAG-decomposition can be computed in time $O(kk')$.

We need to compute at most $|V(G)||V(D)| = n \cdot 4kk'$ many types, and computing one type $\mathcal{T}(v, d)$ takes time $O(g(k, |\varphi|))$ for some computable function g. Computing one type is independent of n because the sizes of all the sets that Theorem 5.14 needs are independent of n.

In total we get a running time of

$$O(kk' + g(k, |\varphi|) \cdot n \cdot 4kk') \subseteq O(f(k, |\varphi|) \cdot k' \cdot n)$$

for some computable function f, as claimed. ∎

One might expect that this upper bound could be simplified to $O(f(k, |\varphi|) \cdot n^c)$ for some constant c, removing the odd dependency on the size of the DAG-decomposition.

However, Saeed Amiri, Stephan Kreutzer and Roman Rabinovich showed recently that DAG-decompositions of a graph G are sometimes necessarily super-polynomial in the size of G [AKR14]. So in general, k' can be super-polynomial in n, which prevents the proposed simplification.

Analogous to Corollary 5.46, by Theorem 4.7 we get the same result for parity games.

Corollary 5.49 *There exists an algorithm that, given a parity game $P = (V, E, V_\diamond, \omega)$ with d distinct priorities, a vertex $v \in P$, and a DAG-decomposition of (V, E) of width k and size k', decides whether player $\diamond$ wins from (P, v) in time $O(f(k, d) \cdot k' \cdot |V|)$ for some computable function f.*

Note that this does not follow from the result by Dietmar Berwanger, Anuj Dawar, Paul Hunter, and Stephan Kreutzer that parity games on classes of bounded DAG-width can be solved in polynomial time because their result is not an FPT result [Ber+06].

5.5 Conclusions

We presented a decomposition theorem for the modal μ-calculus, introducing a notion of L_μ-types in the process. This theorem, interesting already all by itself, further allowed us to prove fixed-parameter tractability results for the L_μ model-checking problem on classes of bounded Kelly-width or of bounded DAG-width.

Open questions arise from the diverse number of decompositions for directed graphs. In particular, we think it could be promising to analyze D-width, introduced by Mohammad Ali Safari [Saf05]. Another notable width measure is directed tree-width, introduced by Thor Johnson, Neil Robertson, Paul D. Seymour, and Robin Thomas [Joh+01].

6 Graph Operations on Parity Games

As discussed in Chapter 4, the computational complexity of solving parity games remains a major open question. It is known that solving parity games is in UP ∩ coUP, and there also exists a sub-exponential algorithm [JPZ06].

One way of tackling this problem is to consider the problem on restricted classes of parity games. For some classes, polynomial-time algorithms are known, for example on classes of bounded tree-width [Obd03; FS12; Gan15], bounded entanglement [BG04], bounded DAG-width [Ber+06], and bounded clique-width [Obd07]. See Section 4.1.2 on page 77 for a discussion of these algorithms. In Chapter 5, we showed that solving parity games is fixed-parameter tractable on classes of bounded Kelly-width and classes of bounded DAG-width, provided that a decomposition is given as part of the input.

In this chapter we consider graph classes that are constructed from simpler graph classes. The idea is that if we can solve parity games in polynomial time on a given class $\mathcal{C}$, can we then also solve it on classes constructed from $\mathcal{C}$? The answer turns out to be yes, for certain constructions.

In particular, using these methods, we provide polynomial time algorithms for solving parity games whose underlying graph is an orientation of a complete graph (such as tourna-

ments), a complete bipartite graph, a block graph, or a cactus graph. A block graph is a graph where every biconnected component is a clique and a cactus graph is a graph where every edge lies on at most one cycle (see e.g., [BLS99]).

The results of this chapter have been published in [DKT16].

6.1 Preliminaries

6.1.1 Basic Definitions

First we need a few basic definitions. For these, we will follow the book by Jørgen Bang-Jensen and Gregory Gutin [BJG09]. Because we consider only algorithms, all graphs in this chapter are finite.

orientation

Definition 6.1 An *orientation* of an undirected graph $G = (V, E)$ is a digraph obtained from G by replacing every edge $\{x, y\} \in E$ by one of the arcs (x, y) or (y, x), but not both. ⊣

biorientation

Definition 6.2 A *biorientation* of G is a digraph obtained from G by replacing every edge $\{x, y\} \in E$ with the arcs (x, y) or (y, x) or both. ⊣

See Figure 6.1 for an illustration of these concepts.

We recall the definition of parity games and the associated notions from Section 3.2 on page 47. We will also use attractor sets quite often in this chapter. We defined attractor sets in Definition 3.58 as follows: $\mathrm{attr}_i(A)$ is the set of vertices in V from which player i has a strategy to enter A at least once.

For a parity game $P = (V, E, V_\diamond, \omega)$ and some priority $d \in \mathbb{N}$, we denote by $\omega^{-1}(d)$ the set of vertices having priority d.

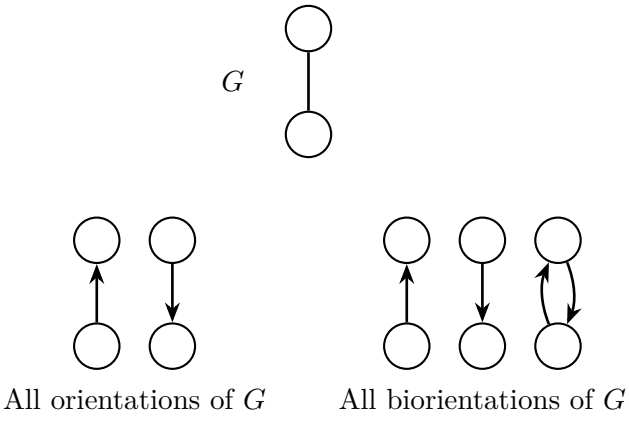


Figure 6.1: Orientations and biorientations

Definition 6.3 Given a parity game $P = (V, E, V_\Diamond, \omega)$ and a set $A \subseteq V$, we denote by $P \cap A$ the *restriction of P to A*: restriction

$$P \cap A := \big(V \cap A, E \cap (A \times A), V_\Diamond \cap A, \omega{\restriction_A}\big).$$

We write $P \setminus A$ as an abbreviation for the game $P \cap (V \setminus A)$.⊣

Definition 6.4 Given a class of parity games $\mathcal{C}$, we say that $\mathcal{C}$ is *hereditary* if for all $P \in \mathcal{C}$ and $A \subseteq V(P)$, we have $P \cap A \in \mathcal{C}$.⊣ hereditary

Definition 6.5 Given parity games P and P', we call P' a *proper subgame of P* if $P' = P \cap V(P')$ and $P' \neq P$. ⊣ proper subgame

Definition 6.6 A *single-player game* is a parity game where all vertices belong to one of the players, that is, $V = V_\Diamond$ or $V = V_\Box$. If all the vertices of a single-player game P belong to player i, we will say that P *belongs to* player i. ⊣ single-player game

We will use an algorithm Solve-Single-Player-Game(P) which solves single-player games in time cubic in the number of vertices, by returning the two winning regions $(W_\Diamond(P), W_\Box(P))$. A description of such an algorithm is given, for example, by Oliver Friedmann and Martin Lange [FL09]. We also gave an explicit construction in the proof of Theorem 3.51 on page 51.

6.1.2 Half-Solving Parity Games

First, let us simplify the problem of finding an algorithm for solving parity games to the problem of *half-solving* them.

half-solve

Definition 6.7 (Half-Solving Parity Games) Let $\mathcal{C}$ be a class of parity games. An algorithm *half-solves* $\mathcal{C}$ if for every input game $P \in \mathcal{C}$ it returns one of the following three results.

1. $W_\Diamond(P)$ and $W_\Box(P)$.
2. A proper subgame P' of P and sets $W^*_\Diamond, W^*_\Box \subseteq V(P) \setminus V(P')$ such that $W_\Diamond(P) = W^*_\Diamond \cup W_\Diamond(P')$ and $W_\Box(P) = W^*_\Box \cup W_\Box(P')$.
3. The game P itself, but only if either $W_\Diamond(P) = \emptyset$ or $W_\Box(P) = \emptyset$. ⊣

Intuitively, an algorithm half-solves a parity game by either solving it on the spot, by reducing the problem to a proper subgame, or by determining that one of the winning regions is empty (but without telling us which one is empty).

We claim that efficiently half-solving a hereditary class $\mathcal{C}$ suffices to efficiently solve $\mathcal{C}$ completely. This is the statement of the following lemma, whose proof is based on the recursive algorithm by Robert McNaughton [McN93].

Lemma 6.8 *Let $\mathcal{C}$ be a hereditary class of parity games. If there is an algorithm half-solving parity games in $\mathcal{C}$ running in time $O(n^c)$ for some $c \geq 1$, where n is the number of vertices, then there is an algorithm that computes the winning regions on all parity games in $\mathcal{C}$ in time $O(n^{c+1})$.*

Proof. Let Half-Solve be the given algorithm. Consider Algorithm 6.1.

Algorithm 6.1: From partial to full solution.

```
Solve(P = (V, E, V_◇, ω))
    R ← Half-Solve(P)
    if R = (W_◇, W_□) then
        return (W_◇, W_□)
    if R = (P', W*_◇, W*_□) then
        (W_◇(P'), W_□(P')) ← Solve(P')
        return (W*_◇ ∪ W_◇(P'), W*_□ ∪ W_□(P'))
    if R = P then
        d ← Minimum-Priority(ω)
        i ← ◇ if d is even, □ otherwise
        (C_◇, C_□) ← Solve(P \ attr_i(ω^{-1}(d)))
        if C_ī ≠ ∅ then
            (W_i, W_ī) ← (∅, V)
        else
            (W_i, W_ī) ← (V, ∅)
        return (W_◇, W_□)
```

First let us analyze the running time of Solve. Let $T(n)$ be the running time of Solve and kn^c be the running time of

Half-Solve with $k \in \mathbb{N}$. Then we have

$$T(n) \leq O(n) + kn^c + f(n) + T(n-1)$$

where $f(n)$ is the time used to compute the attractor set $\mathrm{attr}_i(\omega^{-1}(d))$.

An attractor set can be computed in time $O(n^2)$, but this would only give the bound $T(n) \in O(n^{c+2})$. However, we observe that in total each arc is only considered once because $P \setminus \mathrm{attr}_i(\omega^{-1}(d))$ does not contain any arcs with their start or end point in $\mathrm{attr}_i(\omega^{-1}(d))$. So the algorithm only needs $O(n^2)$ time to compute all the attractor sets in total over all recursive steps. So we have $T(n) \in O(n^{c+1} + n^2)$ which gives the desired bound of $T(n) \in O(n^{c+1})$.

To prove the correctness, it is enough to consider the case where Half-Solve returns P unmodified because the other cases are trivially correct.

Let d be the minimum priority of P and let $D := \omega^{-1}(d)$. Assume that d is good for player $\Diamond$ (the case of player $\Box$ is similar). Then the algorithm solves the game $P' := P \setminus \mathrm{attr}_\Diamond(D)$.

If $W_\Box(P') \neq \emptyset$, then $W_\Box(P) \neq \emptyset$ because every vertex winning for player $\Box$ in P' is also winning for player $\Box$ in P, because we only removed a $\Diamond$-attractor (Lemma 3.65 on page 57). Since we are in the case where one of the winning regions of P is empty, this implies $W_\Diamond(P) = \emptyset$.

If $W_\Box(P') = \emptyset$, then it is easy to see that player $\Diamond$ wins everywhere in P. Indeed, if a play eventually stays in P', then player $\Diamond$ wins because $W_\Box(P') = \emptyset$. If a play visits $\mathrm{attr}_\Diamond(D)$ an infinite number of times, then player $\Diamond$ can force to visit D an infinite number of times, and hence wins the play. ∎

6.1.3 Recognizing Winning Regions

Lemma 6.8 shows a method to decide the global winner under the restriction that there is one definite global winner (that is, $W_\diamond = \emptyset$ or $W_\square = \emptyset$) and under the assumption that we can solve subgames recursively. A natural question is if this can be generalized to recognize winning regions without the restriction that $W_\diamond = \emptyset$ or $W_\square = \emptyset$. Unfortunately, the answer is no, as shown by Dietmar Berwanger, Krishnendu Chatterjee, Martin De Wulf, Laurent Doyen and Thomas A. Henzinger.

Proposition 6.9 ([Ber+10, Proposition 2])
The following two problems are polynomial-time equivalent.

1. *Given a parity game P and $v \in V(P)$, decide whether player $\diamond$ wins from (P, v).*
2. *Given a parity game P and $W \subseteq V(P)$, decide whether $W = W_\diamond$.*

The direction from (2) to (1) says that recognizing a winning region as correct is as hard as solving parity games.

6.2 The Join of Two Parity Games

In this section we will show that if $\mathcal{C}$ and $\mathcal{C}'$ are hereditary classes of parity games that can be solved in polynomial time, then the class of parity games obtained by *joining* games from $\mathcal{C}$ and $\mathcal{C}'$ can also be solved in polynomial time.

Definition 6.10 (Join of parity games) Given two parity games $P' = (V', E', V'_\diamond, \omega')$ and $P'' = (V'', E'', V''_\diamond, \omega'')$ with $V' \cap V'' = \emptyset$, we say that the game $P = (V, E, V_\diamond, \omega)$ is a *join of P' and P''* if the following conditions hold.

- $V = V' \cup V''$,
- $E = E' \cup E'' \cup E^*$, where for all $x \in V'_\Box$, $y \in V''_\Diamond$ and for all $x \in V'_\Diamond$, $y \in V''_\Box$, the set $E^* \subseteq (V' \times V'') \cup (V'' \times V')$ contains at least one arc (x, y) or (y, x),
- $V_\Diamond = V'_\Diamond \cup V''_\Diamond$,
- the vertices of P have the same priorities as they have in P' and P''.

Given two classes of parity games $\mathcal{C}$ and $\mathcal{C}'$, we define

HalfJoin
$$\mathsf{HalfJoin}(\mathcal{C}) := \{P \mid P \text{ is a join of a single-player game } P' \text{ and a game } P'' \in \mathcal{C}\},$$

Join
$$\mathsf{Join}(\mathcal{C}, \mathcal{C}') := \{P \mid P \text{ is a join of } P' \in \mathcal{C} \text{ and } P'' \in \mathcal{C}'\}.$$

⊣

Remark 6.11 Observe that if $\mathcal{C}$ and $\mathcal{C}'$ are hereditary, then so are $\mathsf{HalfJoin}(\mathcal{C})$ and $\mathsf{Join}(\mathcal{C}, \mathcal{C}')$.

We will first show how to solve parity games obtained by joining a polynomial time solvable parity game with a single-player game and then extend this construction to the general case of joining arbitrary parity games.

As an immediate corollary we get that parity games whose underlying undirected graph is a complete graph, so-called *tournaments*, can be solved in polynomial time. As a corollary from the more general case of arbitrary joins we get that

parity games whose underlying undirected graph is a complete bipartite graph can be solved in polynomial time.

Note that the result for tournaments is not a special case of Obdržálek's polynomial time algorithm for parity games of bounded directed clique-width [Obd07]. Biorientations of complete graphs and of complete bipartite graphs do not have bounded directed clique-width, although their underlying undirected graphs have bounded undirected clique-width.

6.2.1 Adjoining Vertices Belonging to One Player

Let us now show how to solve half-joins efficiently. We will use the same technique also in the proof of the main theorem on joins in Section 6.2.2.

Lemma 6.12 *If $\mathcal{C}$ is a hereditary class of parity games that can be solved in time $O(n^c)$, then all games $P \in \mathsf{HalfJoin}(\mathcal{C})$ can be solved in time $O\big(n^{1+\max\{3,c\}}\big)$, provided that a decomposition of P as a join of a game in $\mathcal{C}$ with a single-player parity game is given.*

Proof. We claim that Algorithm 6.2 half-solves $\mathsf{HalfJoin}(\mathcal{C})$ in time $O\big(n^{\max\{3,c\}}\big)$. By Lemma 6.8 we then have an algorithm for solving $\mathsf{HalfJoin}(\mathcal{C})$ in time $O\big(n^{1+\max\{3,c\}}\big)$.

In order to prove the correctness, let $P = (V, E, V_\diamond, \omega) \in \mathsf{HalfJoin}(\mathcal{C})$ be a join of the single-player game $P' = (V', E', V'_\diamond, \omega')$ and the game $P'' = (V'', E'', V''_\diamond, \omega'') \in \mathcal{C}$. We may assume without loss of generality that $V'_\diamond$ is empty, so that $V' = V'_\Box$ and $V_\diamond = V''_\diamond$.

We assume that there is an algorithm Solve-$\mathcal{C}$-Game that solves games in $\mathcal{C}$ in time $O(n^c)$ where n is the number of

Algorithm 6.2: A polynomial-time algorithm for half-solving parity games on half joins where P is a join of the $\Diamond$- or $\Box$-player game P' and the game $P'' \in \mathcal{C}$.

HALF-SOLVE-HALF-JOIN($\boldsymbol{P = (V, E, V_\Diamond, \omega)}$, $\boldsymbol{P'}$, $\boldsymbol{P''}$)

- **if** $V = \emptyset$ **then**
  - **return** $(\emptyset, \emptyset)$
- $i \leftarrow \Diamond$ **if** $V_\Diamond(P') = \emptyset$, $\Box$ **otherwise**
- $(A_i,\, A_{\bar{i}}) \leftarrow$ SOLVE-SINGLE-PLAYER-GAME$(P \setminus \mathrm{attr}_i(V_i(P'')))$
- **if** $A_{\bar{i}} \neq \emptyset$ **then**
  - $W_i^* \leftarrow \emptyset$
  - $W_{\bar{i}}^* \leftarrow \mathrm{attr}_{\bar{i}}(A_{\bar{i}})$
  - **return** $(P \setminus W_{\bar{i}}^*, W_\Diamond^*, W_\Box^*)$
- $(B_i,\, B_{\bar{i}}) \leftarrow$ SOLVE-$\mathcal{C}$-GAME$(P \setminus \mathrm{attr}_{\bar{i}}(V_{\bar{i}}(P')))$
- **if** $B_i \neq \emptyset$ **then**
  - $W_i^* \leftarrow \mathrm{attr}_i(B_i)$
  - $W_{\bar{i}}^* \leftarrow \emptyset$
  - **return** $(P \setminus W_i^*, W_\Diamond^*, W_\Box^*)$
- **return** P

vertices of the game. Recall from Section 6.1.1 that single-player parity games can be solved in cubic time by the algorithm Solve-Single-Player-Game(P).

The algorithm first solves the single-player game $P_1 := P \setminus \mathrm{attr}_\Diamond(V''_\Diamond)$, in time $O(n^3)$. See Figure 6.2 for an illustration of the situation, where vertices lie above the dotted line if and only if they are winning for player $\Box$. By Lemma 3.65 we know that $W_\Box(P_1) \subseteq W_\Box(P)$ and by Lemma 3.66 we get that

$$W_\Box(P) = \mathrm{attr}_\Box(W_\Box(P_1)) \cup W_\Box(P \setminus \mathrm{attr}_\Box(W_\Box(P_1))).$$

Therefore, if $W_\Box(P_1)$ is not empty, then we can return

$$(P \setminus \mathrm{attr}_\Box(W_\Box(P_1)), \emptyset, \mathrm{attr}_\Box(W_\Box(P_1))),$$

which is the second outcome of the algorithm according to Definition 6.7.

