Memory and Delay in Regular Infinite Games

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften
der RWTH Aachen University zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Diplom-Informatiker
Michael Holtmann

aus Düsseldorf

Berichter: Universitätsprofessor Dr. Dr.h.c. Wolfgang Thomas
Universitätsprofessor Dr. Erich Grädel

Tag der mündlichen Prüfung: 9. Mai 2011

Zusammenfassung

Unendliche Zwei-Personen-Spiele sind ein ausdrucksstarkes und anpassungsfähiges Werkzeug bei der Modellierung und Verifikation reaktiver Systeme. Es ist allgemein bekannt, dass beispielsweise die Konstruktion eines Controllers, der beliebig lange in der Umgebung eines Systems agiert, reduziert werden kann auf die Berechnung einer Gewinnstrategie in einem unendlichen Spiel (Pnueli und Rosner, 1989). Für die Klasse der Spiele mit regulärer Gewinnbedingung haben Büchi und Landweber 1969 gezeigt, dass einer der Spieler eine Gewinnstrategie hat, die durch einen endlichen Automaten realisierbar ist. Basierend auf diesem grundlegenden Resultat hat die Forschung viele Versuche unternommen, die Lösungsverfahren für unendliche Spiele weiterzuentwickeln. Dies betrifft sowohl den zeitlichen Aufwand, den man benötigt, um Gewinnstrategien zu berechnen, als auch den Speicherbedarf, um diese dann zu programmieren. In der vorliegenden Arbeit geht es hauptsächlich um den zweiten Aspekt. Es werden zwei Probleme betrachtet, die im Hinblick auf die Konstruktion kleiner Controllerspezifikationen von Bedeutung sind.

Im ersten Teil der Arbeit beschäftigen wir uns mit dem klassischen Problem, endlich repräsentierbare Gewinnstrategien zu berechnen, die von Automaten mit möglichst wenig Speicher (das heißt mit möglichst wenig Zuständen) realisiert werden können. Auch wenn ein Resultat von Dziembowski et al. aus dem Jahre 1997 besagt, dass es exotische reguläre Spiele gibt, für die Gewinnstrategien nur durch Automaten implementiert werden können, die gemessen an der Größe der Spielarena mindestens exponentiell groß sind, so erfordern die meisten praktischen Beispiele doch weit weniger Speicherplatz. Wir stellen einen effizienten Algorithmus für die Reduktion des für die Implementierung von Gewinnstrategien verwendeten Speichers vor und wenden ihn auf mehrere Klassen regulärer Bedingungen an. Außerdem zeigen wir, dass mit unserem Verfahren ein exponentieller Gewinn bezüglich der Speichergröße erzielt werden kann.

Im zweiten Teil dieser Arbeit führen wir einen verallgemeinerten Begriff von Strategien ein. Einer der Spieler darf jeden seiner Züge für eine endliche Anzahl von Schritten hinauszögern. Mit anderen Worten, er kann bei seinen Entscheidungen einen Vorgriff auf die Züge des Gegners ausnutzen. Dieses Szenario lässt sich beispielsweise in verteilten Systemen wiederfinden, zum Beispiel, wenn Pufferspeicher für die Kommunikation zwischen entfernten Komponenten eingesetzt werden. Unser Konzept von Strategien erfasst die Klasse der stetigen Operatoren und ist damit eine Erweiterung der Arbeit von Hosch und Landweber (1972) und insbesondere auch der von Büchi und Landweber (1969). Wir zeigen, dass das Problem, ob eine gegebene reguläre Spezifikation durch einen stetigen Operator erfüllt werden kann, entscheidbar ist und dass jede solche Lösung auch auf eine mit beschränktem Vorgriff reduziert werden kann. Aus unseren Ergebnissen leiten wir eine verallgemeinerte Determiniertheit regulärer Bedingungen ab.

Abstract

Infinite two-player games constitute a powerful and flexible framework for the design and verification of reactive systems. It is well-known that, for example, the construction of a controller acting indefinitely within its environment can be reduced to the computation of a winning strategy in an infinite game (Pnueli and Rosner, 1989). For the class of regular games, Büchi and Landweber (1969) showed that one of the players has a winning strategy which can be realized by a finite-state automaton. Based on this fundamental result, many efforts have been made by the research community to improve the solution methods for infinite games. This is meant with respect to both the time needed to compute winning strategies and the amount of space required to implement them. In the present work we are mainly concerned with the second aspect. We study two problems related to the construction of small controller programs.

In the first part of the thesis, we address the classical problem of computing finite-state winning strategies which can be realized by automata with as little memory, i.e., number of states, as possible. Even though it follows from a result of Dziembowski et al. (1997) that there exist exotic regular games for which the size of automata implementing winning strategies is at least exponential in the size of the game arena, most practical cases require far less space. We propose an efficient algorithm which reduces the memory used for the implementation of winning strategies, for several classes of regular conditions, and we show that our technique can effect an exponential gain in the size of the memory.

In the second part of this thesis, we introduce a generalized notion of a strategy. One of the players is allowed to delay each of his moves for a finite number of steps, or, in other words, exploit in his strategy some lookahead on the moves of the opponent. This setting captures situations in distributed systems, for example, when buffers are present in communication between remote components. Our concept of strategies corresponds to the class of continuous operators, thereby extending the work of Hosch and Landweber (1972) and, in particular, that of Büchi and Landweber (1969). We show that the problem whether a given regular specification is solvable by a continuous operator is decidable and that each continuous solution can be reduced to one of bounded lookahead. From our results, we derive a generalized determinacy of regular conditions.

Acknowledgments

I am very grateful to all the people who have contributed to the success of this thesis, either directly or indirectly.

First and foremost, I would like to thank my supervisor Wolfgang Thomas for giving me the opportunity to work as an academic researcher. His continuing support and helpful advice have been of great value to me throughout the past years. It was him who suggested to me to apply for the AlgoSyn Research Training Group, where I enjoyed three years of interdisciplinary exchange.

I would like to thank Erich Grädel for his kind readiness to act as a co-examiner of this thesis.

I would like to thank all my co-authors whom I have had the pleasure to work with: Marcus Gelderie, Łukasz Kaiser, Christof Löding and, again, Wolfgang Thomas.

I would like to thank all my colleagues at I7 and MGI, especially those who helped me with proofreading a first draft of this thesis and who made many helpful comments on its contents: Tobias Ganzow, Marcus Gelderie, Łukasz Kaiser, Daniel Neider, Bernd Puchala, Roman Rabinovich, Frank Radmacher and Michaela Slaats.

Finally, I am deeply indebted to my friends and family for their constant support throughout the past years. Especially, I would like to mention my affectionate girlfriend Franzi for continuing encouragement and plenty of patience she had with me.

Contents

Introduction

The theory of infinite games in computer science was initiated by work of Alonzo Church, who raised a problem today referred to as the *Church Synthesis Problem* [Chu57, Chu63]: one is given a circuit and a requirement between input sequences to the circuit and the corresponding outputs, i.e., a binary relation between infinite sequences over finite alphabets. Church asked whether it was possible to algorithmically synthesize a new circuit which satisfies the requirement in the sense that, when faced with any input, it generates an output which is in relation to the input. Alternatively, one should be able to determine that no such circuit exists. Moreover, generation of the output should be done in an on-demand fashion, i.e., the i-th output letter should depend only on the first i letters of the input sequence.

The above scenario naturally arises as a simple format of an infinite game, called *Gale-Stewart game* [GS53, Mos80]. There are two players, *Player I* choosing the input sequence and *Player O* choosing the output sequence. The game proceeds in rounds where in each round Player I chooses one letter from a given input alphabet, and Player O answers by one letter from an output alphabet, afterwards. The requirement is modeled by a *winning condition*, which is given as an omega-language L over pairs of input and output letters. A play is winning for Player O if the pair of sequences produced by the players is contained in the language L; otherwise Player I wins. To this end, Player I tries to take appropriate actions to violate the given specification; his adversary must find convenient answers in order to verify that the specification can in fact be satisfied.

In the game setting described above the players have opposing objectives, therefore each pair of sequences is winning for either of the two. In the study of infinite games one goes even further, by asking the question of whether a given Gale-Stewart game is *determined*. A game is said to be determined if one of the players has a *winning strategy*, and a player has a winning strategy if he can continuously react to the decisions taken by the opponent such that the resulting pair of sequences satisfies his objective, no matter what letters the adversary chooses. In their work [BL69], Büchi and Landweber deal with *regular* Gale-Stewart games. A (winning) condition is called regular if it is the infinite behavior of a finite transition system (with a standard acceptance condition), or, equivalently, if it can be expressed in monadic second-order logic. Büchi and Landweber showed that regular games are determined and that they can be solved effectively. This means that one can decide which player has a winning strategy and compute a finite-state automaton (with output) implementing such a winning strategy.