If $W_\Box(P_1)$ is empty, then we solve the game $P_2 := P \setminus \mathrm{attr}_\Box(V'_\Box)$ in time $O(n^c)$ and proceed as before (see Figure 6.3). By Lemma 3.65 we know that $W_\Diamond(P_2) \subseteq W_\Diamond(P)$. Thus, if $W_\Diamond(P_2)$ is not empty, then we can return

$$(P \setminus \mathrm{attr}_\Diamond(W_\Diamond(P_2)), \mathrm{attr}_\Diamond(W_\Diamond(P_2)), \emptyset),$$

which again is the second possible outcome according to Definition 6.7.

Finally, suppose both $W_\Box(P_1)$ and $W_\Diamond(P_2)$ are empty. We claim that either $W_\Box(P) = \emptyset$ or $W_\Diamond(P) = \emptyset$. In both cases we return P unmodified as this is the third possible outcome according to Definition 6.7. To establish the claim we distinguish two cases, depending on whether $V'_\Box \cap W_\Diamond(P) = \emptyset$ or not.

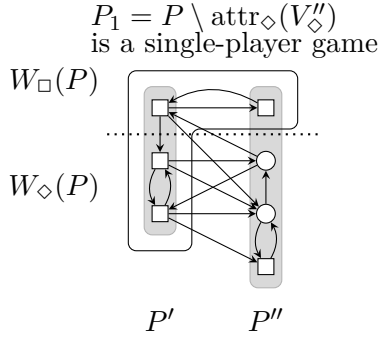


Figure 6.2: The first subgame solved by Algorithm 6.2, where P is a join of the single-player parity game P' and the game $P'' \in \mathcal{C}$.

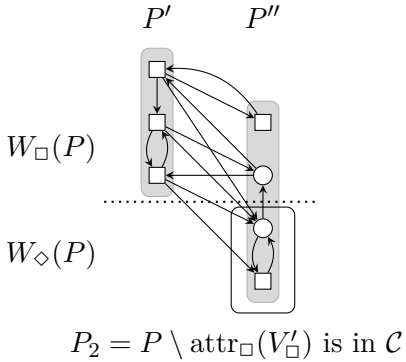


Figure 6.3: The second subgame solved by Algorithm 6.2

Case 1. $V'_\Box \cap W_\Diamond(P) \neq \emptyset$ (see Figure 6.4a).

Observe that

$$V''_\Diamond \subseteq W_\Diamond(P).$$

Assume to the contrary that there is $v \in V''_\Diamond \cap W_\Box(P)$ and let $w \in V'_\Box \cap W_\Diamond(P)$. But then there can be no arc (v, w) in P because if there were such an arc, then player $\Diamond$ would have a winning strategy from v by choosing w as v's successor. Similarly, there cannot be an arc (w, v). This contradicts the definition of P. By Lemma 3.66, we have that

$$W_\Box(P) = W_\Box(P \setminus \mathrm{attr}_\Diamond(V''_\Diamond)) = W_\Box(P_1).$$

As $W_\Box(P_1) = \emptyset$, this implies that $W_\Diamond(P) = V$.

Case 2. $V'_\Box \cap W_\Diamond(P) = \emptyset$, equivalently $V'_\Box \subseteq W_\Box(P)$ (see Figure 6.4b). Again by Lemma 3.66, we have that

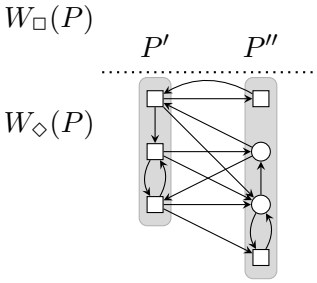
$$W_\Diamond(P) = W_\Diamond(P \setminus \mathrm{attr}_\Box(V'_\Box)) = W_\Diamond(P_2).$$

As $W_\Diamond(P_2) = \emptyset$, this implies that $W_\Box(P) = V$.

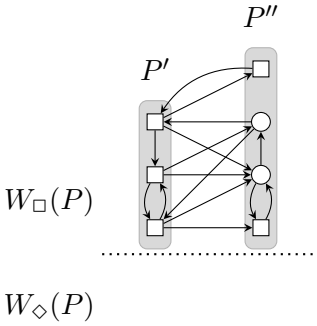
We see that the running time of the algorithm is $O(n^3 + n^c)$ because solving the single-player game and the game in $\mathcal{C}$ are the most expensive operations. As claimed, this is in $O(n^{\max\{3,c\}})$. ∎

Definition 6.13 We say that a digraph $D = (V, E)$, with a partition of its vertices $V = V_\Diamond \cup V_\Box$, is a *weak tournament* if between every two vertices $v \in V_\Diamond$, $w \in V_\Box$ we have that $(v, w) \in E$ or $(w, v) \in E$ (or both).

weak tournament



(a) Case 1



(b) Case 2

Figure 6.4: The case distinction of the proof of Lemma 6.12, where P is a join of the single-player parity game P' and the game $P'' \in \mathcal{C}$.

We denote by wTournaments the class of parity games on a weak tournament, that is,

$$\begin{aligned}\mathsf{wTournaments} := \{P = (V, E, V_\Diamond, \omega) \mid \\ (V, E) \text{ with the partition } V = V_\Diamond \cup V_\Box \\ \text{is a weak tournament}\}.\end{aligned}$$ ⊣

Note that every *tournament* (that is, an orientation of a complete graph) is a weak tournament.

An easy corollary of Lemma 6.12 is the following.

Corollary 6.14 *There is an algorithm that solves all parity games* $P = (V, E, V_\Diamond, \omega) \in \mathsf{wTournaments}$ *and runs in time* $O(|V|^4)$.

Proof. Let $\mathcal{C}$ be the class of single-player games. Then we have $\mathsf{HalfJoin}(\mathcal{C}) = \mathsf{wTournaments}$. The required decompositions can be easily computed. ■

For $i \in \{\Diamond, \Box\}$, we define $\mathsf{HalfJoin}_i(\mathcal{C})$ to be the class of joins of single-player games that belong to player i with games from $\mathcal{C}$. Note that $\mathsf{HalfJoin}(\mathcal{C}) = \mathsf{HalfJoin}_\Diamond(\mathcal{C}) \cup \mathsf{HalfJoin}_\Box(\mathcal{C})$. We call a class $\mathcal{C}$ of parity games *polynomial-time decidable* if there is a polynomial-time algorithm that decides $\mathcal{C}$, that is, tests membership $P \in \mathcal{C}$.

$\mathsf{HalfJoin}_i(\mathcal{C})$

poly-time decidable

We see that in some cases we do not need to provide a decomposition as required by Lemma 6.12.

Corollary 6.15 *Let* $\mathcal{C}$ *be a hereditary class of parity games and* $i \in \{\Diamond, \Box\}$.

1. *If* $\mathcal{C}$ *is polynomial-time solvable, then* $\mathsf{HalfJoin}_i(\mathcal{C})$ *is polynomial-time solvable.*

2. *If $\mathcal{C}$ is polynomial-time solvable and polynomial-time decidable, then* $\mathsf{HalfJoin}(\mathcal{C})$ *is polynomial-time solvable.*

Proof. Given $P = (V, E, V_\Diamond, \omega) \in \mathsf{HalfJoin}_i(\mathcal{C})$, we can compute a decomposition of it in time linear in the number of arcs. Indeed, we can take

- $P_i' := P \cap \{v \in V_i \mid N(v) \supseteq V_{\bar{i}}\}$ as the single-player game,
- $P_i'' := P \setminus \{v \in V_i \mid N(v) \supseteq V_{\bar{i}}\}$ as the game from the hereditary class $\mathcal{C}$.

This proves the first statement.

If the player to which the vertices of the single-player game belong to is not known, then we have to check for both $i \in \{\Diamond, \Box\}$ which game P_i'' considered above belongs to $\mathcal{C}$ in order to find a decomposition. This proves the second statement. ∎

Note that if we could also compute the winning strategies for the graphs in $\mathsf{HalfJoin}_i(\mathcal{C})$, then we could solve games in $\mathsf{HalfJoin}(\mathcal{C})$ without having to test membership to $\mathcal{C}$. Indeed, one could construct a game which assumes that player $\Diamond$ owns the single player game, and a game which assumes that player $\Box$ owns the single player game, run the algorithm from Lemma 6.12 on these two instances in parallel, and then check the validity of the returned winning strategies. However, computing the winning strategies for games in $\mathsf{HalfJoin}_i(\mathcal{C})$ is not trivial, as in our current proof we reduce the game to one in which we are only guaranteed that one player wins from all vertices. Moreover, since the $\mathsf{HalfJoin}$ operation is not closed under arc deletions, we cannot pass from winning regions to winning strategies by the simple policy of removing arcs one

at a time and checking in which resulting game the winning regions change.

Even though not needed in the next section, we introduce here another kind of adjoining operation between a single-player parity game P' and an arbitrary parity game P''. In this case, assuming the vertices of P' belong to player $\bar{i}$, we fix a subset M of vertices of player i of P'' and connect every vertex of P' with every vertex in M. These are the only arcs that may exist between P' and P''. So if $M = \emptyset$, we have the disjoint union of P' and P''.

Definition 6.16 Given $i \in \{\Box, \Diamond\}$, a single-player game $P' = (V', E', V'_\Diamond, \omega')$ with $V'_i = \emptyset$ and a parity game $P'' = (V'', E'', V''_\Diamond, \omega'')$ with $V' \cap V'' = \emptyset$, we say that a game $P = (V, E, V_\Diamond, \omega)$ is a *generalized single-player join (G-join) of P' and P''* if P is the join of P' and P'' except that the set of arcs is defined differently: There exists a set $M \subseteq V''_i$ such that $E = E' \cup E'' \cup E^*$ and $E^* \subseteq (M \times V'_{\bar{i}}) \cup (V'_{\bar{i}} \times M)$ and there is at least one arc (x, y) or (y, x) for all $x \in M$, $y \in V'_{\bar{i}}$.

For a class $\mathcal{C}$ of parity games, we denote by $\mathsf{HalfJoin}_G(\mathcal{C})$ the class

$$\mathsf{HalfJoin}_G(\mathcal{C}) := \qquad \mathsf{HalfJoin}_G$$
$$\{P \mid P \text{ is a G-join of a single-player game } P' \text{ and a game } P'' \in \mathcal{C}\}. \qquad \dashv$$

Theorem 6.17 *If $\mathcal{C}$ is a hereditary class of parity games that can be solved in polynomial time, then all games $P \in \mathsf{HalfJoin}_G(\mathcal{C})$ can be solved in polynomial time, provided that a decomposition of P as a G-join of a single-player game and a game in $\mathcal{C}$ is given.*

Proof. The algorithm and proof are very similar to what we saw in Lemma 6.12, the only difference being the subgame the algorithm solves in the first case. We briefly sketch the difference here, under the same notations and assumptions as before. Let M be as given in the definition of the G-join.

In the original algorithm, Algorithm 6.2, the first subgame considered by the algorithm is $P \setminus \mathrm{attr}_i(V_i(P''))$, which is then solved in polynomial time because it is a single-player game. The difference for $\mathsf{HalfJoin}_G(\mathcal{C})$ is that for the first subgame we instead consider $P_1 := P \setminus \mathrm{attr}_i(M)$. Now P_1 will not be single-player game in general. However, P_1 is the disjoint union of the single player game $P_1 \cap P'$ and the game $P_1 \cap P'' \in \mathcal{C}$, so we can solve P_1 in polynomial time.

What remains is to prove that this algorithm is correct. The proof proceeds along the lines of Lemma 6.12, the only difference being in Case 1 because the first subgame is different.

If the set $V'_\Box \cap W_\Diamond(P)$ is not empty, then let v be an arbitrary vertex in this set. Note that $M = N(v) \cap V''_\Diamond$. By the same argument given in the proof of Lemma 6.12, we get $M \subseteq W_\Diamond(P)$. Therefore, $W_\Box(P) = W_\Box(P_1)$, where $P_1 := P \setminus \mathrm{attr}_\Diamond(M)$, which is what we needed to show.

The complementary case $V'_\Box \cap W_\Diamond(P) = \emptyset$ is identical to Case 2 in the proof of Lemma 6.12, and the result immediately follows. ∎

6.2.2 Joining Two Parity Games

Now we can state our main theorem on the join of two parity games.

Theorem 6.18 *If $\mathcal{C}$ and $\mathcal{C}'$ are hereditary classes of parity games that we can solve in time $O(n^c)$ and $O(n^{c'})$, respectively,*

then there exists an algorithm for solving $\mathsf{Join}(\mathcal{C}, \mathcal{C}')$ *in time* $O\big(n^{2+\max\{3,c,c'\}}\big)$*, assuming a decomposition as a join of a game in* $\mathcal{C}$ *and a game in* $\mathcal{C}'$ *is given.*

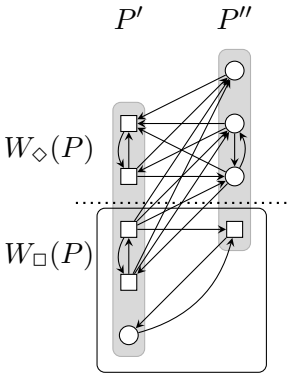
Proof. Let $P = (V, E, V_\diamond, \omega) \in \mathsf{Join}(\mathcal{C}, \mathcal{C}')$ be a join of a game $P' = (V', E', V'_\diamond, \omega') \in \mathcal{C}$ with a game $P'' = (V'', E'', V''_\diamond, \omega'') \in \mathcal{C}'$.

We follow the lines of the proof of Lemma 6.12. The case distinctions and their conclusions are exactly the same. The only difference is that in the proof of Lemma 6.12, the two subgames $P \setminus \operatorname{attr}_\diamond(V''_\diamond)$ and $P \setminus \operatorname{attr}_\square(V'_\square)$ were solvable in polynomial time because they were single-player games or in $\mathcal{C}$, respectively. Here these games can be solved in time $O\big(n^{1+\max\{3,c,c'\}}\big)$ by Lemma 6.12 because both games are in $\mathsf{HalfJoin}(\mathcal{C})$ and $\mathsf{HalfJoin}(\mathcal{C}')$, respectively. See Figure 6.5 and note the similarity to figs. 6.2 and 6.3. Note that in contrast to the other figures, in Figure 6.5a the winning region for player $\diamond$ is at the top.

The remaining part of the algorithm is exactly the same as in the proof of Lemma 6.12: If both of these subgames do not provide a usable winning region, then we have $W_\square(P) = \emptyset$ or $W_\diamond(P) = \emptyset$ and we return the original game unmodified.

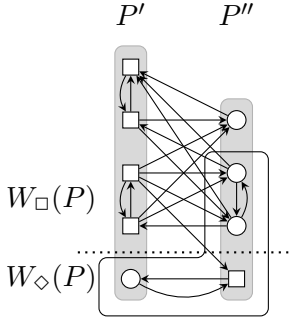
In total this gives a running time of $O\big(n^{1+\max\{3,c,c'\}}\big)$ for the half-solving algorithm. Lemma 6.8 turns this into an algorithm that solves all parity games in $\mathsf{Join}(\mathcal{C}, \mathcal{C}')$ in time $O\big(n^{2+\max\{3,c,c'\}}\big)$. ∎

We remark that if $\mathcal{C}$ is the class of parity games without arcs, then the class $\mathsf{Join}(\mathcal{C}, \mathcal{C})$ contains the class of parity games whose underlying graph is a biorientation of a complete bipartite graph.



$P \setminus \mathrm{attr}_\diamond(V''_\diamond) \in \mathsf{HalfJoin}(\mathcal{C})$

(a) Case 1a



$P \setminus \mathrm{attr}_\square(V'_\square) \in \mathsf{HalfJoin}(\mathcal{C}')$

(b) Case 2a

Figure 6.5: An illustration of the proof of Theorem 6.18, where P is a join of $P' \in \mathcal{C}$ and $P'' \in \mathcal{C}'$.

6.3 Pasting of Parity Games

Definition 6.19 Let $P' = (V', E', V'_\diamond, \omega')$ and $P'' = (V'', E'', V''_\diamond, \omega'')$ be two parity games with $V' \cap V'' = \emptyset$, and let $v' \in V'$ and $v'' \in V''$. Assume that v', v'' have the same priority and belong to the same player in P' and in P'', respectively, say player i.

The result of *pasting* P', P'' *at* v', v'' is the game $P = (V, E, V_\diamond, \omega)$, defined as the disjoint copy of P' and P'' with v' and v'' identified (see Figure 6.6 on the following page). Formally, we define this game as follows. pasting

$$
\begin{aligned}
V &:= (V' \cup V'' \cup \{v^*\}) \setminus \{v', v''\} \\
E &:= \{(u, w) \in E' \cup E'' \mid \{v', v''\} \cap \{u, w\} = \emptyset\} \\
&\quad \cup \{(u, v^*) \mid (u, v') \in E' \text{ or } (u, v'') \in E''\} \\
&\quad \cup \{(v^*, u) \mid (v', u) \in E' \text{ or } (v'', u) \in E''\} \\
V_\diamond &:= \begin{cases} (V'_\diamond \cup V''_\diamond \cup \{v^*\}) \setminus \{v', v''\} & \text{if } i = \diamond \\ (V'_\diamond \cup V''_\diamond) \setminus \{v', v''\} & \text{otherwise} \end{cases} \\
\omega(v) &:= \begin{cases} \omega'(v) & \text{if } v \in V' \\ \omega''(v) & \text{if } v \in V'' \\ \omega'(v') & \text{if } v = v^*. \end{cases}
\end{aligned}
$$

For a class of parity games $\mathcal{C}$, we denote by $\mathsf{RepeatedPasting}(\mathcal{C})$ the class of games obtained by repeated pasting of a finite number of games from $\mathcal{C}$. ⊣ RepeatedPasting

We observe that if $\mathcal{C}$ is hereditary and is closed under disjoint unions, then $\mathsf{RepeatedPasting}(\mathcal{C})$ is hereditary. Moreover, every $P \in \mathsf{RepeatedPasting}(\mathcal{C})$ has a decomposition into components

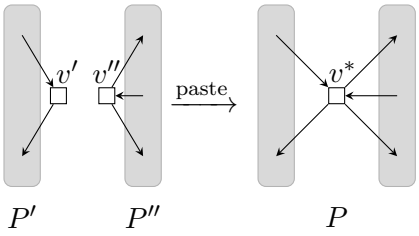


Figure 6.6: The paste of P' and P'' at v', v''

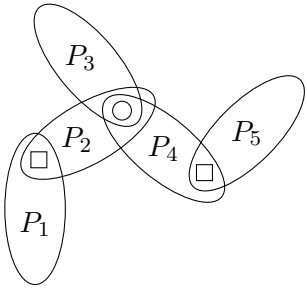


Figure 6.7: A game from $\mathsf{RepeatedPasting}(\mathcal{C})$

$P_1, P_2, \ldots, P_k$ with $P_i \in \mathcal{C}$, such that every distinct P_i and P_j are either disjoint or share exactly one vertex. This means that games $P_1, \ldots, P_k$ form a tree-like structure, in the sense that the graph T_P obtained by adding a vertex i for every P_i and an arc (i, j) if P_i and P_j share a vertex is a tree (see Figure 6.7 on the preceding page).

Theorem 6.20 *If $\mathcal{C}$ is a hereditary class of parity games that can be solved in time $O(n^c)$, then games in* $\mathsf{RepeatedPasting}(\mathcal{C})$ *can be solved in time $O\big(n^{1+\max\{2,c\}}\big)$.*

Proof. Let $P \in \mathsf{RepeatedPasting}(\mathcal{C})$ and let $T(n)$ be the running time of the algorithm we are constructing for $\mathsf{RepeatedPasting}(\mathcal{C})$, where n is the number of vertices of P. In order to find a decomposition of P into games from $\mathcal{C}$, we compute the biconnected components (that is, maximal 2-connected subgraphs) of the underlying undirected graph of P, say $P_1, \ldots, P_k$, in time linear in the number of arcs of P, for example by the algorithm presented by John E. Hopcroft and Robert E. Tarjan [HT73]. Since $P \in \mathsf{RepeatedPasting}(\mathcal{C})$ and $\mathcal{C}$ is hereditary, each such biconnected component belongs to $\mathcal{C}$.

Let $L \in \mathcal{C}$ be a leaf-component of the tree T_P associated with P. This means that L shares at most one vertex with all other components of P. If there is no such vertex, we are done because the graph is disconnected and we can easily solve different components separately. Otherwise, let v be this vertex. Without loss of generality we assume that v is a vertex that belongs to player $\Diamond$. Since all paths between L and $P' := P \setminus (L \setminus \{v\})$ use vertex v, and thus there are no cycles without repeated vertices spanning both L and P', we have that $v \in W_\Diamond(P)$ if and only if $v \in W_\Diamond(L)$ or $v \in W_\Diamond(P \setminus (L \setminus \{v\}))$.

First we solve the parity game L, which can be done in time $O(n^c)$ since $L \in \mathcal{C}$. If player $\diamond$ wins on v in L, then we solve $P \setminus (L \cup \mathrm{attr}_\diamond(v))$ recursively, in time $T(n-1)$. Since all paths between L and $P \setminus (L \cup \mathrm{attr}_\diamond(v))$ use vertex v, from which player $\diamond$ has a winning strategy inside L, we have, thanks to Lemma 3.66:

- $W_\diamond(P) = W_\diamond(L) \cup \mathrm{attr}_\diamond(v) \cup W_\diamond(P \setminus (L \cup \mathrm{attr}_\diamond(v)))$,
- $W_\square(P) = W_\square(L) \cup W_\square(P \setminus (L \cup \mathrm{attr}_\diamond(v)))$.

However, if player $\square$ wins on v in L, then we solve P' recursively, in time $T(n-1)$. Then we have two cases. If player $\square$ wins on v in P', then we merge the winning regions of L and P', that is:

- $W_\diamond(P) = W_\diamond(L) \cup W_\diamond(P')$,
- $W_\square(P) = W_\square(L) \cup W_\square(P')$.

This is correct because no matter which successor player $\diamond$ chooses on v, the game will either continue in L or in P', and $v \in W_\square(L)$ and $v \in W_\square(P')$.
Otherwise, if player $\diamond$ wins on v in P', then we have to recompute the winning regions of L, from which we remove $\mathrm{attr}_\diamond(v)$. Since $\mathcal{C}$ is hereditary, then $L \setminus \mathrm{attr}_\diamond(v) \in \mathcal{C}$, and can be solved in time $O(n^c)$. Thus, again by Lemma 3.66, and by the fact that all paths between $L \setminus \mathrm{attr}_\diamond(v)$ and P' use vertex v, from which player $\diamond$ has a winning strategy inside P', we put:

- $W_\diamond(P) = W_\diamond(L \setminus \mathrm{attr}_\diamond(v)) \cup \mathrm{attr}_\diamond(v) \cup W_\diamond(P')$,
- $W_\square(P) = W_\square(L \setminus \mathrm{attr}_\diamond(v)) \cup W_\square(P')$.

Finally, the running time of the algorithm satisfies

$$T(n) \leq n^2 + tn^c + T(n-1),$$

for some fixed $t > 1$, where n^2 accounts for the computation of the attractor sets. Thus, $T(n) \in O(n^{1+\max\{2,c\}})$. ∎

As a corollary of Corollary 6.14 and Theorem 6.20, we can solve parity games in polynomial time on every biorientation of a block graph, that is, a graph where every biconnected component is a clique.

Furthermore, this also gives us a polynomial time algorithm for solving parity games on *cactus graphs*, that is, graphs where every edge lies on at most one cycle.

On the other hand, it is unlikely that we can easily extend the above method by pasting along 2 vertices since this immediately leads to a polynomial-time algorithm for the class of all parity games by pasting along complete graphs with 2 vertices.

6.4 Adding a Single Vertex

Definition 6.21 If $\mathcal{C}$ is a class of parity games, we let $\mathsf{AddVertex}(\mathcal{C})$ denote the class of parity games obtained by adding a single vertex to every game in $\mathcal{C}$ in any possible way. Formally,

$\mathsf{AddVertex}(\mathcal{C})$

$$\mathsf{AddVertex}(\mathcal{C}) := \{P \mid P \text{ is a parity game and there exists a vertex } v \text{ such that } P \setminus \{v\} \in \mathcal{C}\}. \quad \dashv$$

Theorem 6.22 *If $\mathcal{C}$ is a hereditary class of parity games which we can solve in time $O(n^c)$ with $c \geq 2$ and we can test membership to $\mathcal{C}$ in time $O(n^d)$, then we can solve parity games*

in $\mathsf{AddVertex}(\mathcal{C})$ *in time* $O\big(n^{\max\{c,d\}+1}\big)$ *and test membership to* $\mathsf{AddVertex}(\mathcal{C})$ *in time* $O(n^{d+1})$*, where* n *is the number of vertices of a parity game.*

Proof. Let $P = (V, E, V_\Diamond, \omega)$. In order to find a vertex v such that $P \setminus \{v\} \in \mathcal{C}$, and at the same time test whether $P \in \mathsf{AddVertex}(\mathcal{C})$, we can iterate over all $v \in V$ and test the membership of $P \setminus \{v\}$ to $\mathcal{C}$, with an overall complexity of $O(n^{d+1})$.

First, we solve the two subgames $P_1 := P \setminus \mathrm{attr}_\Diamond(v) \in \mathcal{C}$ and $P_2 := P \setminus \mathrm{attr}_\Box(v) \in \mathcal{C}$ in time $O(n^c)$. If $W_\Box(P_1)$ or $W_\Diamond(P_2)$ is not empty, we return the corresponding subgame $P \setminus P_1$ or $P \setminus P_2$.

If $W_\Box(P_1) = W_\Diamond(P_2) = \emptyset$, we conclude that $W_\Box(P) = \emptyset$ or $W_\Diamond(P) = \emptyset$, analogous to the proof of Lemma 6.12 and Theorem 6.18. We can then return the original game unchanged.

The running time of this part is $O(n^c)$ because computing the attractor sets is in $O(n^2) \subseteq O(n^c)$. Lemma 6.8 then yields an algorithm that solves all games in $\mathsf{AddVertex}(\mathcal{C})$ in time $O(n^{c+1})$. Finding the vertex v that we have to remove takes time $O(n^{d+1})$, so the total running time is $O\big(n^{\max\{c,d\}+1}\big)$. ∎

This theorem implies, for example, that if parity games can be solved in polynomial time on orientations of planar graphs, then they can also be solved in polynomial time on orientations of apex graphs, which are planar graphs with one additional vertex.

6.5 Conclusions

We presented some graph operations that preserve solvability of parity games in polynomial time. In Section 6.2.2, we saw that the join of two classes of parity games is as easy to solve as the individual classes up to a polynomial factor provided a decomposition of the join is available. In Section 6.3 we considered the case of pasting many games together along vertices to form a larger game and in Section 6.4 we analyzed the problem of adding a single vertex to a parity game. In both cases we showed that the resulting classes can be solved in time only a small polynomial factor slower than the original classes.

Recently, Gajarský et al. successfully solved more general cases [Gaj+15]. They parameterized the problem in the size of a feedback vertex set, by the distance to a tournament, or by modular width and found that it is fixed-parameter tractable in these cases, which in particular implies our polynomial-time result for tournaments.

For more graph operations, it is an open problem whether our approach can be adapted. One graph operation that comes to mind is the operation of *substitution*. Particular instances of this operation are obtained by starting from a tree T whose vertices are coloured with either $\diamond$ or $\Box$ such that no two adjacent vertices have the same colour. Then we replace every i-coloured vertex with a single-player game consisting of vertices of player i and connect games that correspond to adjacent vertices in T analogous to the Join-operation defined in Section 6.2. The resulting game can be solved with dynamic programming and the help of Theorem 6.18 in time $O(n^c)$, but c depends on the depth of the tree T. These games seem to be

simpler than general parity games, so it could be reasonable to expect a polynomial time algorithm.

On the other hand, it is interesting to study whether the graph operations considered here preserve the polynomial time solvability of other games. A starting point could be parity games whose *arcs* are assigned priorities; more generally, one could consider some of our operations for mean payoff games, energy games, or simple stochastic games.

7 A Formal Proof of Positional Determinacy

> Es soll in mathematischen Angelegenheiten prinzipiell keine Zweifel, es soll keine Halbwahrheiten [...] geben können.
>
> *(David Hilbert, [Hil22])*

In this chapter we present a formal proof that parity games are positionally determined. We already saw this theorem in Section 3.2 on page 52. For convenience, we restate the theorem.

Theorem 7.1 (Theorem 3.55) *For every parity game* $P = (V, E, V_\Diamond, \omega)$ *it holds that* $V = W_\Diamond \cup W_\Box$.

This theorem has many proofs [BSV04; Küs01; Zie98], so a new proof by itself is not very exciting. What distinguishes our proof from the previous proofs is that we provide a complete formalization in the formal logic "Isabelle/HOL". This formalization has been published on the Archive of Formal Proofs (AFP) [Dit15], a reviewed collection of formalized proofs. Although the AFP has a respectable size of 1,018,800 lines of code as of April 2015, so far there had been no published proofs about games played on graphs [Bla+15]. For this reason, our publication led to the creation of the new category "Computer

Science/Games" and laid the foundation for more formalized proofs in this area.

7.1 Background

7.1.1 Formal Proofs

Usually, mathematicians write their proofs in a combination of English, formulas, and pictures. We did very much the same in the previous chapters. We call this an *informal proof*. One of the most fundamental question about proofs is how to discern correct proofs from invalid proofs.

informal proof

After a long history, modern mathematics has found a seemingly satisfying answer to this question [Eve97]. We define a formal logic by syntactically describing strings of symbols that we call *formulas*. Then we define manipulation rules for these strings of symbols, and call the collection of these rules a *calculus*. Finally we claim that everything derivable by this calculus is "true", and we call the derivations *formal proofs*. A formal proof of a statement "A implies B" then boils down to applying the rules of the calculus to the string A in such a way as to arrive at the string B.

formula

calculus

formal proof

Often, we also associate semantics with these strings of symbols, that is, meaning. For example, we want to be able to say that some symbol strings are tautologies while others are unsatisfiable. Then one crucial property of a calculus is *correctness*. Correctness means that the calculus always turns tautologies into tautologies, with the goal that starting out with a tautology and ending in some symbol string φ means that φ is a tautology, too.

While justifying every theorem with a formal proof is the

theoretical foundation of modern mathematics, in most cases this ideal is elusive. The symbol manipulation rules are far too low-level to make proving non-trivial statements by hand feasible. Furthermore, even checking a formal proof is often beyond humans because although each step in the proof is easily verified, a formal proof could have millions or more of tiny inference steps.

So most proofs in modern mathematics are still written in English with the silent assumption that every step should be expressible in a calculus, if only someone would invest the time to do so. Of course, this makes it possible for logical gaps or errors to hide in informal proofs.

With the exponential growth of computing power in recent years, the goal of having verified formal proofs for everything has become within reach by shifting the work of rigorously using the calculus rules from the user to the computer, allowing the user to write their proof in a high-level language similar to a very restricted form of English with a computer program compiling the proof down to simple rule inferences. These programs are called *proof assistant* or *theorem prover*. Usually, these programs also help the user in finding a formal proof by suggesting lemmas or even automatically proving some parts.

proof assistant

A proof assistant will not accept proofs that it cannot compile down to correct rule inferences. Thus logical gaps or errors are all but impossible in formal proofs, provided the proof assistant works correctly.

7.1.2 Computer-Assisted Proofs

When speaking of proofs assisted by a computer, what could also come to mind is the four-color theorem and the controversy

around its initial proof by Wolfgang Appel and Kenneth Haken in 1976 [AH76]. Their proof was *computer-aided* because they used a computer program to generate a comprehensive list of possible counterexamples and to refute each one of them. That this program was essential to the proof made the proof hard to verify to the same rigor as you would verify a handwritten proof.

However, while the initial proof of the four-color theorem used a computer program, it did not use a theorem prover. Only in 2005, 29 years after the first proof, Georges Gonthier succeeded in formalizing a proof in the interactive theorem prover *Coq* [Gon08]. This is the crucial difference between computer-aided proofs and formal proofs. A computer-aided proof is in no way more formal or more trustworthy than an informal proof (quite the opposite, as we saw with the four-color theorem).

What we are after is a formal proof that can be verified by a small computer program unspecific to the particular proof. For example, with Gonthier's formal proof a reviewer does not need to trust the proof or some computer program specific to this proof. Instead, they only need to trust the Coq kernel, which is intentionally small (see [Pau11] for an introduction to Coq and its architecture). Because of this, Gonthier goes as far as claiming that his proof is *more* rigorous than any handwritten proof could possibly be [Gon07].

7.1.3 Isabelle/HOL

Isabelle/HOL

A small trusted kernel is also the foundation of the interactive theorem prover *Isabelle/HOL*, the prover that we are going to use. For a formal proof in Isabelle/HOL, the user usually writes

text in a high-level language called "Isar", an abbreviation for "intelligible semi-automated reasoning". But the kernel does not understand Isar: To make it easier for the user to trust the kernel, it only accepts tiny inference steps based on natural deduction.

This design with a small reliable kernel is knows as the *LCF approach*. The idea is that except for a very small logical core, everything else can be developed by possibly declaring a few additional axioms and by building more and more complex definitions, but crucially without modifying the kernel itself.

Technically, the kernel in the case of Isabelle/HOL accepts only inferences of a far simpler language called *Isabelle/Pure*, which implements a minimal calculus of higher-order logic. Isabelle/HOL then implements the full HOL calculus (higher-order logic) on top of Isabelle/Pure. For this, Isabelle/HOL declares a small set of additional axioms for logic and set theory. The Isar language is common to both Isabelle/Pure and Isabelle/HOL. For more details, we refer the reader to [WPN08] for a short overview of the architecture of Isabelle/HOL.

Isabelle/ Pure

The Isabelle/HOL system then has the task to generate tiny inference steps from the high level proof language entered by the user. This design makes it possible for the user to formalize large proofs within a reasonable time.

A formal proof of a mathematical theorem provides very strong evidence for its truth, but it may also offer new insights because no form of handwaving is possible. All edge cases and all assumptions that are imprecise or possibly incomplete in a handwritten proof must be made explicit in a formal proof. Furthermore, a theorem prover can also generate a list of axioms that appear in the proof, which is interesting for proof theory.

For our formal proof of positional determinacy, we follow a proof by Stephan Kreutzer [Kre15], which is based on the proof given by Wiesław Zielonka [Zie98]. Because our formalization is rather long and technical with 3,579 lines of code, we will only discuss key design decisions and refer the reader to the Archive of Formal Proofs [Dit15] for the full proof text.

7.2 The Informal Proof

Because Kreutzer's proof is in German and, as lecture notes, has not been properly published, let us reproduce the proof here in natural language.

We will use many definitions from Section 3.2, in particular parity games (Definition 3.43 on page 47), winning regions (Definition 3.52), attractor sets (Definition 3.58), and attractor strategies (Lemma 3.62).

First, let us prove a slightly weaker statement.

Lemma 7.2 *Let $P = (V, E, V_\Diamond, \omega)$ a parity game without dead ends, that is, every vertex has at least one outgoing arc. Then $V = W_\Diamond \cup W_\Box$.*

Proof. We prove the statement by induction on $|\omega(V)|$, which is finite. Suppose the theorem holds for all parity games $P' = (V', E', V'_\Diamond, \omega')$ without dead ends with $|\omega'(V')| < |\omega(V)|$.

Let $k := \min \omega(V)$ be the minimum priority of P. Without loss of generality, we assume that k is even. We define the following sets.

$$U := V \setminus W_\Box$$
$$K := U \cap \omega^{-1}(k)$$

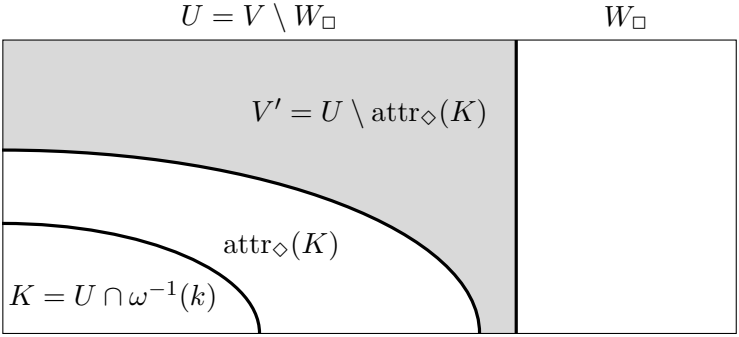


Figure 7.1: The situation in the proof of Lemma 7.2

$$V' := U \setminus \mathrm{attr}_\Diamond(K).$$

By basic set theory it follows that

$$V = \mathrm{attr}_\Diamond(K) \cup V' \cup W_\Box. \tag{7.1}$$

Let $P' = (V', E', V'_\Diamond, \omega')$ be the subgame of P induced by V'. See Figure 7.1 for an illustration of the situation here with the subgame P' indicated by the shaded area. Note that $\mathrm{attr}_\Diamond(K) \cap W_\Box = \emptyset$ because player $\Diamond$ cannot force a play away from $W_\Box$ (player $\Box$'s winning region), and $K \subseteq \mathrm{attr}_\Diamond(K)$ by the definition of the attractor set.

We have $|\omega'(V')| < |\omega(V)|$ because $k \notin \omega'(V')$. To use the induction hypothesis, we also need that P' has no dead ends. Assume to the contrary that $v \in V'$ has no successor in V', so all successors of v in P must be in $W_\Box \cup \mathrm{attr}_\Diamond(K)$, and there is at least one such successor (recall that P has no dead ends). We distinguish two cases.

1. If $v \in V'_\Box$, then no successor can be in $W_\Box$ or we would have $v \in W_\Box$. So all successors are in $\text{attr}_\Diamond(K)$, but this implies $v \in \text{attr}_\Diamond(K)$, a contradiction.

2. If on the other hand $v \in V'_\Diamond$, then not all successors can be in $W_\Box$ or we would have $v \in W_\Box$. So there exists a successor in $\text{attr}_\Diamond(K)$, but this implies $v \in \text{attr}_\Diamond(K)$, again a contradiction.

In both cases we get a contradiction, so v cannot be a dead end in P'. This allows us to apply the induction hypothesis to P'.

By the induction hypothesis, we get $V' = W'_\Diamond \cup W'_\Box$. However, it turns out that $W'_\Box = \emptyset$. Suppose to the contrary that there exists a $v \in W'_\Box$ and let $\sigma_\Box$ be player $\Box$'s winning strategy on $W'_\Box$ in P'. We arbitrarily extend $\sigma_\Box$ to all of V. We have two cases.

1. Every $\sigma_\Box$-conforming play starting from v stays in $W'_\Box$ and is thus winning for player $\Box$. Then v should have been in $W_\Box$, so we cannot have this case.