Embedding it into the wider context of automated verification, Church's setting can be reformulated as the synthesis problem for *reactive systems*. In its most general form a reactive system is any discrete event system in which several agents interact with each other. One agent is a programmable entity, called *Controller*, acting indefinitely within the environment of the system. The task is to program the controller such that the run of the system satisfies a given specification. During the last decades, the research community has been successfully concerned with reducing the synthesis problem for reactive systems to questions on infinite games [PR89, ALW89, MP95]. According to that, the construction of a correct controller amounts to the computation of a winning strategy. Many approaches to find algorithmic solutions to infinite games have been developed, providing methods for automatic construction of controllers. For a comprehensive overview of fundamental results and the connection to both logic and the theory of omega-automata, the reader is referred to the survey [Tho97] and the textbook [PP04]; for further references to the topic see also [GTW02].

Whereas Büchi's and Landweber's work served as one of the first steps into the theory of program synthesis, many efforts have been made to come up with generalizations and extensions of their fundamental result. To this end, a huge part of the literature deals with modifications of the basic setting. The first one concerns the system or the specification (or both), for example, infinite game arenas and non-regular winning conditions are considered [Wal96, Cac03, BSW03]. Another aspect is the way games are played, regarding in particular the influence of a probabilistic nature or the effect of multiple players [CMJ04, Kuč11].

In this work we mostly restrict to the basic setting, i.e., we consider games between two players on a finite game arena, with a regular winning condition. One of the players represents a possibly hostile environment, modeling for example a user making corruptive requests to a system. The other player has the role of the controller. She has to react to the adversary's behavior such that the run of the system under consideration satisfies a given requirement.

Aims of the Thesis. Due to the enormous size of real systems, many efforts have been made to improve the solution methods for infinite games. We mean this with respect to both the time needed to determine the winner and the space required to implement winning strategies. In this work we are mainly interested in the second aspect. The thesis consists of two parts, each of which is devoted to the study of one problem that is related to the size needed for the construction of controller programs.

In the first part of the thesis we deal with regular infinite games on finite graphs. We address the classical problem of *memory reduction*, i.e., the reduction of the amount of memory that is needed to implement winning strategies for a given game. The memory is an abstraction containing relevant information about the players' behavior in the history of a play. We propose an efficient algorithm which reduces the size of the memory that is needed to realize winning strategies by finite automata, and we prove that our technique can result in an exponential gain (compared to known approaches) in the number of states.

In the second part of the thesis we deal with regular Gale-Stewart games. We introduce the notion of a strategy with *finite delay*, which incorporates information about the adversary's future behavior. This means that one player has to make a commitment to some of his upcoming moves, and the other player can rely on this knowledge when taking a decision. We use this concept to show an extension of the Büchi-Landweber Theorem, namely that the problem whether a regular condition is solvable by a continuous operator is decidable.

Part I: Memory Reduction for Strategies in Infinite Games

As noted in [ALW89, PR89], the amount of memory that is required to realize a particular winning strategy in an infinite game corresponds to the size of the respective controller program for a reactive system. Thus, besides studying the algorithmic complexity of deciding the winner, the problem of finding winning strategies which are easy to implement has always been an intensely investigated branch in the field of infinite games. As already indicated, it was shown in [BL69] that the winner of a regular game has a *finite-state* winning strategy, i.e., a winning strategy which can be executed by a finite *strategy automaton*. The size of the *memory*, i.e., the number of states, of such an automaton is an appropriate measure for the quality of the underlying strategy. Many results of the past three decades have revealed both upper and lower bounds for the size of strategy automata required to solve infinite games. Whereas some conditions (like, for example, *reachability* or *parity* objectives) can be solved by *positional* winning strategies, i.e., they can be implemented by an automaton with only one state, others require an amount of memory exponential in the size of the game arena or the winning condition [EJ91, Tho95, DJW97, Zie98].