2. There exists a $\sigma_\Box$-conforming play starting from v that leaves $W'_\Box$. This play then visits a vertex in $V \setminus V'$, because if it would stay in V', we would have a contradiction to the definition of $W'_\Box$.

Let $w \in V \setminus V'$ be the first such vertex. We consider two cases.

1. If for every such play we have $w \in W_\Box$, then v should have been in $W_\Box$ to begin with, so this cannot happen.

2. There exists some play starting in v with $w \in (V \setminus V') \setminus W_\Box = \text{attr}_\Diamond(K)$. But such a play also cannot exist,

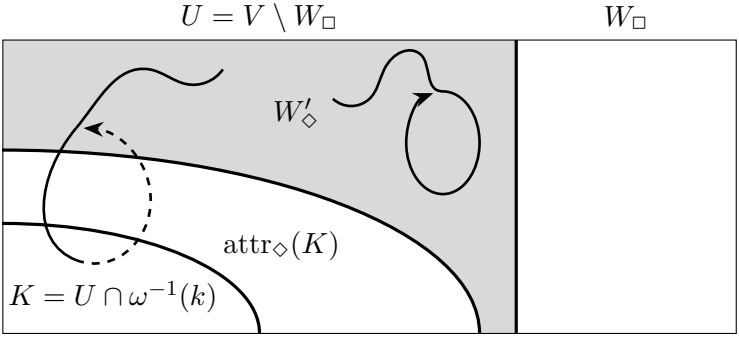


Figure 7.2: Every play is winning for player $\Diamond$

because then the predecessor of w, which is in $W'_\Box$, should have been in $\text{attr}_\Diamond(K)$ by the definition of the attractor set.

So we have $W'_\Box = \emptyset$.

Now we can finally construct a winning strategy for player $\Diamond$ on P. Let σ_A be an attractor strategy on $\text{attr}_\Diamond(K)$, let $\sigma_\Diamond$ be a winning strategy for player $\Diamond$ on $W'_\Diamond = V'$, and let σ^* be some arbitrary strategy on $V \setminus (\text{attr}_\Diamond(K) \cup W'_\Diamond)$ that avoids $W_\Box$. That is, we take an arbitrary strategy with the restriction that $\sigma^*(v) \notin W_\Box$ if v has a successor in $V \setminus W_\Box$, which is trivially possible. For all $v \in V_\Diamond$, define the strategy σ as

$$\sigma(v) = \begin{cases} \sigma_A(v) & \text{if } v \in \text{attr}_\Diamond(K) \\ \sigma_\Diamond(v) & \text{if } v \in V' \\ \sigma^*(v) & \text{otherwise.} \end{cases}$$

We claim that σ is a winning strategy for player $\Diamond$ in P on

U. Let $v \in U$ be arbitrary and let $\overline{v}$ be σ-conforming play in P starting in v. Then $\overline{v}$ is infinite because P has no dead ends.

We see by induction that $\overline{v}$ never visits $W_\Box$; here we use that σ^* avoids $W_\Box$. So we have that $\overline{v}$ stays in $V' \cup \text{attr}_\Diamond(K)$.

The situation is illustrated in Figure 7.2 with two paths $\overline{v}$ could take. The path leaving K is dashed to indicate that $\overline{v}$ cannot go from K through $\text{attr}_\Diamond(K)$ to $W'_\Diamond$, but instead it must jump away from K without going through $\text{attr}_\Diamond(K)$. Note that $\overline{v}$ may also jump from K to any other vertex in U, not necessarily in $W'_\Diamond$, and $\overline{v}$ may start in any vertex of U. Furthermore, $\overline{v}$ does not need to end up in a cycle because P could be infinite.

The only important point here is that as soon as $\overline{v}$ visits $\text{attr}_\Diamond(K)$, it will eventually also visit K. It follows that the play either eventually stays in the shaded $W'_\Diamond$ region and is thus winning for player $\Diamond$ because $\sigma_\Diamond$ is a winning strategy on $W'_\Diamond$, or the play visits K infinitely often and is winning for player $\Diamond$ because k is even. ∎

The next step is to drop the requirement that P contains no dead ends and prove the full theorem.

Theorem 7.3 (Theorem 3.55) *For every parity game $P = (V, E, V_\Diamond, \omega)$ it holds that $V = W_\Diamond \cup W_\Box$.*

Proof. For $i \in \{\Diamond, \Box\}$, define the sets of dead ends

$$D_i := \{v \in V_i \mid \text{there is no } w \text{ with } (v, w) \in E\}$$

and also define

$$A_i := \text{attr}_i(D_{\bar{i}}).$$

It is easy to see that for all $v \in A_i$ there exists a strategy σ such that v is winning for player i: the attractor strategy. In other words, $v \in A_i$ implies $v \in W_i$. It remains to consider the case $v \in V'$ with $V' := (V \setminus A_\Diamond) \setminus A_\Box$.

First, we observe that the induced subgame $P[V']$ has no dead ends. If $v \in V'$ is a dead end, then all outgoing arcs must point towards $A_\Diamond \cup A_\Box$ and there is at least one such arc because was not a dead end in P.

Suppose $v \in V_\Diamond$. If there is an arc pointing towards $A_\Diamond$, then we have $v \in A_\Diamond$ because v is in the $\Diamond$-attractor. Because we assumed $v \in V'$, this cannot be the case. So all arcs from v point towards $A_\Box$ and there is at least one such arc. But this implies $v \in A_\Box$ by the definition of $\Box$-attractor, again a contradiction. The case $v \in V_\Box$ follows analogously. This proves that $P[V']$ has no dead ends.

Now let us fix some $v \in V'$ and apply Lemma 7.2 to get a player i and a strategy σ' on $P[V']$ which is winning for player i from v. We extend σ' to a strategy σ on P with the attractor strategies of $A_\Diamond$. If a σ-conforming play starting from v stays in V', then it follows σ' and is winning for player i. If it enters $A_\Diamond$, then it is also winning for player $\Diamond$ because the attractor strategy on $A_\Diamond$ makes sure that eventually the play hits a dead end of player $\Box$.

Assume the play enters $A_\Box$. This can only happen from a *player* $\Box$-vertex w in V'. But then w has an outgoing arc towards $A_\Box$, so w should have been in $A_\Box$ already. So the play cannot enter $A_\Box$.

In all cases, the play is winning for player i, so σ is a winning strategy for player i from v in P. Because $v \in V'$ was arbitrary and because we already know the winning strategies for $v \in A_\Diamond \cup A_\Box$, we have $V = W_\Diamond \cup W_\Box$. ∎

7.3 Isabelle Primer

We would like to sketch a few key technical points of our formal proof. However, this requires a little familiarity with the Isabelle/HOL system. A thorough introduction to Isabelle/HOL is given in [NPW16], which is a more recent version of [NPW02].

Because that introduction has over 200 pages, we will here present a heavily summarized and shortened tutorial sufficient to give the reader an idea of how our formal proof in this system works. Note that we will pick and choose and present mostly the aspects of Isabelle/HOL that are relevant to our formal proof and omit many features that you would see in a more general introduction.

If you are already familiar with Isabelle/HOL, you may safely skip ahead to Section 7.4 on page 226, where we will present the formal proof of positional determinacy.

7.3.1 Syntax

syntax

Syntactically, Isabelle/HOL distinguishes between the *outer syntax* and the *inner syntax*. Intuitively, the inner syntax describes formulas and the outer syntax describes the scaffolding of a proof. For now, let us assume for simplicity that every mathematical formula that we encounter, such as $\forall x.P(x)$ or $\exists y.f(x) = y$, is an example of inner syntax.

Of course, the inner syntax has strict syntax rules, but the above are proper examples of formulas written in inner syntax because Isabelle/HOL accepts and understands Unicode characters such as $\forall$, and the grammar of the inner syntax is extremely flexible.

The default is to parse a proof text according to the rules

of the outer syntax and all strings enclosed in "..." or ⟨...⟩ according to the rules of the inner syntax. So it is easy to distinguish if something is outer syntax or inner syntax.

7.3.2 Context

One of the most fundamental parts of a proof in Isabelle/HOL is the *context*. Essentially no matter where we are in a proof document, at every cursor position there is a current context. context

A simple context consists of a finite set of *known facts* and a finite set of *goals*, the formulas that we want to prove. facts

A *theory context* is special in the sense that it has no goals, only a set of known facts. After a tiny setup, every proof document starts in a theory context. theory context

A context is *local* if it is not a theory context. Every known fact in a theory context is called a *lemma*, a *theorem*, a *corollary*, or a *proposition* (which are all the same internally). Other than that, a theory context is not much different from local contexts, except that most keywords for manipulating contexts have different names and different syntax in theory contexts and in local contexts. local context

There is one more minor complication: Every local context can be in one of two *modes*: *state* mode or *prove* mode. Technically, there is more, but we simplify here for the purposes of this exposition. The rule of thumb is that we are always in *state* mode except under certain circumstances that we will point out. mode

Let us begin with a very simple lemma, a lemma stating "$A \wedge B$". We note that as a general lemma, this will be unprovable (and false) unless A and B have definitions evaluating to equal objects.

The reason is that as a general lemma, A and B will be implicitly universally quantified, so the lemma states that "A and B" is true for every A and every B, which is obviously false.

Let us assume for now that A and B are something meaningful. We will soon see how we can add local assumptions to write a lemma of the form "A and B implies $A \wedge B$", which is true also with the implicit universal quantification of variables.

Going back to our simple lemma stating $A \wedge B$, we can write this in a theory context as

```
lemma "A ∧ B" proof-
  ...
```

This opens a new local context with the single goal $A \wedge B$ and a set of known facts consisting of every lemma and every theorem known to the system.

prove mode

Directly before **proof-** in the above line, the new local context already exists, but it is in *prove* mode. The keyword

state mode

proof- switches the context into *state* mode. This is a minor technicality but useful to know because most keywords are valid only in one of the two modes. For example, **proof-** is valid only in *prove* mode and **lemma** is valid only in *state* mode (however, as we will see, **lemma** is invalid in local contexts).

In the Isar language, a good first approximation is that we can never modify a goal in *state* mode, only remove them, and that in the end we want to have an empty set of goals. If the set of goals becomes empty, then we consider the current local context as done and can close it with the keyword **qed**.

As one might expect, we can remove a goal X and consider it *solved* by adding X to the set of known facts. Here it is

crucial that exactly X must become a known fact. For example, the goal $A \wedge B$ will not be considered solved if $B \wedge A$ becomes a known fact.

7.3.3 Adding New Facts

The main way to manipulate a local context is the keyword **have**, which adds something to the set of known facts. This corresponds to **lemma**, which does the same to a theory context. They keyword **lemma** is invalid in local contexts and **have** is invalid in theory contexts, but apart from that they are essentially the same. For example, consider the line

```
have "A ∧ B" proof-
  ...
```

This line is perfectly equivalent to the **lemma** line we saw above. The only difference if that **lemma** is for theory contexts and **have** is for local contexts. We can also think of **have** as adding local lemmas to the local context.

This example with **have** adds the formula $A \wedge B$ to the set of known facts in the current local context. Analogous to **lemma**, **proof**- opens a local context with the single goal $A \wedge B$.

7.3.4 Solving Goals

For readability, the Isabelle/HOL system insists on seeing the keyword **show** instead of **have** when we want to solve a goal in a local context. The keyword **show** works exactly the same as **have** except that in addition to adding the formula to the set of known facts, it also marks the matching goal as solved and removes this goal from the set of goals (and fails if it cannot

find a matching goal). So in order to finish the proof of $A \wedge B$ in the local context, we would need to write

```
show "A ∧ B" ...
```

In all other aspects, **show** is identical to **have**. In particular, **show** is invalid in a theory context.

If there is only one goal, then *?thesis* (optionally with quotation marks) is an abbreviation for this goal. So you will often see proofs ending with

```
show ?thesis ...
```

So the general structure of a proof of a lemma will be as follows.

```
lemma shows "A ∧ B" proof-
  ...
  show "A ∧ B" ...
qed
```

We can write **qed** here to finish the proof because **show** removes the single goal from the context, leaving us with a context with no more goals.

7.3.5 Assumptions

Most lemmas in mathematics are of some form of implication such as "A and B implies $A \wedge B$". We write these implications in Isabelle/HOL as $A \Longrightarrow B \Longrightarrow A \wedge B$, where $\Longrightarrow$ is right-associative.

However, for technical reasons this form turns out to be inconvenient for proving all but the simplest of lemmas. The reason is that the local context of such a lemma contains the single goal $A \Longrightarrow B \Longrightarrow A \wedge B$ and no additional facts, whereas we would like to have the single goal $A \wedge B$ with the additional facts A and B.

Whenever we have a local context with a goal of the form of an implication $A \Longrightarrow B$, we can pull the assumption A into the set of known facts by writing **assume** *"A"* somewhere in the local context.

For example, we can write

```
lemma shows "A ⟹ B ⟹ A ∧ B" proof-
  assume "A" and "B"
  ...
  show "A ∧ B" ...
qed
```

But there is an easier way. We can write our lemma as follows.

```
lemma assumes "A" and "B" shows "A ∧ B" proof-
  ...
  show "A ∧ B" ...
qed
```

Then our local context automatically contains the additional facts A and B and the single goal $A \wedge B$, as expected. When the proof is done, then the above lemma adds to the set of facts of the enclosing theory context the new fact $A \Longrightarrow B \Longrightarrow A \wedge B$.

This is the same as if we would have written

```
lemma "A ⟹ B ⟹ A ∧ B" proof-
  ...
```

So in the end, using **assumes**...**shows** makes no difference, it only simplifies the local context created for proving the lemma.

When adding local facts with **have** or **show**, we can do exactly the same, but the syntax is completely different. It is a postfix syntax as follows.

```
have "A ∧ B" if "A" and "B" proof-
  ...
 show "A ∧ B" ...
qed
```

Other than the different way of writing this, this is completely equivalent to the **assumes**...**shows** syntax for **lemma**.

Let us summarize with Table 7.1 the differences between the three keywords that introduce new facts. There are more differences, but these are the most important ones.

Let us also remark here that the keywords **theorem**, **proposition**, and **corollary** all are aliases for **lemma**. They can make proofs more readable but their distinction has no relevance to the Isabelle/HOL system.

7.3.6 Proving Facts

Of course, we cannot add something to the set of known facts by simply saying with **lemma**, **have**, or **show** that we would like to do so. We need to *prove* that the new fact really follows logically from the currently known facts. This is why in place

	lemma	**have**	**show**
Permissible	only in theory context	only in local context	only in local context
Special feature	—	—	looks for and removes matching goal
A and B implies $A \wedge B$	**lemma assumes** *"A"* **and** *"B"* **shows** *"A ∧ B"*	**have** *"A ∧ B"* **if** *"A"* **and** *"B"*	**show** *"A ∧ B"* **if** *"A"* **and** *"B"*

Table 7.1: Different keywords for proving new facts

of the ellipsis ... in our examples, we need to indicate to the Isabelle/HOL system how it can derive the new fact $A \wedge B$ from the current set of known facts.

As mentioned before, when we write a global or local lemma with **lemma**, **have**, or **show**, immediately after the statement is the only place where a local context is in *prove* mode. As the name indicates, this is the mode where we can prove something.

There are essentially two ways to tell the Isabelle/HOL system how it can prove something from the set of known facts. One is to write **proof-** to switch to *state* mode. The new local context then has the same set of known facts as the enclosing context, but the single goal is now $A \wedge B$. Switching the local context to *state* mode, showing all goals with **show** and then closing the context with **qed** is the usual way of proving non-trivial facts.

So we could write

```
lemma "A ∧ B" proof-
  show "A ∧ B" proof-
    ...
```

But this gets us nowhere. The context after the **show** is exactly the same as the enclosing context; it consists of the single goal $A \wedge B$, and the set of known facts is unchanged.

7.3.7 Proof Methods

The other way of proving facts is the keyword **by**, valid only in *prove* mode, followed by the name of a proof method. There is a large number of proof methods in Isabelle/HOL, and many of them have numerous customization options. Proof methods

make up a large part of the complexity of Isabelle/HOL from a user's perspective.

Probably the most common methods are *blast*, *simp*, and *auto*, which implement first-order solvers, simplifications, and heuristics to prove statements. Usually, *auto* can solve more complex statements than *blast* or *simp*, but this is not always the case. In practice, one may need to try several methods until one succeeds.

In our case of proving $A \wedge B$ from A and B, all three methods work fine. So one would expect the following line to be a correct Isar proof.

```
lemma assumes "A" and "B" shows "A ∧ B" by blast
```

Unfortunately, this line will be marked red in the visual editor Isabelle/JEdit and receive the unspecific error "Failed to apply initial proof method".

This error is a common cause of frustration among new Isabelle users. But the reason here is obvious in hindsight if you know how *blast* works.

What happens here is that *blast* does not look at the whole set of known facts because it is huge (it contains every lemma and every theorem proved so far). We need to explicitly tell the proof method with the keyword **using** which of the known facts it should consider, quoting facts by enclosing them in $\langle \ldots \rangle$.

So a correct proof is the following.

```
lemma assumes "A" and "B" shows "A ∧ B"
  using ⟨A⟩ ⟨B⟩ by blast
```

A perfectly equivalent, but overly verbose, proof would be the following.

```
lemma assumes "A" and "B" shows "A ∧ B" proof-
  show "A ∧ B" using ⟨A⟩ ⟨B⟩ by blast
qed
```

7.3.8 Rules

One mystery that remains is how exactly *blast* manages to prove the goal. Here is where the calculus called HOL (higher-order logic) comes in, because *blast* uses the rules of the calculus (and more) automatically. This calculus is defined in the Isabelle/HOL standard library in the file `src/HOL/HOL.thy` and consists mainly of lemmas such as

```
lemma "P ⟹ Q ⟹ P ∧ Q"
```

This lemma says that P and Q together imply $P \wedge Q$, where P and Q are implicitly universally quantified. We can use this lemma directly with the proof method *rule* as follows.

rule

```
lemma assumes "A" and "B" shows "A ∧ B"
  using ⟨A⟩ ⟨B⟩ by (rule ⟨P ⟹ Q ⟹ P ∧ Q⟩)
```

The *rule* method looks at the currently available facts (made available by **using**) and tries to match these facts with the assumptions of the rule and the current goal with the conclusion of the rule. If at any point the matching fails, then the method fails.

Formally, the matching is done by *unification*. The unification process takes two formulas and instantiates the free variables in both formulas in such a way that the two formula become identical. unification

In the case of the above proof, the *rule* method works fine because we easily have an exact match (P to A and Q to B). But we see that it is very inconvenient to manually quote lemmas like this. This is why Isabelle/HOL allows *named* facts.

7.3.9 Naming Facts

As we saw, in order to use a lemma, we need to quote it like $\langle P \Longrightarrow Q \Longrightarrow P \wedge Q \rangle$. This quickly becomes unwieldy, especially the more complex a lemma gets. There is a way of *naming* facts by writing an identifier followed by a colon in front of the fact, as illustrated by this lemma from the HOL library.

```
lemma conjI: "P ⟹ Q ⟹ P ∧ Q"
```

This allows us to quote this lemma by writing `conjI` instead of $\langle P \Longrightarrow Q \Longrightarrow P \wedge Q \rangle$. Naming facts works for every keyword that introduces new facts into a context, so in particular it works for **lemma**, **have**, and **show**.

So our proof of $A \wedge B$ can be simplified to

```
lemma myConjI: assumes "A" and "B" shows "A ∧ B"
  using ⟨A⟩ ⟨B⟩ by (rule conjI)
```

We also named our lemma `myConjI` to illustrate where the identifier goes in the **assumes**...**shows** syntax.

Again, proof by *rule* is possible, but usually it is easier to use other, more powerful proof methods such as *auto*. However,

if you know exactly which lemma you want to apply, then *rule* may be preferable.

7.3.10 Finding Facts

One question that probably every beginner asks is how to find the name of a lemma, for example *conjI*. Finding useful lemmas among the existing lemmas is crucial for successful formalizations. The HOL library defines explicitly more than 10,000 lemmas, each with its own name (one of which is *conjI*). In addition to this, it defines many lemmas implicitly via definitions, locales, and other syntactic features (we will explain some of them later). It is reasonable to assume, and true, that many non-trivial results already exist in some form, but it seems hopeless to find them.

Fortunately, there are several ways of finding lemmas. One way is to guess the name of the lemma such as *conjI*. This works for basic lemmas because the HOL library aims to follow a consistent naming scheme.

sledge-hammer

Arguably the most productive way to find lemmas in Isabelle2016 is to run *sledgehammer* on a statement. This takes a few seconds and can be achieved either by clicking the sledgehammer button in the visual editor or by writing "sledgehammer" in *prove* mode where the proof should go. Sledgehammer is a powerful tool of Isabelle/HOL that applies heuristics and calls external SAT and SMT[1] solvers to come up with a fully automatic proof.

Another method of finding useful lemmas is to read the HOL

[1] SMT stands for "satisfiability modulo theories". These are first-order solvers with a background theory such as the theory of the natural numbers.

library. The HOL library is mostly well-organized and easy to browse, so even though it has more than 10,000 lemmas, it is feasible to find specific lemmas if you broadly know what you need. This works well if the lemma already exists in slightly different form than what you need, preventing sledgehammer from finding it. For example the pigeonhole principle has many different forms of stating it and the HOL library contains only a selection of these variants.

Yet another way to find lemmas is the interactive theorem search in Isabelle/jEdit. Here you can enter some assumptions and goals and it looks for matching lemmas. This search can sometimes be fickle, so the author had less success using this method.

All the above assumes that you have a specific lemma or local fact that you expect to prove with a one-line proof. Coming up with good local facts or good lemmas is a whole other category. The mentioned tool sledgehammer does not help here. This is the point where you need to come up with the right lemmas, theorems, data structures and predicates so that in the end you have a complete proof. This is the creative and difficult part in writing a formal proof, and essentially no different from writing a proof on paper in detail.

7.3.11 Quantification

Up to now, we skipped over quantification of variables and said that everything undefined is implicitly universally quantified. While this is mostly true, in some cases it is preferable (and sometimes necessary) to make quantification explicit.

Let us revisit the definition of *context*. We introduced contexts (theory and local contexts) as a set of known facts

together with a set of goals. What we omitted was that a context also contains a list of fixed variables.

Every new variable that a **lemma** introduces in its statement automatically becomes a fixed variable in the local context created by this lemma. A theory context also has a list of fixed variables; we will see later in sections 7.3.14 and 7.3.15 how to add variables to this list.

A lemma such as

```
lemma "A" and "B" shows "A ∧ B"
```

cannot be fully understood without knowing the list of fixed variables of the enclosing theory context. If A and B are not fixed, then the lemma implicitly universally quantifies A and B, and the lemma might as well talk about P and Q or any other pair of fresh variables.

If, however, A and B are fixed by the enclosing theory context, then the lemma talks about the same A and B as the theory context. The theory context might contain other facts about A and B, and those will be available in the proof of the lemma.

There are two ways for a variable that occurs free in the statement of a **lemma** to become a universally quantified variable for this lemma:

1. If it is not fixed in the enclosing context or
2. if it is explicitly universally quantified in the statement of the lemma (introduced below).

Arguably, in the second case the variable is not free, but the explicit quantification does not usually happen in the inner syntax, as we will see.

Universal quantification in the first case does not apply to local lemmas stated with **have**/**show**; local lemmas must always quantify their new variables explicitly.

As with most language features, the syntax of *universal quantification* of variables is different for **lemma** and for **have**/**show**. For **lemma**, universal quantification of A and B uses the keyword **fixes**, as follows.

universal quantification

```
lemma fixes A and B assumes "A" and "B"
  shows "A ∧ B"
  using ⟨A⟩ ⟨B⟩ by (rule conjI)
```

In contrast to this, the postfix notation for **have**/**show** is reversed and looks as follows.

```
have "A ∧ B" if "A" and "B" for A and B
  using ⟨A⟩ ⟨B⟩ by (rule conjI)
```

We already introduced the notation $A \Longrightarrow B$ for **assumes** *"A"* **shows** *"B"*. Similarly, explicit universal quantification can be incorporated into this notation with the operator $\bigwedge$, read "for all". So an equivalent way of stating the local lemma above is:

```
have "⋀A B. A ⟹ B ⟹ A ∧ B" by (rule conjI)
```

We do not need **using** here because, as discussed in Section 7.3.5, the single goal is $A \Longrightarrow B \Longrightarrow A \wedge B$, so the assumptions are already available.

The reader might have noticed that *existential quantification* is suspiciously absent. The reason is that the quantification

existential quantification

described so far is the low-level quantification of Isabelle/Pure, the foundation of Isabelle/HOL, which does not have existential quantification. Instead, existential quantification is modeled via two rules defined by Isabelle/HOL:

```
lemma exI:
  shows "P x ⟹ ∃x. P x"
lemma exE: assumes "∃x. P x" and "⋀x. P x ⟹ Q"
  shows "Q"
```

The lemma `exI` says that if $P\ x$ holds for some fixed P and some witness x, then we can conclude that $\exists x.\ P\ x$ holds. The lemma `exE` says that if $\exists x.\ P\ x$ is true and if every x satisfying $P\ x$ yields a true Q, then Q must be true (by simply plugging into the second assumption the witness whose existence the first assumption postulates). These two rules allow us to introduce and eliminate $\exists$ without knowing anything else about $\exists$.

For syntactical reasons, the universal quantifier $\bigwedge$ cannot occur arbitrarily nested inside of terms, so Isabelle/HOL defines similarly a universal quantifier $\forall$ and two rules for introducing and eliminating $\forall$. Whenever possible, it is best to prefer $\bigwedge$ over $\forall$, because $\forall$ introduces another layer of indirection, making lemmas using $\forall$ harder to apply in practice.

Similarly, Isabelle/HOL defines $\longrightarrow$ as logical implication because $\Longrightarrow$ cannot occur arbitrarily nested inside of terms. Here, too, it is best to prefer $\Longrightarrow$ whenever possible.

Although the rule `exI` for $\exists$-introduction looks like constructive mathematics where we can only prove $\exists x.\ P\ x$ by constructing a witness term t such that $P\ t$ holds, Isabelle/HOL is not constructive. Together with the other rules such as classical proof by contradiction $(\neg P \Longrightarrow \bot) \Longrightarrow P$ and the

rules for $\forall$, it is easy to derive non-constructive rules such as $(\forall x.\ \neg P\ x \Longrightarrow \bot) \Longrightarrow \exists x.\ P\ x$.

7.3.12 Types

types

Up to now we successfully ignored *types* because Isabelle/HOL offers good type inference, so explicit type annotations are rarely necessary. Let us remark that if a term τ has type $'a$, then this is denoted as $\tau\ ::\ 'a$.

The type system of Isabelle/HOL is based on the simply typed λ-calculus [Chu40] and uses Hindley-Milner type inference. Most types are built inductively according to few rules.

- Basic types such as nat, the type of the natural numbers, and bool, the type containing the two elements *true* and *false*.
- Type variables such as $'a$, standing for an arbitrary but fixed type.
- Function types $'a \Rightarrow 'b$. An element of $'a \Rightarrow 'b$ is a total function mapping elements of type $'a$ to elements of type $'b$.
- Set types $'a$ set. This is the type containing all sets of elements of type $'a$.
- Tuple types $'a \times 'b$ containing all tuples (x, y) with $x\ ::\ 'a$ and $y\ ::\ 'b$.
- Record types. These correspond to records in programming languages and are basically extensible tuple types with names for the individual entries.

- Alias types. The keyword **type_synonym** declares that a type is an alias for a different type.
- Algebraic and coalgebraic datatypes such as finite or infinite lists. We will introduce these in Section 7.5.2.

7.3.13 Exploring Facts

When you know the name of a fact, you can print its meaning with **print_statement**. This is very useful for debugging the current context or for exploring available facts. For example,

```
print_statement myConjI
```

This prints the following, no matter whether we explicitly or implicitly quantified our variables.

```
theorem myConjI:
  fixes A :: "bool" and B :: "bool"
  assumes "A" and "B"
  shows "A ∧ B"
```

As you see, this is identical to the lemma that we wrote, except that it defaults to **theorem** instead of **lemma** (which is the same) and that it adds type annotations to the variables.

A variant of **print_statement** is **thm**. The output is similar but much more concise. For example, **thm** `myConjI` prints only

```
"⟦ ?A; ?B ⟧ ⟹ ?A ∧ ?B"
```

The technical details are of little importance here, but the question marks here identify variables that are universally

quantified on a low level, and $[\![?A; ?B]\!] \Longrightarrow ?A \wedge ?B$ is an abbreviation for $?A \Longrightarrow ?B \Longrightarrow ?A \wedge ?B$, a syntax we have already seen.

7.3.14 Definitions

Definitions are very simple in Isabelle/HOL. Say we would like to define a function $f(x) = x + x$ on natural numbers (or more generally, on anything that has an addition operator), then we can give this definition to Isabelle/HOL as follows. definitions

```
definition "f x ≡ x + x"
```
definition

From this syntax, Isabelle/HOL automatically determines that we want to define f as a function with one parameter. Essentially all this does semantically is to introduce a new constant f of the appropriate function type together with the following lemma, which is proved automatically.

```
lemma f_def: fixes x shows "f x ≡ x + x"
```

The symbol ≡ denotes equivalence in Isabelle/Pure, the underlying framework of Isabelle/HOL, and is necessary in the definition for Isabelle/HOL to determine which symbol we want to define.

There is also a long form for definitions:

```
definition f :: "nat ⇒ nat" where "f x ≡ x + x"
```

The long form makes the defined symbol and optionally the

type explicit. Apart from this, the long form is equivalent to the abbreviated form.

The keyword **definition** is valid only in a theory context. As with most language features, local contexts have an equivalent keyword called **def**. The keyword **def** has no long form, it has only an abbreviated form with the additional restrictions that the defined symbol cannot have parameters and that it must be outside the quotes:

def

```
def f ≡ "λx. x + x"
```

def lacks the syntactic sugar for defining $f\ x$ directly. Instead we need to define f as a *lambda function*. In general, a term of the form $\lambda x.\ t$ describes a function with one parameter evaluating to $t[x/a]$ on input a, where $t[x/a]$ denotes the result of substituting in t all free occurrences of x by a. In the end, this is almost equivalent to the direct **definition** of $f\ x$, the only difference being that for **definition**, the lemma `f_def` says $f\ x \equiv x + x$, but for **def**, `f_def` says $f \equiv \lambda x.\ x + x$.

lambda function

The introduction of the lemma `f_def` (by **definition** or by **def**) means that later in proofs we can work with f as an opaque function, and whenever we need to use the definition of f, we can prove a local lemma with

```
have ... using f_def ...
```

Another way, stating the intention more clearly, is to use **unfolding** instead of **using**. This also makes sure that *f_def* will be used to unfold every occurrence of f and then be forgotten, so that the proof method afterwards sees neither f

nor *f_def*. This usually makes **unfolding** preferable over **using**, when possible.

```
have ... unfolding f_def ...
```

We can add an arbitrary number of **using** and **unfolding** clauses in any order. An **unfolding** statement applies to all facts that appear before it in the **have** line, as well as the goal. Note that this means that **unfolding** is one of the commands that manipulates a goal without removing it. This is fine, because we are in *prove* mode.

7.3.15 Locales

In the context of a mathematical theory, for example graphs, we usually prove many lemmas that start with "Let G be a graph". We could quantify G and add the assumption that G is a graph to each lemma individually. But Isabelle/HOL offers a simpler way called a *locale*. locale

A locale opens a new theory context but with some additional variables and some additional facts that we can choose freely. In particular, because it is a theory context and not a local context, in a locale we can use **lemma** to add a fact to the locale, but not **have**. If we add a lemma, then the same fact will be added to the enclosing theory context, enriched with the assumption that the locale is satisfied.

Let us give a simple example.

```
locale L = fixes A B assumes a: "A" and b: "B" begin
  lemma myConjI: "A ∧ B" using a b by blast
end
```

The proof of the lemma `myConjI` works because A and B are available as facts in the context of L.

Outside the locale, the lemma will not be available under the name `myConjI`. But it will be available under the qualified name `L.myConjI` in slightly different form. If we look at the definition of `L.myConjI` with **print_statement**, we find

```
theorem L.myConjI:
  fixes A :: "bool" and B :: "bool"
  assumes "L A B"
  shows "A ∧ B"
```

We see that the locale introduced a predicate L with two parameters. The definition of this predicate should reflect the assumptions of the locale. And indeed it does, because the locale automatically proves the following lemma.

```
theorem L.intro:
  fixes A :: "bool" and B :: "bool"
  assumes "A" and "B"
  shows "L A B"
```

7.3.16 Further Reading

There are many more very useful features of Isabelle/HOL that we did not mention. One example is algebraic (co-)datatypes, which we will introduce later. As a first approximation for the rest of Isabelle/HOL, you may safely assume that everything we introduced has dozens of special cases and extensions that we skipped.

initial proof method

For example, the dash at the end of **proof**- is called the *initial proof method*. An initial proof method modifies the

newly created context, including the goals, before switching to *state* mode. The initial proof method does not need to be the no-op method denoted by the dash. Usually, however, it is advisable to only use simple rules or induction rules as the initial proof method, and to avoid complex methods such as *auto*, *simp*, and similar because of the unpredictability of how they affect the context.

raw proof blocks

final proof methods

Other important features are *raw proof blocks* (another way of adding local facts, equivalent to **have**) and *final proof methods*. Similar to an initial proof method you can also provide a final proof method after **qed** to prove the remaining goals, which can be useful if the remaining goals are very simple to prove, for example by *simp*.

Another aspect that we ignored are the many different ways of how to present local facts to proof methods and the many abbreviations for commonly used statements such as "use the previous fact". Yet another aspect that we did not mention is how to instantiate existentially quantified variables. This is nothing essentially new if you know contexts, but of course it has its own syntax and its own quirks.

For details on these and more features, we refer the reader to the Isabelle/HOL documentation [NPW16]. However, in order to effectively write formal proofs you also need to know what theories already exist. In Section 7.3.10, we already gave some advice on how to find interesting lemmas. We recommend to everyone who wants to learn to write formal proofs to read existing theories of high quality. We think that the primary source of high-quality theories should be the HOL source code in the directory `src/HOL/` in the Isabelle home directory.

The reader should be aware that the quality of the theories published on the Archive of Formal Proofs varies wildly, partly

apply style

due to old theories written in an outdated style called the *apply style*, that modern Isabelle/HOL has deprecated. In the apply style, you do not only add new facts to a context like we did, but instead you freely modify the goals and known facts directly (still according to the logic, of course), usually without saying what the result should be, only specifying the methods, by staying in *prove* mode for an extended length. This can make proofs hard to read because by specifying only the methods and not the results, the current state of the context is difficult to determine.

7.4 The Formal Proof

After this very broad overview over Isabelle/HOL, let us come back to the proof of positional determinacy of parity games. The formal proof of Lemma 7.2 and of Theorem 7.3 can be found in the file `PositionalDeterminacy.thy` of [Dit15]. The main result expressed in the Isar language reads as follows.

```
theorem partition_into_winning_regions:
  shows
    "V = winning_region Even ∪ winning_region Odd"
    and
    "winning_region Even ∩ winning_region Odd = {}"
```

Let us look into the definition of *winning_region*, where $p \in \{\diamond, \Box\}$ is a player.

```
definition "winning_region p ≡
    { v ∈ V. ∃σ.
      strategy p σ ∧ winning_strategy p σ v }"
```

This defines the (positional) winning region of player p as the set of vertices for which there exists a strategy σ for player p which is a winning strategy from v. Let us look at the next definition.

```
definition "winning_strategy p σ v ≡
    ∀P. vmc_path G P v p σ ⟶ winning_path p P"
```

A strategy σ is a winning strategy from a vertex v for a player p if all valid maximal σ-conforming paths are winning paths for player p. Here we use the abbreviation *vmc_path* for the concept of "valid maximal σ-conforming path", where "valid" simply means that the path walks along arcs.

Consider again the statement of Lemma 7.2, which proves positional determinacy without dead ends. The full statement and proof of this lemma is as follows.

```
theorem positional_strategy_exists_without_deadends:
  assumes "v ∈ V" "⋀v. v ∈ V ⟹ ¬deadend v"
  shows "∃p. v ∈ winning_region p"
using assms ParityGame_axioms
  by (induct "card (ω ‘ V)"
      arbitrary: G v rule: nat_less_induct)
     (rule
      ParityGame.positional_strategy_induction_step,
      simp_all)
```

The basic idea is that the proof runs the *induct* tactic on the cardinality of the set $\omega(V)$. In Isar notation, this is *card* $(\omega$ ‘ $V)$. The induction hypothesis should be:

For all games $G = (V', E', V'_\diamond, \omega')$ with $|\omega'(V')| <$

> $|\omega(V)|$ and for all $v \in V'$, there exists a player i and a winning strategy σ for player i from v in G.

Because G and v are universally quantified in this hypothesis, we need to declare these variables as "arbitrary" in the Isar code. Finally, we use the rule *nat_less_induct*, which is a basic induction rule for natural numbers without special treatment of the case $n = 0$:

$$\Big(\forall n \in \mathbb{N}.\ (\forall m < n.\ \varphi(m)) \Longrightarrow \varphi(n)\Big) \Longrightarrow \forall n \in \mathbb{N}.\ \varphi(n).$$

The other key ingredient is *ParityGame.positional_strategy_induction_step*, which is a lemma that proves the induction step. This lemma contains the meat of the proof. The formalization is rather long, but it stays very close to the informal proof of Lemma 7.2 given in Section 7.2.

7.5 Technical Aspects

Let us now sketch a few technical aspects of our proof of Theorem 7.3. The most important question in this formalization is how to represent graphs (parity games) and paths with concrete data structures.

7.5.1 Graphs

First, let us consider graphs and parity games. Mathematically, a directed graph with loops is a tuple (V, E) of a set of vertices and a set of arcs with $E \subseteq V \times V$. In Isabelle/HOL we can express this directly with records.

```
type_synonym 'a Arc = "'a × 'a"
record 'a Graph =
  verts :: "'a set" ("V₁")
  arcs :: "'a Arc set" ("E₁")
locale Digraph =
  fixes G (structure)
  assumes valid_arc_set: "E ⊆ V × V"
```

First, we define an arc as a pair of vertices. As described in Section 7.3.12, the syntax $'a$ denotes a *type variable*, that is, a polymorphic type. In our case this means that the vertex type could be any arbitrary but fixed type, for example the type of natural numbers. A graph with vertex type $'a$ is then a record consisting of a set of vertices and a set of arcs. A record cannot have constraints such as $E \subseteq V \times V$, so we declare a simple locale enforcing this constraint. A locale is essentially a context declaring some fixed variables together with some assumptions.