Let us mention some results on standard winning conditions relevant to our work. *Staiger-Wagner* and *Muller* conditions depend on the set of vertices that are visited at least once and on the set of vertices that are visited infinitely often, respectively. (Staiger-Wagner games are sometimes also referred to as *weak Muller* games, for example in [Tho02].) Both types of objective are given by a family $\mathcal{F} = \{F_1, \dots, F_k\}$, where each F_i is a subset of vertices of the underlying game graph (cf. [Mul63]). A play satisfies the Staiger-Wagner condition induced by $\mathcal{F}$ if all vertices occurring in the play (at least once) form one of the sets F_i. Analogously, a play is good with respect to the Muller condition $\mathcal{F}$ if the set of all vertices occurring infinitely often coincides with some F_i. The optimal bound for the size of the memory required to solve a Staiger-Wagner game over an arena with n vertices is 2^n. In a technical report, McNaughton introduced the name "order vector" for the set of all permutations of a given set of vertices [McN65]. Gurevich and Harrington renamed the data structure as *Latest Appearance Record* (short: LAR) and used it to solve Muller games [GH82]. Later on, Dziembowski et al. showed that $n!$, i.e., the amount of memory provided by a LAR, is a matching upper and lower bound for the realization of a winning strategy in a Muller game with $\mathcal{O}(n)$ vertices [DJW97].

A similar result was achieved in [Hor05] for *Streett* games, which capture the notion of strong fairness in reactive systems and are given by a family $\Omega = \{(E_1, F_1), \dots, (E_k, F_k)\}$ of pairs of subsets of vertices. To satisfy the Streett condition induced by Ω, the following must hold for each i: if F_i is visited infinitely often, then E_i must also be visited infinitely often. Horn showed that there exist Streett games with both an arena and a winning condition of size $\mathcal{O}(k^2)$ whose solution require a memory of size at least $k!$, i.e., factorial in the number of Streett pairs, whereas the best upper bound known is $k! \cdot k^2$ [BLV96].

Another class of winning conditions are *Request-Response* conditions, which capture basic liveness requirements [WHT03]. Such a condition is given in the same format as a Streett condition, and a play satisfies the induced Request-Response condition if every request, i.e., a visit to some E_i, is eventually followed by a matching response, i.e., a visit to the corresponding set F_i. As shown in [WHT03], winning strategies in Request-Response games may require a memory of size $2^k \cdot k$, i.e., exponential in the number of request-response pairs. For a short survey on algorithms for the computation of winning strategies and for some further references to related literature, the reader is referred to Chapter 2.

By the current state of research, the problem of finding winning strategies of optimal size is intractable, i.e., it cannot be solved in polynomial time. Hunter and Dawar have shown that the problem of deciding the winner of a Muller game is Pspace-complete if the winning condition is represented appropriately [HD05]. As a compromise, we consider the problem of memory reduction. Our motivation arises from the fact that, even though there exist exponential lower bounds for the size of the memory needed for implementing winning strategies, most practical examples require far less space. The problem with standard algorithms is that they take account of the aforementioned bounds. If a given game is to be solved, then the maximal amount of memory that may be needed is a priori allocated, without further considering the structure of the game graph or the winning condition. For example, winning strategies in a Staiger-Wagner game are implemented by storing the set of vertices which are visited in a play. This entails a memory of size exponential in the size of the arena, no matter how much memory is actually needed for solving the game at hand. To overcome this (possibly unavoidable) weakness, we intend to investigate properties of the game in order to reduce the used memory *before* the computation of winning strategies.

In the first part of this thesis, we propose an efficient technique for the algorithmic reduction of the memory needed for the implementation of winning strategies in infinite games on finite graphs, for several classes of omega-regular objectives. Our algorithm is based on the notion of *game simulation* (see for example [Tho95]). The idea of a game simulation is to reduce the solution of a given game to the solution of a new one, which has a simpler winning condition but an extended arena containing the memory needed to solve the original game. Our approach to memory reduction is the simplification of the extended game graph, such that the properties of game simulation are preserved. Applying efficient techniques for state space reduction of omega-automata, we compute an equivalence relation on the set of memory contents. The new memory is then the set of equivalence classes of this relation.

We apply our technique to Staiger-Wagner, Request-Response, Muller and Streett conditions and show that our approach can result in an exponential gain in the size of the memory. For Staiger-Wagner, Request-Response and Streett games, we present examples where standard algorithms yield very complicated winning strategies (of exponential size), but our algorithm produces a memory of constant size.

Organization of Part I. The first part of the thesis comprises Chapter 3. In Section 3.1, we review the standard minimization technique for finite automata with output, in a context where they are used for the implementation of finite-state strategies. In Section 3.2 we present our approach to memory reduction and formally prove its correctness. Sections 3.3 through 3.5 are devoted to the application of our technique to games with particular omega-regular winning conditions. Section 3.6 serves as an evaluation of our approach.

The results presented in the first part of the thesis were obtained in collaborations with Christof Löding and Marcus Gelderie. They are published in [HL07, GH11].