The special declaration "(**structure**)" is only for convenience. It allows us to write V and E most of the time without explicitly referring to the graph G. If we omitted the **structure** keyword, we would need to write V_G and E_G every time, adding useless clutter. This corresponds to our convention in this thesis of writing V and E instead of $V(G)$ and $E(G)$ whenever there is no confusion.

Even with this declaration, we have the option of explicitly writing V_G and E_G whenever we want to, for example when we prove a lemma mentioning more than one graph.

The symbol ı is special in the Isar language and represents the current graph G in syntax declarations. This is why we need to write Vı and not only V in the definition of the record.

A parity game is a natural extension of the above record and locale.

```
record 'a ParityGame = "'a Graph" +
  player0 :: "'a set" ("V0₁")
  priority :: "'a ⇒ nat" ("ω₁")

locale ParityGame = Digraph G
  for G :: "('a, 'b) ParityGame_scheme" (structure)
  +
  assumes valid_player0_set: "V0 ⊆ V"
    and priorities_finite: "finite (ω ‘ V)"
```

We extend the record with a set $V_\diamond$, called $V0$ in the code, and with a function $\omega : V \to \mathbb{N}$. The locale adds the restrictions that $V_\diamond \subseteq V$ and that $|\omega(V)|$ is finite. Although not necessary for the proof of positional determinacy, we refer to *ParityGame_scheme* instead of *ParityGame* to allow instantiation of this locale with *ParityGame*-derived records.

We would like to mention that it is also possible to represent a graph as a function $'a \Rightarrow 'a \Rightarrow bool$ without further assumptions. An example of this technique can be seen in the formalization of a proof of König's Lemma by Andreas Lochbihler [Loc10]. This technique simplifies the representation in some sense because it needs neither records nor additional assumptions. However, we believe that our approach is better suited to prove results for parity games, because we do not require that the set of vertices is $UNIV :: 'a\ set$, the universe of the type $'a$. Our approach allows arbitrary sets to be the set of vertices, so it becomes easier to reason about subgraphs.

7.5.2 Paths

Compared to the definition of parity games, the definition of a path is a far more subtle question. If a path could only be finite, then the answer would be easy: Isabelle comes with finite lists, so we can use these to define paths. Unfortunately, the paths we need for parity games can be finite or infinite. Ideally, we would like to handle both kinds in a uniform way because for many proofs, the difference is irrelevant.

One approach is to recall basic mathematics, where infinite sequences of vertices are usually formally defined as functions $\mathbb{N} \to V$. We can use this definition in Isabelle, too. However, then the question arises of how to represent finite paths. There are two natural choices that the author is aware of: One is to use the *option* type, which defines $V_\bot := V \cup \{\bot\}$ with a bottom element $\bot$, and then require that a path $f : \mathbb{N} \to V_\bot$ either never becomes $\bot$ or if it becomes $\bot$, then it stays $\bot$.

The other approach is to use functions $f : \mathbb{N} \to V$ and never look at values beyond n with $f(n) = v$ if v has no successors.

We found that both methods have a clear-cut distinction between finite and infinite paths and they also make it rather difficult to append, cut or extend paths.

Thus we looked at another method of defining paths and found it with the coinductive lists developed by Andreas Lochbihler in [Loc10]. To explain coinductive lists, let us first look at the standard list definition (for finite lists) in Isabelle/HOL, omitting a few non-essential parts:

```
datatype 'a List = Nil | Cons 'a "'a List"
```

This defines the algebraic datatype *List* inductively:

1. *Nil* is a list.

2. If x has type $'a$ and if x' is a list, then *Cons* x x' is also a list.

Everything not constructible by these two rules is not a list.

Thus, for $'a = \mathbb{N}$, the following are lists:

Nil
Cons 1 *Nil*
Cons 42 (*Cons* 0 *Nil*).

Coinductive lists, also called lazy lists, have a nearly identical definition:

```
codatatype 'a LList = LNil | LCons 'a "'a LList"
```

The only difference besides different identifiers is the use of **codatatype** instead of **datatype**. With **codatatype**, the type contains all the elements that *List* contains, but also infinitely nested elements such as

Cons 0 (*Cons* 1 (*Cons* 2 (*Cons* 3 (...)))).

Formally, the universe is defined as all things that can be *deconstructed* with the two rules given in the definition:

1. *LNil* is a lazy list.

2. If $x' \neq LNil$ is a lazy list, then there exists an x of type $'a$ and a lazy list y' with $x' = LCons\ x\ y'$.

Two lazy lists are equivalent if there exists a bisimulation relating them. Formally, for two lazy lists x', y' to be the same there must exist a relation R such that

1. $R(x', y')$ holds.
2. For all lazy lists x', y': If $R(x', y')$ holds, then
   a) $x' = y' = \mathit{LNil}$ or
   b) There exists an x with $x' = \mathit{LCons}\ x\ x''$ and $y' = \mathit{LCons}\ x\ y''$ and $R(x'', y'')$.

Proof schemas like the above are also called *coinductive proofs*.

The idea is that two lists are equivalent if and only if they are observationally equivalent, which means that all finite prefixes are equal. It follows that lazy lists allow indexing by natural numbers, that is, every element of a lazy list is behind a *finite* nesting of *LCons* constructors. There cannot be elements beyond this, at some ordinal index greater or equal than ω, because every such list would be equivalent to a list without these unreachable elements.

This is what allows us to use coinductive lists to represent paths in parity games: A coinductive list can represent a finite or infinite, but still $\mathbb{N}$-indexed, sequence of vertices.

For the sake of completeness we should add that technically, the above notion of equivalence is a consequence of the implementation and not a definition. The conclusion that lazy lists are $\mathbb{N}$-indexed is also not a conclusion, but part of the internal construction of codatatypes in Isabelle/HOL, which is based on so-called *bounded natural functors*. The "bounded" part refers to a cardinal bound, which in the case of lazy lists is $\aleph_0$, the

cardinality of the natural numbers. See [Bla+14] for the implementation and [TPB12] for the category-theoretic background of the internal construction of codatatypes in Isabelle/HOL.

The great thing about **codatatype** is that it allows a unified handling of finite and infinite elements by defining coinductive predicates. Take for example the definition of a valid path:

```
coinductive valid_path :: "'a Path ⇒ bool" where
  "valid_path LNil"
| "v ∈ V ⟹ valid_path (LCons v LNil)"
| "⟦ v ∈ V; w ∈ V; v→w; valid_path P;
     ¬lnull P; lhd P = w ⟧
   ⟹ valid_path (LCons v P)"
```

We want to declare as valid all the paths that walk along vertices and arcs, corresponding to Definition 2.3 on page 13. The first case is that the empty path *LNil* is a valid path. The second case is that if $v \in V$, then the path consisting of only the vertex v is valid. For the last case, if $v, w \in V$ and $(v, w) \in E$ (denoted as $v \rightarrow w$ in Isar), and if P is a non-empty valid path whose first element is w, then *LCons* v P is also a valid path.

Similarly, we define maximal and σ-conforming paths. With coinductive lists, these definitions become fairly straight-forward, and it turns out that the proofs are far simpler compared to using functions $f : \mathbb{N} \rightarrow V$ as path representations.

To further simplify proofs about *valid_path*, we prove the following coinduction schema, where Q is an arbitrary predicate on paths.

```
lemma valid_path_coinduct:
  assumes major: "Q P"
    and base: "⋀v P. Q (LCons v LNil) ⟹ v ∈ V"
    and step: "⋀v w P. Q (LCons v (LCons w P))
        ⟹ v→w
         ∧ (Q (LCons w P) ∨ valid_path (LCons w P))"
  shows "valid_path P"
```

In English the lemma says that if the following three conditions are true (for a fixed predicate Q), then P is valid path:

1. Q holds for P.
2. For all $P = (v, \ldots)$, if Q holds for P, then $v \in V$.
3. For all $P = (v, w, \ldots)$, if Q holds for P, then $(v, w) \in E$ and Q holds for $(w, \ldots)$ or $(w, \ldots)$ is a valid path.

Observe the similarities to the definition of equivalence of coinductive lists.

7.5.3 Well-Ordered Strategies

We glossed over some detail in the informal proof of Lemma 7.2. In particular, we completely ignored the construction of *uniform* attractor and *uniform* winning strategies. More formally, let $P = (V, E, V_\diamond, \omega)$ be a parity game, $S \subseteq V$ and $i \in \{\diamond, \Box\}$.

Let g be a map from V to non-empty sets of strategies and $<$ be a well-order on the set $\bigcup g(S)$. We call $g(v)$ the set of *good* strategies on v. We want to prove the following lemma.

Lemma 7.4 *Assume that for all $v \to w$ with $v \in S$ and all $\sigma \in g(v)$ it holds that*

1. *if* $v \in V_i$ *and* $\sigma(v) = w$, *then* $\sigma \in g(w)$ *and*
2. *if* $v \notin V_i$, *then* $\sigma \in g(w)$.

Then there exists a strategy σ *for player* i *on* S *such that for every* σ*-conforming path in* S *there exists a suffix* $(v, \ldots)$ *of this path and* $\sigma_v \in g(v)$ *such that*

1. $v \in S$ *and*
2. $(v, \ldots)$ *follows* σ_v *and*
3. σ_v *is* $<$*-minimal on all* $w \in (v, \ldots)$.

The assumption of this lemma says that if σ is a good strategy on v and if we follow σ, then it keeps being a good strategy.

Proof. Let σ' be an arbitrary strategy and define σ as

$$\sigma(v) := \begin{cases} \sigma_v(v) & \text{if } v \in S \cap V_i \text{ and } \sigma_v \text{ is } <\text{-minimal in } g(v) \\ \sigma'(v) & \text{otherwise.} \end{cases}$$

By the assumption that good strategies are still good on successors it is easy to see that σ has the property that every path in S follows smaller and smaller strategies. Because our order $<$ is a well-order, it follows that eventually the path follows a constant strategy. ∎

Let us remark that the above is a more general case of Lemma 3.62 on page 56. The approximations attr_i^α of attr_i, defined in Section 3.2.3, immediately give us for every vertex $v \in \mathrm{attr}_i(A)$ a positional strategy such that every conforming play starting in v visits A. The above lemma shows that we can merge all these different strategies for different v into one uniform strategy for $\mathrm{attr}_i(A)$.

We formalize this statement as follows.

```
locale WellOrderedStrategies = ParityGame +
  fixes S :: "'a set"
    and p :: Player
    — The set of good strategies on a vertex v
    and good :: "'a ⇒ 'a Strategy set"
    and r :: "('a Strategy × 'a Strategy) set"
  assumes S_V: "S ⊆ V"
    — r is a wellorder on the set of all strategies
    — which are good somewhere.
    and r_wo:
        "well_order_on {σ. ∃v ∈ S. σ ∈ good v} r"
    — Every vertex has a good strategy.
    and good_ex: "⋀v. v ∈ S ⟹ ∃σ. σ ∈ good v"
    — Good strategies are well-formed strategies.
    and good_strategies:
        "⋀v σ. σ ∈ good v ⟹ strategy p σ"
    — A good strategy on v is also good on
    — possible successors of v.
    and strategies_continue:
        "⋀v w σ. ⟦ v ∈ S; v→w;
          v ∈ VV p ⟹ σ v = w; σ ∈ good v ⟧
        ⟹ σ ∈ good w"
```

For a given vertex v, we define a predicate *minimal_good_strategy* v on strategies that tells us if a strategy is good and minimal. Formally,

```
definition "minimal_good_strategy v σ ≡
    σ ∈ good v
     ∧ (∀σ'. (σ', σ) ∈ r - Id ⟶ σ' ∉ good v)"
```

Here $r - Id$ is the well-order without the reflexive pairs. This is necessary because in Isabelle well-orders are reflexive.

Now for every $v \in S$, we can select the minimum strategy from all good strategies of v:

```
definition "choose v ≡
    THE σ. minimal_good_strategy v σ"
```

THE The *THE* operator is the definite description operator in Isabelle/HOL and can only be usefully applied to predicates that are true for exactly one element. If f is such a predicate, then the term

$$THE\ x.\ f\ x$$

denotes the unique x such that $f\ x$ is true.

This lets us finally define the strategy we are after, the strategy that uses the minimum strategy on all vertices of S.

```
definition "well_ordered_strategy ≡
    override_on σ_arbitrary (λv. choose v v) S"
```

Here *override_on f g S* overrides the function f with g on the set S. Like in the informal proof above, this defines σ via a case distinction: If $v \in S$, we use the minimal strategy of $g(v)$, and for $v \notin S$, we use an arbitrary (but fixed) strategy.

Finally, we show that under these assumptions and definitions, every path that stays in S eventually follows a constant strategy.

We use this method for two theorems: First, we show that every attractor has a uniform attractor strategy. Second, we

show that the winning region of player i

$$W_i = \{v \in V \mid \text{there exists a strategy } \sigma \text{ for } i \text{ that is winning from } v\}$$

has a uniform strategy σ that is winning on all $v \in W_i$.

7.6 Conclusions

We presented a formal proof of positional determinacy for parity games. Hopefully, this proof will provide the foundation for other formal proofs about parity games.

Possible future work would be, for example, to consider parity games with infinitely many priorities. Surprisingly, such games are positionally determined under certain conditions, as shown by Erich Grädel and Igor Walukiewicz [GW06]. A formalization of this result would extend ours.

Another interesting project would a formalization of the modal μ-calculus, proving for example bisimulation invariance, or even deeper theorems such as the strictness of the alternation hierarchy, the finite model theorem or the completeness of an axiomatization.

Furthermore, assuming we have a basic formalization of the modal μ-calculus, proving the equivalence between modal μ-calculus model checking and parity games seems well within reach.

7.6.1 Time Complexity

However, there is one aspect that seems to be more difficult. The equivalence between modal μ-calculus model checking and

parity games is not only a mathematical equivalence between satisfied formulas and winning players, but also a *polynomial-time* equivalence (Corollary 4.8 on page 83).

We expect that proving polynomial-time equivalence between the L_μ model-checking problem and parity games will be much harder because, as far as the author is aware, there is no general framework in the Isabelle community for proving time bounds of algorithms. The usual way of formally proving time complexity is to add a time counter to the measured function, counting for example the number of recursive calls and the number of arithmetic operations, and then bounding the value of this counter.

However, mathematically this is hardly satisfying because this is an ad-hoc machine model. The proof crucially depends on the human to make sure that the time counter really counts all operations that should be counted. This mirrors English language proofs where proving time bounds usually involves a pseudo-code implementation with costs applied to certain operations in a convincing, but due to the nature of pseudo-code, ultimately informal way.

7.6.2 Turing Machines

Traditionally, complexity classes are introduced with a fixed machine model such as *Turing machines*. We think that ideally, a formalization for reasoning about time complexity should be grounded in standard complexity theory by ultimately expressing everything in terms Turing machines. This is a daunting task. It is so daunting that the author is unaware of *any* formalization of even the basics of standard complexity theory.

The current state of the art, as of 2016, is a formalization

of multi-tape Turing machines by Andrea Asperti and Wilmer Ricciotti [AR15] in the Matita theorem prover [Mat], which is similar to Coq. They formalize a multi-tape universal Turing machine simulating a mono-tape normal[2] machine and prove its correctness, but they have not yet proved its complexity nor generalized it to simulating multi-tape machines. They observe that working with Turing machines is fiddly, and rightly state as their long-term goal the formalization of complexity theory "at a convenient level of abstraction" removed from the gritty details of states and transition functions.

Two years earlier, Jian Xu, Xingyuan Zhang and Christian Urban [XZU13] came up with a different formalization of Turing machines. However, Asperti and Ricciotti criticize their formalization as too complex and as not compositional enough, and thus hard to work with [AR15]. Moreover, the machines in Xu, Zhang and Urban's formalization are not efficient enough to be useful for developing complexity theory, although they are perfectly suitable for computability theory (which was, to be fair, the authors' intention).

However, Turing machines are still very removed from how people practically prove time bounds for their algorithms. The practical way is a pseudo-code implementation with associated costs for each instruction. For formal proofs, this is a problem because obviously, there is no formal specification of pseudo-code or we would call it "code".

So a formal specification of pseudo-code would end up as

[2] A *normal* Turing machine has the alphabet $\{0, 1\}$, the set of states $\{0, 1, \ldots, k\}$ and the initial state 0. It is a standard result from complexity theory that every Turing machine with alphabet Γ running in time $T(n)$ can be simulated by a normal machine running in time $O(\log |\Gamma| T(n))$ [AB09, Claim 1.5].

a formal specification of the semantics of some programming language. This alone is a major project, as evidenced by, for example, the recent formalization of the specification of ECMAScript 5, more commonly known as JavaScript [Bod+14; Jsc]. This specification measures more than 23,000 lines of code written for the Coq proof assistant.

But even such a formalized specification is insufficient: We need a formally verified translation of programs into equivalent Turing machines if we want to stay within the traditional framework of complexity theory. The scope of these problems together is immense and makes it unlikely to see a formalization of traditional complexity theory in the near future.

7.6.3 Restricted Graph Classes

Another aspect of parity games is that they can be efficiently solved on many restricted classes of graphs, for example on classes of bounded treewidth, bounded DAG-width, and more (see Section 4.1.2 on page 77). In order to formalize correctness or complexity proofs for these algorithms, we also need formalizations of these restricted classes of graphs.

As of 2016 there is very little progress in this area, either. One starting point, but also a work in progress, is a recent formalization of tree decompositions by the author [Dit16]. We are not aware of any other attempts of formalizing graphs of bounded treewidth, bounded clique width or graphs bounded by any of the directed width measures.

Bibliography

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity – A Modern Approach*. Cambridge University Press, 2009. ISBN: 978-0-521-42426-4. URL: http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264.

[AH76] Kenneth Appel and Wolfgang Haken. "Every planar map is four colorable". In: *Bulletin of the American Mathematical Society* 82 (1976), pp. 711–712. DOI: 10.1090/S0002-9904-1976-14122-5.

[AKR14] Saeed Akhoondian Amiri, Stephan Kreutzer, and Roman Rabinovich. "DAG-width is PSPACE-complete". In: *CoRR* (2014). arXiv: 1411.2438.

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. "PRIMES is in P". In: *Annals of Mathematics* 160.2 (2004), pp. 781–793. DOI: 10.4007/annals.2004.160.781.

[AR05] Dorit Aharonov and Oded Regev. "Lattice problems in NP ∩ coNP". In: *J. ACM* 52.5 (2005), pp. 749–765. DOI: 10.1145/1089023.1089025.

[AR15] Andrea Asperti and Wilmer Ricciotti. "A formalization of multi-tape Turing machines". In: *Theor. Comput. Sci.* 603 (2015), pp. 23–42. DOI: 10.1016/j.tcs.2015.07.013.

[Bab92] László Babai. “Bounded Round Interactive Proofs in Finite Groups”. In: *SIAM J. Discrete Math.* 5.1 (1992), pp. 88–111. DOI: 10.1137/0405008.

[BB07] Patrick Blackburn and Johan van Benthem. “Modal logic: a semantic perspective”. In: *Handbook of Modal Logic*. Ed. by Johan Van Benthem Patrick Blackburn and Frank Wolter. Vol. 3. Studies in Logic and Practical Reasoning. Elsevier, 2007, pp. 1–84. DOI: 10.1016/S1570-2464(07)80004-8.

[BC96] Girish Bhat and Rance Cleaveland. “Efficient Model Checking via the Equational μ-Calculus”. In: *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996*. IEEE Computer Society, 1996, pp. 304–312. ISBN: 0-8186-7463-6. DOI: 10.1109/LICS.1996.561358. URL: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4265.

[BDK14] Mikołaj Bojańczyk, Christoph Dittmann, and Stephan Kreutzer. “Decomposition theorems and model-checking for the modal μ-calculus”. In: *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*. Ed. by Thomas A. Henzinger and Dale Miller. ACM, 2014, 17:1–17:10. ISBN: 978-1-4503-2886-9. DOI: 10.1145/2603088.2603144. URL: http://dl.acm.org/citation.cfm?id=2603088.

[Ben84] Johan van Benthem. "Correspondence Theory". In: *Handbook of Philosophical Logic: Volume II: Extensions of Classical Logic*. Ed. by D. Gabbay and F. Guenthner. Dordrecht: Springer Netherlands, 1984, pp. 167–247. ISBN: 978-94-009-6259-0. DOI: 10.1007/978-94-009-6259-0_4.

[Ber+06] Dietmar Berwanger et al. "DAG-Width and Parity Games". In: *STACS*. Ed. by Bruno Durand and Wolfgang Thomas. Vol. 3884. Lecture Notes in Computer Science. Springer, 2006, pp. 524–536. ISBN: 3-540-32301-5. DOI: 10.1007/11672142_43.

[Ber+10] Dietmar Berwanger et al. "Strategy construction for parity games with imperfect information". In: *Inf. Comput.* 208.10 (2010), pp. 1206–1220. DOI: 10.1016/j.ic.2009.09.006.

[Ber+12] Dietmar Berwanger et al. "The DAG-width of directed graphs". In: *J. Comb. Theory, Ser. B* 102.4 (2012), pp. 900–923. DOI: 10.1016/j.jctb.2012.04.004.

[BFL15] Florian Bruse, Oliver Friedmann, and Martin Lange. "On guarded transformation in the modal μ-calculus". In: *Logic Journal of the IGPL* 23.2 (2015), pp. 194–216. DOI: 10.1093/jigpal/jzu030.

[BG04] Dietmar Berwanger and Erich Grädel. "Entanglement – A Measure for the Complexity of Directed Graphs with Applications to Logic and Games". In: *LPAR*. Vol. 3452. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2004, pp. 209–

223. ISBN: 978-3-540-25236-8. DOI: 10.1007/978-3-540-32275-7_15.

[BJG09] Jørgen Bang-Jensen and Gregory Gutin. *Digraphs: Theory, Algorithms and Applications*. Second. Monographs in Mathematics. Springer, London, 2009. ISBN: 978-1-84800-997-4. DOI: 10.1007/978-1-84800-998-1.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008. ISBN: 978-0-262-02649-9.

[Bla+14] Jasmin Christian Blanchette et al. "Truly Modular (Co)datatypes for Isabelle/HOL". In: *Interactive Theorem Proving - 5th International Conference, ITP 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*. Ed. by Gerwin Klein and Ruben Gamboa. Vol. 8558. Lecture Notes in Computer Science. Springer, 2014, pp. 93–110. ISBN: 978-3-319-08969-0. DOI: 10.1007/978-3-319-08970-6_7.

[Bla+15] Jasmin Christian Blanchette et al. "Mining the Archive of Formal Proofs". In: *Intelligent Computer Mathematics - International Conference, CICM 2015, Washington, DC, USA, July 13-17, 2015, Proceedings*. Ed. by Manfred Kerber et al. Vol. 9150. Lecture Notes in Computer Science. Springer, 2015, pp. 3–17. ISBN: 978-3-319-20614-1. DOI: 10.1007/978-3-319-20615-8_1.

[BLS99] Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes: A Survey*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1999. ISBN: 978-0-89871-432-6. DOI: 10.1137/1.9780898719796.

[BM88] László Babai and Shlomo Moran. "Arthur-Merlin Games: A Randomized Proof System, and a Hierarchy of Complexity Classes". In: *J. Comput. Syst. Sci.* 36.2 (1988), pp. 254–276. DOI: 10.1016/0022-0000(88)90028-1.

[Bod+14] Martin Bodin et al. "A Trusted Mechanised JavaScript Specification". In: *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '14. New York, NY, USA: ACM, 2014, pp. 87–100. ISBN: 978-1-4503-2544-8. DOI: 10.1145/2535838.2535876.

[BS07] Julian Bradfield and Colin Stirling. "Modal mu-calculi". In: *Handbook of Modal Logic*. Ed. by Johan Van Benthem Patrick Blackburn and Frank Wolter. Vol. 3. Studies in Logic and Practical Reasoning. Elsevier, 2007, pp. 721–756. DOI: 10.1016/S1570-2464(07)80015-2.

[BS12] Dietmar Berwanger and Olivier Serre. "Parity games on undirected graphs". In: *Inf. Process. Lett.* 112.23 (2012), pp. 928–932. DOI: 10.1016/j.ipl.2012.08.021.

[BSV03] Henrik Björklund, Sven Sandberg, and Sergei G. Vorobyov. "A Discrete Subexponential Algorithm for Parity Games". In: *STACS 2003, 20th Annual*

Symposium on Theoretical Aspects of Computer Science, Berlin, Germany, February 27 - March 1, 2003, Proceedings. Ed. by Helmut Alt and Michel Habib. Vol. 2607. Lecture Notes in Computer Science. Springer, 2003, pp. 663–674. ISBN: 3-540-00623-0. DOI: 10.1007/3-540-36494-3_58.

[BSV04] Henrik Björklund, Sven Sandberg, and Sergei G. Vorobyov. "Memoryless determinacy of parity and mean payoff games: a simple proof". In: *Theor. Comput. Sci.* 310.1-3 (2004), pp. 365–378. DOI: 10.1016/S0304-3975(03)00427-4.

[CGR11] Sjoerd Cranen, Jan Friso Groote, and Michel A. Reniers. "A linear translation from CTL* to the first-order modal μ-calculus". In: *Theor. Comput. Sci.* 412.28 (2011), pp. 3129–3139. DOI: 10.1016/j.tcs.2011.02.034.

[Chu40] Alonzo Church. "A Formulation of the Simple Theory of Types". In: *J. Symb. Log.* 5.2 (1940), pp. 56–68. DOI: 10.2307/2266170.

[Con92] Anne Condon. "The Complexity of Stochastic Games". In: *Inf. Comput.* 96.2 (1992), pp. 203–224. DOI: 10.1016/0890-5401(92)90048-K.

[Dam94] Mads Dam. "CTL* and ECTL* as Fragments of the Modal mu-Calculus". In: *Theor. Comput. Sci.* 126.1 (1994), pp. 77–96. DOI: 10.1016/0304-3975(94)90269-0.

[Dev93] Keith Devlin. *The Joy of Sets: Fundamentals of Contemporary Set Theory*. Undergraduate Texts in Mathematics. Springer, New York, 1993. ISBN:

978-1-4612-6941-0. DOI: 10.1007/978-1-4612-0903-4.

[DG08] Anuj Dawar and Erich Grädel. "The Descriptive Complexity of Parity Games". In: *CSL*. Ed. by Michael Kaminski and Simone Martini. Vol. 5213. Lecture Notes in Computer Science. Springer, 2008, pp. 354–368. ISBN: 978-3-540-87530-7. DOI: 10.1007/978-3-540-87531-4_26.

[DH04] Erik D. Demaine and Mohammad Taghi Hajiaghayi. "Equivalence of local treewidth and linear local treewidth and its algorithmic applications". In: *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*. Ed. by J. Ian Munro. SIAM, 2004, pp. 840–849. ISBN: 0-89871-558-X. URL: http://dl.acm.org/citation.cfm?id=982792.982919.

[Die10] Reinhard Diestel. *Graph Theory, 4th Edition*. Vol. 173. Graduate texts in mathematics. Springer, 2010. ISBN: 978-3-642-14278-9.

[Dit15] Christoph Dittmann. "Positional Determinacy of Parity Games". In: *Archive of Formal Proofs* (Nov. 2015). http://www.isa-afp.org/entries/Parity_Game.shtml, Formal proof development. ISSN: 2150-914x.

[Dit16] Christoph Dittmann. "Tree Decomposition". In: *Archive of Formal Proofs* (May 2016). http://isa-afp.org/entries/Tree_Decomposition.shtml, Formal proof development. ISSN: 2150-914x.

[DKT13] Zdeněk Dvořák, Daniel Král', and Robin Thomas. "Testing first-order properties for subclasses of sparse graphs". In: *J. ACM* 60.5 (2013), p. 36. DOI: 10.1145/2499483.

[DKT16] Christoph Dittmann, Stephan Kreutzer, and Alexandru I. Tomescu. "Graph operations on parity games and polynomial-time algorithms". In: *Theor. Comput. Sci.* 614 (2016), pp. 97–108. DOI: 10.1016/j.tcs.2015.11.044.

[EC80] E. Allen Emerson and Edmund M. Clarke. "Characterizing Correctness Properties of Parallel Programs Using Fixpoints". In: *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherland, July 14-18, 1980, Proceedings*. Ed. by J. W. de Bakker and Jan van Leeuwen. Vol. 85. Lecture Notes in Computer Science. Springer, 1980, pp. 169–181. ISBN: 3-540-10003-2. DOI: 10.1007/3-540-10003-2_69.

[EF99] Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite model theory*. Perspectives in Mathematical Logic. Springer, 1999. ISBN: 978-3-540-65758-3.

[EL86] E. Allen Emerson and Chin-Laung Lei. "Efficient Model Checking in Fragments of the Propositional Mu-Calculus (Extended Abstract)". In: *Proceedings of the Symposium on Logic in Computer Science (LICS '86), Cambridge, Massachusetts, USA, June 16-18, 1986*. IEEE Computer Society, 1986, pp. 267–278. ISBN: 0-8186-0720-3.

[Eve97] Howard Eves. *Foundations and Fundamental Concepts of Mathematics*. Dover Books on Mathematics. Dover Publications, Inc., 1997. ISBN: 978-0486696096.

[FG06] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. ISBN: 3540299521.

[FL09] Oliver Friedmann and Martin Lange. "Solving Parity Games in Practice". In: *Automated Technology for Verification and Analysis*. Ed. by Zhiming Liu and Anders Ravn. Vol. 5799. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2009, pp. 182–196. ISBN: 978-3-642-04760-2. DOI: 10.1007/978-3-642-04761-9_15.

[Fri09] Oliver Friedmann. "An Exponential Lower Bound for the Parity Game Strategy Improvement Algorithm as We Know it". In: *LICS*. IEEE Computer Society, 2009, pp. 145–156. ISBN: 978-0-7695-3746-7. DOI: 10.1109/LICS.2009.27.

[Fri10] Oliver Friedmann. "The Stevens-Stirling-Algorithm for Solving Parity Games Locally Requires Exponential Time". In: *Int. J. Found. Comput. Sci.* 21.3 (2010), pp. 277–287. DOI: 10.1142/S0129054110007246.

[Fri11] Oliver Friedmann. "An Exponential Lower Bound for the Latest Deterministic Strategy Iteration Algorithms". In: *Logical Methods in Computer Science* 7.3 (2011). DOI: 10.2168/LMCS-7(3:23)2011.

[FS12] John Fearnley and Sven Schewe. "Time and Parallelizability Results for Parity Games with Bounded Treewidth". In: *ICALP (2)*. Ed. by Artur Czumaj et al. Vol. 7392. Lecture Notes in Computer Science. Springer, 2012, pp. 189–200. ISBN: 978-3-642-31584-8. DOI: 10.1007/978-3-642-31585-5_20.

[FV59] Solomon Feferman and Robert L. Vaught. "The first-order properties of algebraic systems". In: *Fundamenta Mathematicae* 47 (1959), pp. 57–103.

[Gaj+15] Jakub Gajarský et al. "Parameterized Algorithms for Parity Games". In: *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II*. Ed. by Giuseppe F. Italiano, Giovanni Pighizzini, and Donald Sannella. Vol. 9235. Lecture Notes in Computer Science. Springer, 2015, pp. 336–347. ISBN: 978-3-662-48053-3. DOI: 10.1007/978-3-662-48054-0_28.

[Gan15] Moses Ganardi. "Parity Games of Bounded Tree- and Clique-Width". In: *Foundations of Software Science and Computation Structures - 18th International Conference, FoSSaCS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*. Ed. by Andrew M. Pitts. Vol. 9034. Lecture Notes in Computer Science. Springer, 2015, pp. 390–404. ISBN: 978-3-662-46677-3. DOI: 10.1007/978-3-662-46678-0_25.

[GKS14] Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. "Deciding first-order properties of nowhere dense graphs". In: *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*. Ed. by David B. Shmoys. ACM, 2014, pp. 89–98. ISBN: 978-1-4503-2710-7. DOI: 10.1145/2591796.2591851. URL: http://dl.acm.org/citation.cfm?id=2591796.

[Gon07] Georges Gonthier. "The Four Colour Theorem: Engineering of a Formal Proof". In: *Computer Mathematics, 8th Asian Symposium, ASCM 2007, Singapore, December 15-17, 2007. Revised and Invited Papers*. Ed. by Deepak Kapur. Vol. 5081. Lecture Notes in Computer Science. Springer, 2007, p. 333. ISBN: 978-3-540-87826-1. DOI: 10.1007/978-3-540-87827-8_28.

[Gon08] Georges Gonthier. "Formal Proof—The Four-Color Theorem". In: *Notices of the AMS* 55.11 (2008), pp. 1382–1393. URL: http://www.ams.org/notices/200811/tx081101382p.pdf.

[Grä+07] Erich Grädel et al. *Finite Model Theory and Its Applications*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2007. ISBN: 978-3-540-00428-8. DOI: 10.1007/3-540-68804-8.

[GTW02] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, eds. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*. Vol. 2500. Lecture Notes

in Computer Science. Springer, 2002. ISBN: 3-540-00388-6.

[GW06] Erich Grädel and Igor Walukiewicz. "Positional Determinacy of Games with Infinitely Many Priorities". In: *Logical Methods in Computer Science* 2.4 (2006). DOI: 10.2168/LMCS-2(4:6)2006.

[Hel+12] Keijo Heljanko et al. "Solving parity games by a reduction to SAT". In: *J. Comput. Syst. Sci.* 78.2 (2012), pp. 430–440. DOI: 10.1016/j.jcss.2011.05.004.

[Hil22] David Hilbert. *Neubegründung der Mathematik: Erste Mitteilung*. Printed in [Hil35]. 1922. URL: http://resolver.sub.uni-goettingen.de/purl?PPN237834022.

[Hil35] David Hilbert. *Gesammelte Abhandlungen*. Vol. 3. Berlin: Springer, 1935. URL: http://resolver.sub.uni-goettingen.de/purl?PPN237820250.

[HK08] Paul Hunter and Stephan Kreutzer. "Digraph measures: Kelly decompositions, games, and orderings". In: *Theor. Comput. Sci.* 399.3 (2008), pp. 206–219. DOI: 10.1016/j.tcs.2008.02.038.

[Hol93] Gerard J. Holzmann. "Design and Validation of Protocols: A Tutorial". In: *Computer Networks and ISDN Systems* 25.9 (1993), pp. 981–1017. DOI: 10.1016/0169-7552(93)90095-L.

[HT73] John E. Hopcroft and Robert Endre Tarjan. "Efficient Algorithms for Graph Manipulation [H] (Algorithm 447)". In: *Commun. ACM* 16.6 (1973), pp. 372–378. DOI: 10.1145/362248.362272.

[Joh+01] Thor Johnson et al. “Directed tree-width”. In: *J. Comb. Theory Ser. B* 82.1 (May 2001), pp. 138–154. ISSN: 0095-8956. DOI: 10.1006/jctb.2000.2031.

[JPZ06] Marcin Jurdziński, Mike Paterson, and Uri Zwick. “A deterministic subexponential algorithm for solving parity games”. In: *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*. SODA ’06. New York, NY, USA: ACM, 2006, pp. 117–123. ISBN: 0-89871-605-5. DOI: 10.1145/1109557.1109571.

[Jsc] *JSCert*. https://github.com/jscert/jscert, Commit 8e1b8a39. 2015.

[Jur00] Marcin Jurdziński. “Small Progress Measures for Solving Parity Games”. In: *STACS 2000*. Ed. by Horst Reichel and Sophie Tison. Vol. 1770. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2000, pp. 290–301. ISBN: 978-3-540-67141-1. DOI: 10.1007/3-540-46541-3_24.

[Jur98] Marcin Jurdziński. “Deciding the Winner in Parity Games is in UP ∩ co-UP”. In: *Inf. Process. Lett.* 68.3 (1998), pp. 119–124. DOI: 10.1016/S0020-0190(98)00150-1.

[JW96] David Janin and Igor Walukiewicz. “On the Expressive Completeness of the Propositional mu-Calculus with Respect to Monadic Second Order Logic”. In: *CONCUR*. Ed. by Ugo Montanari and Vladimiro Sassone. Vol. 1119. Lecture Notes in Computer Science. Springer, 1996, pp. 263–277. ISBN: 3-540-61604-7. DOI: 10.1007/3-540-61604-7_60.

[Kal92] Gil Kalai. “A Subexponential Randomized Simplex Algorithm (Extended Abstract)”. In: *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*. Ed. by S. Rao Kosaraju et al. ACM, 1992, pp. 475–482. ISBN: 0-89791-511-9. DOI: 10.1145/129712.129759.

[KD09] Stephan Kreutzer and Anuj Dawar. “Parameterized Complexity of First-Order Logic”. In: *Electronic Colloquium on Computational Complexity (ECCC)* 16 (2009), p. 131. URL: http://eccc.hpi-web.de/report/2009/131.

[Klo94] Ton Kloks. *Treewidth, Computations and Approximations*. Vol. 842. Lecture Notes in Computer Science. Springer, 1994. ISBN: 3-540-58356-4. DOI: 10.1007/BFb0045375.

[KM08] Michael Kaminski and Simone Martini, eds. *Computer Science Logic, 22nd International Workshop, CSL 2008, 17th Annual Conference of the EACSL, Bertinoro, Italy, September 16-19, 2008. Proceedings*. Vol. 5213. Lecture Notes in Computer Science. Springer, 2008. ISBN: 978-3-540-87530-7.

[Koz83] Dexter Kozen. “Results on the propositional μ-calculus”. In: *Theoretical Computer Science* 27.3 (1983). Special Issue Ninth International Colloquium on Automata, Languages and Programming (ICALP) Aarhus, Summer 1982, pp. 333–354. ISSN: 0304-3975. DOI: 10.1016/0304-3975(82)90125-6.

[Kre15] Stephan Kreutzer. *Logik, Spiele und Automaten*. `http://logic.las.tu-berlin.de/Teaching/index.html`. Unpublished lecture notes for a master's course on mathematical logic and games at Technische Universität Berlin. 2015.

[Küs01] Ralf Küsters. "Memoryless Determinacy of Parity Games". In: *Automata, Logics, and Infinite Games*. Ed. by Erich Grädel, Wolfgang Thomas, and Thomas Wilke. Vol. 2500. Lecture Notes in Computer Science. Springer, 2001, pp. 95–106. ISBN: 3-540-00388-6. DOI: `10.1007/3-540-36387-4_6`.

[Loc10] Andreas Lochbihler. "Coinductive". In: *Archive of Formal Proofs* (Feb. 2010). `http://www.isa-afp.org/entries/Coinductive.shtml`, Formal proof development. ISSN: 2150-914x.

[Lon+94] David E. Long et al. "An Improved Algorithm for the Evaluation of Fixpoint Expressions". In: *Computer Aided Verification, 6th International Conference, CAV '94, Stanford, California, USA, June 21-23, 1994, Proceedings*. Ed. by David L. Dill. Vol. 818. Lecture Notes in Computer Science. Springer, 1994, pp. 338–350. ISBN: 3-540-58179-0. DOI: `10.1007/3-540-58179-0_66`.

[Lud95] Walter Ludwig. "A Subexponential Randomized Algorithm for the Simple Stochastic Game Problem". In: *Inf. Comput.* 117.1 (1995), pp. 151–155. DOI: `10.1006/inco.1995.1035`.

[Mak04] Johann A. Makowsky. "Algorithmic uses of the Feferman-Vaught Theorem". In: *Ann. Pure Appl.*

Logic 126.1-3 (2004), pp. 159–213. DOI: 10.1016/j.apal.2003.11.002.

[Mat] *Matita Theorem Prover*. http://matita.cs.unibo.it/. Last accessed June 2016.

[McN93] Robert McNaughton. "Infinite Games Played on Finite Graphs". In: *Ann. Pure Appl. Logic* 65.2 (1993), pp. 149–184. DOI: 10.1016/0168-0072(93)90036-D.

[MRR16] Matthias Mnich, Heiko Röglin, and Clemens Rösner. "New Deterministic Algorithms for Solving Parity Games". In: *LATIN 2016: Theoretical Informatics - 12th Latin American Symposium, Ensenada, Mexico, April 11-15, 2016, Proceedings*. Ed. by Evangelos Kranakis, Gonzalo Navarro, and Edgar Chávez. Vol. 9644. Lecture Notes in Computer Science. Springer, 2016, pp. 634–645. ISBN: 978-3-662-49528-5. DOI: 10.1007/978-3-662-49529-2_47.

[MS06] Nicolas Markey and Philippe Schnoebelen. "Mu-calculus path checking". In: *Inf. Process. Lett.* 97.6 (2006), pp. 225–230. DOI: 10.1016/j.ipl.2005.11.010.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*. Vol. 2283. Lecture Notes in Computer Science. Springer, 2002. ISBN: 3-540-43376-7. DOI: 10.1007/3-540-45949-9.

[NPW16] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*. Part of the Isabelle documentation,

http://isabelle.in.tum.de/dist/Isabelle2016/doc/tutorial.pdf. Feb. 2016.

[Nus] *NuSMV: a new symbolic model checker*. http://nusmv.fbk.eu/. Last accessed June 2016.

[Obd03] Jan Obdržálek. "Fast Mu-Calculus Model Checking when Tree-Width Is Bounded". In: *Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings*. Ed. by Warren A. Hunt Jr. and Fabio Somenzi. Vol. 2725. Lecture Notes in Computer Science. Springer, 2003, pp. 80–92. ISBN: 3-540-40524-0. DOI: 10.1007/978-3-540-45069-6_7.

[Obd07] Jan Obdržálek. "Clique-Width and Parity Games". In: *CSL*. Ed. by Jacques Duparc and Thomas A. Henzinger. Vol. 4646. Lecture Notes in Computer Science. Springer, 2007, pp. 54–68. ISBN: 978-3-540-74914-1. DOI: 10.1007/978-3-540-74915-8_8.

[Pau11] Christine Paulin-Mohring. "Introduction to the Coq Proof-Assistant for Practical Software Verification". In: *Tools for Practical Software Verification, LASER, International Summer School 2011, Elba Island, Italy, Revised Tutorial Lectures*. Ed. by Bertrand Meyer and Martin Nordio. Vol. 7682. Lecture Notes in Computer Science. Springer, 2011, pp. 45–95. ISBN: 978-3-642-35745-9. DOI: 10.1007/978-3-642-35746-6_3.

[Pgs] *PGSolver: A collection of tools for generating, manipulating and – most of all – solving parity games.*

https://github.com/tcsprojects/pgsolver, Commit e651e609. 2016.

[QS82] Jean-Pierre Queille and Joseph Sifakis. "Specification and verification of concurrent systems in CESAR". In: *International Symposium on Programming, 5th Colloquium, Torino, Italy, April 6-8, 1982, Proceedings*. Ed. by Mariangiola Dezani-Ciancaglini and Ugo Montanari. Vol. 137. Lecture Notes in Computer Science. Springer, 1982, pp. 337–351. ISBN: 3-540-11494-7. DOI: 10.1007/3-540-11494-7_22.

[RS84] Neil Robertson and Paul D. Seymour. "Graph minors. III. Planar tree-width". In: *J. Comb. Theory, Ser. B* 36.1 (1984), pp. 49–64. DOI: 10.1016/0095-8956(84)90013-3.

[RS86] Neil Robertson and Paul D. Seymour. "Graph Minors. II. Algorithmic Aspects of Tree-Width". In: *J. Algorithms* 7.3 (1986), pp. 309–322. DOI: 10.1016/0196-6774(86)90023-4.

[Saf05] Mohammad Ali Safari. "D-Width: A More Natural Measure for Directed Tree Width". In: *MFCS*. Ed. by Joanna Jedrzejowicz and Andrzej Szepietowski. Vol. 3618. Lecture Notes in Computer Science. Springer, 2005, pp. 745–756. ISBN: 3-540-28702-7. DOI: 10.1007/11549345_64.

[Sch07] Sven Schewe. "Solving Parity Games in Big Steps". In: *FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science*. Ed. by V. Arvind and Sanjiva Prasad. Vol. 4855. Lecture

Notes in Computer Science. Springer Berlin / Heidelberg, 2007, pp. 449–460. ISBN: 978-3-540-77049-7. DOI: 10.1007/978-3-540-77050-3_37.

[Sch08] Sven Schewe. "An Optimal Strategy Improvement Algorithm for Solving Parity and Payoff Games". In: *Computer Science Logic, 22nd International Workshop, CSL 2008, 17th Annual Conference of the EACSL, Bertinoro, Italy, September 16-19, 2008. Proceedings*. Ed. by Michael Kaminski and Simone Martini. Vol. 5213. Lecture Notes in Computer Science. Springer, 2008, pp. 369–384. ISBN: 978-3-540-87530-7. DOI: 10.1007/978-3-540-87531-4_27.

[SE89] Robert S. Streett and E. Allen Emerson. "An Automata Theoretic Decision Procedure for the Propositional Mu-Calculus". In: *Inf. Comput.* 81.3 (1989), pp. 249–264. DOI: 10.1016/0890-5401(89)90031-X.

[Sei96] Helmut Seidl. "Fast and Simple Nested Fixpoints". In: *Inf. Process. Lett.* 59.6 (1996), pp. 303–308. DOI: 10.1016/0020-0190(96)00130-5.

[Spi] *Spin, a Verifier for Multi-threaded Software*. http://spinroot.com/. Last accessed June 2016.

[SS98] Perdita Stevens and Colin Stirling. "Practical Model-Checking Using Games". In: *Tools and Algorithms for Construction and Analysis of Systems, 4th International Conference, TACAS '98, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'98, Lisbon, Por-*

tugal, March 28 - April 4, 1998, Proceedings. Ed. by Bernhard Steffen. Vol. 1384. Lecture Notes in Computer Science. Springer, 1998, pp. 85–101. ISBN: 3-540-64356-7. DOI: 10.1007/BFb0054166.

[Tar55] Alfred Tarski. "A lattice-theoretical fixpoint theorem and its applications". In: *Pacific Journal of Mathematics* 5.2 (1955), pp. 285–309. URL: http://projecteuclid.org/euclid.pjm/1103044538.

[TPB12] Dmitriy Traytel, Andrei Popescu, and Jasmin Christian Blanchette. "Foundational, Compositional (Co)datatypes for Higher-Order Logic: Category Theory Applied to Theorem Proving". In: *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*. IEEE Computer Society, 2012, pp. 596–605. ISBN: 978-1-4673-2263-8. DOI: 10.1109/LICS.2012.75.

[VJ00] Jens Vöge and Marcin Jurdziński. "A Discrete Strategy Improvement Algorithm for Solving Parity Games". In: *Computer Aided Verification*. Ed. by E. Emerson and Aravinda Sistla. Vol. 1855. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2000, pp. 202–215. ISBN: 978-3-540-67770-3. DOI: 10.1007/10722167_18.

[Wal96] Igor Walukiewicz. "Monadic Second Order Logic on Tree-Like Structures". In: *STACS 96, 13th Annual Symposium on Theoretical Aspects of Computer Science, Grenoble, France, February 22-24, 1996, Proceedings*. Ed. by Claude Puech and Rüdiger

Reischuk. Vol. 1046. Lecture Notes in Computer Science. Springer, 1996, pp. 401–413. ISBN: 3-540-60922-9. DOI: 10.1007/3-540-60922-9_33.

[WPN08] Makarius Wenzel, Lawrence C. Paulson, and Tobias Nipkow. "The Isabelle Framework". In: *Theorem Proving in Higher Order Logics, 21st International Conference, TPHOLs 2008, Montreal, Canada, August 18-21, 2008. Proceedings*. Ed. by Otmane Aït Mohamed, César A. Muñoz, and Sofiène Tahar. Vol. 5170. Lecture Notes in Computer Science. Springer, 2008, pp. 33–38. ISBN: 978-3-540-71065-3. DOI: 10.1007/978-3-540-71067-7_7.

[XZU13] Jian Xu, Xingyuan Zhang, and Christian Urban. "Mechanising Turing Machines and Computability Theory in Isabelle/HOL". In: *Interactive Theorem Proving - 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings*. Ed. by Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie. Vol. 7998. Lecture Notes in Computer Science. Springer, 2013, pp. 147–162. ISBN: 978-3-642-39633-5. DOI: 10.1007/978-3-642-39634-2_13.

[Zap01] Júlia Zappe. "Modal μ-Calculus and Alternating Tree Automata". In: *Automata, Logics, and Infinite Games*. Ed. by Erich Grädel, Wolfgang Thomas, and Thomas Wilke. Vol. 2500. Lecture Notes in Computer Science. Springer, 2001, pp. 171–184. ISBN: 3-540-00388-6. DOI: 10.1007/3-540-36387-4_10.

[Zie98] Wiesław Zielonka. “Infinite Games on Finitely Coloured Graphs with Applications to Automata on Infinite Trees”. In: *Theor. Comput. Sci.* 200.1-2 (1998), pp. 135–183. DOI: 10.1016/S0304-3975(98)00009-7.

Index

Symbols

General

$f[x \mapsto y]$ ..... Function update; 11

$\bigwedge$ ..... The universal quantifier in Isabelle/Pure; 217

Graphs and their Decompositions

G ..... A graph; 12

E ..... The set of edges of a graph G; 12

V ..... The set of vertices of a graph G; 12

$G[X]$ ..... The subgraph induced by X; 12

$\overline{v}$ ..... A possibly infinite sequence of vertices, $v_1, v_2, \ldots$; 13

$d(v, w)$ ..... The distance between vertices v and w; 95

$N_r^G(v)$ ..... The r-neighborhood of v, defined as $\{w \in V \mid d(v, w) \leq r\}$; 95

$\mathrm{tw}(G)$ ..... The treewidth of G; 92

$\overline{X}$ ..... A finite sequence of vertices; 101; see also $\overline{X}$ in the context of the modal μ-calculus

$\mathcal{B}_t^{\downarrow}$ ..... The union of the bags of a subtree in a Kelly decomposition; 153

$\preceq$ ..... The transitive/reflexive closure of the arc relation on DAGs; 152

X_d ..... A bag of a DAG decomposition; 157

$X_{\succeq d}$ ..... The union of a sub-DAG of a DAG decomposition; 157

L_μ, the Modal μ-Calculus

σ ..... A signature, that is, a set of proposition symbols; 16

$L_\mu[\sigma]$ ..... The set of formulas of the modal μ-calculus over the signature σ; 16

$\top$ ..... "Top", always true; 16

$\bot$ ..... "Bottom", always false; 16

$\Box$ ..... L_μ operator "for all successors"; 16

$\Diamond$ ..... L_μ operator "for some successor"; 16

$\mu X.\varphi$ ..... L_μ operator for the least fixed point of φ; 17

$\nu X.\varphi$ ..... L_μ operator for the greatest fixed point of φ; 17

$|\varphi|$ ..... The length of φ; 17

$\mathcal{M}$ ..... A σ-structure; 18

$[\![\varphi]\!]^{\mathcal{M}}_{\mathcal{V}}$ ..... The set of vertices of $\mathcal{M}$ where φ is true; 19

$\mu X^\alpha.\varphi$ ..... The α-approximant of $\mu X.\varphi$; 22

$\nu X^\alpha.\varphi$ ..... The α-approximant of $\nu X.\varphi$; 22

F^X_φ ..... One fixed point iteration of $\varphi(X)$, that is, $F^X_\varphi : 2^{\mathcal{M}} \to 2^{\mathcal{M}},\ S \mapsto [\![\varphi]\!]^{\mathcal{M}}_{\mathcal{V}[X \mapsto S]}$; 20

$\mathrm{sub}(\varphi)$ ..... The set of indexed subformulas of φ; 35

$\mathrm{CL}(\varphi)$ ..... The closure of φ, that is, the set of all closed subformulas; 35

$\mathrm{CL}^+(\varphi)$ ..... Defined as $\mathrm{CL}(\varphi) \setminus \{(\varphi, 0)\}$; 35

$\mathrm{FLC}(\varphi)$ ..... The Fischer-Ladner Closure of φ; 37

$\mathrm{PT}_P(\varphi)$ ..... The set of priority tracking variants of φ; 112

$\mathrm{CL}_P(\varphi)$ ..... Defined as $\mathrm{PT}_P(\mathrm{CL}(\varphi))$; 113

$\overline{X}$ ..... A finite sequence of fixpoint variables; 39; see also $\overline{X}$ in the context of graphs

$\overline{P}$ ..... A finite sequence of proposition symbols; 102

$\partial_{\overline{P}}(\mathcal{M}, \overline{X})$ .... $\mathcal{M}$ with colored interface; 102

$\mathrm{tp}_{L,\overline{P}}(\mathcal{M}, v, \overline{X})$ The $(L, \overline{P})$-type of v in $\mathcal{M}, \overline{X}$, defined as $\{\varphi \in \mathrm{CL}_P(L) \mid \partial_{\overline{P}}(\mathcal{M}, \overline{X}), v \models \varphi\}$; 114

$\mathcal{T}_{L,\overline{P}}(\mathcal{M}, \overline{X})$ .. The set of $(L, \overline{P})$-types realized in $\mathcal{M}$, defined as $\{\mathrm{tp}_{L,\overline{P}}(\mathcal{M}, v, \overline{X}) \mid v \in V(\mathcal{M})\}$; 114

$\mathcal{T}_L(\overline{P})$ ..... The set of all candidates for $(L, \overline{P})$-types, defined as $2^{\mathrm{CL}_P(L)}$; 114

$\mathrm{prio}_\varphi(\psi \rightsquigarrow \chi)$. The minimum priority of all fixpoint operators enclosing χ in ψ; 127

ψ'^y ..... True on x' iff a profile $y' \sqsubseteq y$ is possible on $(P, (x', \psi))$; 127

Parity Games

$\Diamond$ ..... The parity game player "even"; 47

$\Box$ ..... The parity game player "odd"; 47

$\overline{i}$	The other player, that is, $\overline{\Diamond} = \Box$ and $\overline{\Box} = \Diamond$; 47
$V_\Diamond$	The set of vertices of player $\Diamond$; 47
$V_\Box$	The set of vertices of player $\Box$; 47
ω	The priority function $V \to \mathbb{N}$; 47
$\mathcal{M} \ltimes \varphi$	The model-checking game; 60
$\mathcal{M} \ltimes_X \varphi$	The partial model-checking game with interface $\{(v, \psi) \in X \times \mathrm{CL}(\varphi) \mid \psi \text{ starts with } \Diamond \text{ or } \Box\}$; 118
$\sqsubseteq$	The reward ordering on priorities. The "at least as good" relation on profiles; 118, 119
P^φ	A game that simulates $\mathcal{M} \ltimes_X \varphi$; 132

Schriftenreihe **Foundations of computing**
Hrsg.: Prof. Dr. Stephan Kreutzer, Prof. Dr. Uwe Nestmann,
Prof. Dr. Rolf Niedermeier

ISSN 2199-5249 (print)
ISSN 2199-5257 (online)

01: Bevern, René van:
Fixed-Parameter Linear-Time Algorithms for NP-hard Graph and Hypergraph Problems Arising in Industrial Applications. — 2014. — 225 S.
ISBN **978-3-7983-2705-4** (print) EUR **12,00**
ISBN **978-3-7983-2706-1** (online)

02: Nichterlein, André:
Degree-Constrained Editing of Small-Degree Graphs. — 2015. — xiv, 225 S.
ISBN **978-3-7983-2705-4** (print) EUR **12,00**
ISBN **978-3-7983-2706-1** (online)

03: Bredereck, Robert:
Multivariate Complexity Analysis of Team Management Problems. — 2015. — xix, 228 S.
ISBN **978-3-7983-2764-1** (print) EUR **12,00**
ISBN **978-3-7983-2765-8** (online)

04: Talmon, Nimrod:
Algorithmic Aspects of Manipulation and Anonymization in Social Choice and Social Networks. — 2016. — xiv, 275 S.
ISBN **978-3-7983-2804-4** (print) EUR **13,00**
ISBN **978-3-7983-2805-1** (online)

05: Siebertz, Sebastian:
Nowhere Dense Classes of Graphs. Characterisations and Algorithmic Meta-Theorems. — 2016. — xxii, 149 S.
ISBN **978-3-7983-2818-1** (print) EUR **11,00**
ISBN **978-3-7983-2819-8** (online)

06: Chen, Jiehua:
Exploiting Structure in Computationally Hard Voting Problems. — 2016. — xxi, 255 S.
ISBN **978-3-7983-2825-9** (print) EUR **13,00**
ISBN **978-3-7983-2826-6** (online)

07: Arbach, Youssef:
On the Foundations of dynamic coalitions. Modeling changes and evolution of workflows in healthcare scenarios. — 2016. — xv, 171 S.
ISBN **978-3-7983-2856-3** (print) EUR **12,00**
ISBN **978-3-7983-2857-0** (online)

08: noch nicht erschienen

Universitätsverlag der TU Berlin

Parity Games, Separations, and the Modal μ-Calculus

The topics of this thesis are the modal μ-calculus and parity games. The modal μ-calculus is a common logic for model-checking in computer science. The model-checking problem of the modal μ-calculus is polynomial time equivalent to solving parity games, a 2-player game on labeled directed graphs.
We present the first FPT algorithms (fixed-parameter tractable) for the model-checking problem of the modal μ-calculus on restricted classes of graphs, specifically on classes of bounded Kelly-width or bounded DAG-width. In this process we also prove a general decomposition theorem for the modal μ-calculus and define a useful notion of type for this logic.
Then, assuming a class of parity games has a polynomial time algorithm solving it, we consider the problem of extending this algorithm to larger classes of parity games. In particular, we show that joining games, pasting games, or adding single vertices preserves polynomial-time solvability. It follows that parity games can be solved in polynomial time if their underlying undirected graph is a tournament, a complete bipartite graph, or a block graph.
In the last chapter we present the first non-trivial formal proof about parity games. We explain a formal proof of positional determinacy of parity games in the proof assistant Isabelle/HOL.

ISBN 978-3-7983-2887-7 (print)
ISBN 978-3-7983-2888-4 (online)

ISBN 978-3-7983-2887-7

http://verlag.tu-berlin.de