Part II: Infinite Games with Finite Delay

As already remarked, the Church Synthesis Problem was solved by Büchi and Landweber in the framework of regular infinite games. However, the underlying motivation for their work was a slightly different question posed in terms of *operators*, i.e., functions mapping infinite sequences over finite alphabets from one space into another. Following the notation of [BL69], a *condition* is any binary relation between infinite sequences, and we say that an operator *solves* a condition if for each given input the produced output is in relation to the input. Büchi and Landweber motivated their work by descriptive set theory, in the sense that they asked for the particular type of *continuity* of operators which solve a given condition. This served as an entry into the study of the topological properties inherent in binary relations between infinite sequences. As we will point out in the second part of this thesis, there is a close connection between the degree of continuity of an operator and generalized solution concepts for infinite games.

Basically, we deal with three fundamental levels of operators, referring to the Cantor topology over the space of infinite sequences. The most general ones are the *continuous* operators, where each output letter depends on a finite part of the input (see for example [TB73, TL93]). An operator is called *uniformly continuous* if for each position i there exists a bound for the lengths of the input prefixes on which the i-th output letter depends. On a further level of specialization, we are dealing with *Lipschitz continuous* operators. These are operators where, for each i, the i-th output letter depends on at most the first $i+d$ letters of the input, for some natural number d. In this thesis, Lipschitz continuous operators are mostly called *bounded-delay* operators.

Whereas the aforementioned notions of continuity allow that an input prefix is longer than the output prefix it determines, Büchi and Landweber considered in [BL69] a more restricted class of operators. They introduced the notion of a *shift* operator, where the term "shift" refers to the fact that each output prefix is determined by some input prefix of at most the same length. To put it simply, if an output prefix of length i is determined by an input prefix of length j such that $j \leq i$, then the shift at position i is $i-j$. If, for all i, the shift at position i is at least d, then we speak of a d-shift operator.

Büchi and Landweber were particularly interested in deciding whether a given specification allows for a solution of shift *zero*, which captures the case of a classical strategy. In fact, they defined the notion of determinacy using the terminology of shift operators. Accordingly, a condition is determined if either there exists a 0-shift operator which guarantees that the produced output is in relation to the input, or there exists an analogous operator ensuring that the output is *not* in relation to the input. Büchi and Landweber proved that, for each regular condition, precisely one of the aforementioned cases occurs, and that one can compute a corresponding solution which can be realized by a deterministic finite automaton. They obtained these results using the findings achieved by McNaughton on the infinite behavior of transition systems [McN66].

The ideas of [BL69] were resumed by Hosch and Landweber, who introduced the name *delay* for the notion dual to that of a shift [HL72]. In a delay operator, the j-th output letter is determined by an input prefix of at least the same length. (We mostly prefer the term "delay" to the term "shift", because one may intuitively expect that at least one additional input letter should be given until the next output letter can be produced.) In the framework of Gale-Stewart games, a delay operator corresponds to a generalized notion of a strategy for Player O. More precisely, she is allowed to choose an additional move *skip*, instead of some letter of the output alphabet. Each *skip* is disregarded when evaluating the winning condition.

Example. Consider the Boolean alphabet $\mathbb{B} := \{0,1\}$ and the regular omega-language defined by

$$r := \binom{0\,a}{a\,*}(\mathbb{B}^2)^\omega + \binom{1 * * \,b}{b * * *}(\mathbb{B}^2)^\omega,$$

where $a, b \in \mathbb{B}$ and $*$ denotes any bit. If Player I chooses 0 as his first bit, then Player O needs to know a to satisfy the winning condition. To this end, she answers by choosing *skip*, such that Player I has to give the bit a and Player O can respond with the same bit, only afterwards. In this situation, the move *skip* is required only once; accordingly, we speak of delay one. Analogously, if Player I chooses 1 as his first bit, then Player O needs delay three to obtain b.

In the above example Player O wins with delay three, but neither with delay two nor with delay one. That means that the condition defined by the expression r has a 3-delay solution, but it has no solution of smaller delay. The main result of Hosch and Landweber in [HL72] is that, for the class of regular omega-languages, the problem whether there exists a natural number d such that a given condition has a d-delay solution is decidable. Their proof also exhibits an upper bound for the required constant, by exploiting findings from the field of Boolean circuit theory obtained by Even and Meyer [EM69].

As a practical application of Hosch's and Landweber's result, one may assume that the controller of a system is equipped with a finite buffer of length d such that the input letters supplied by the environment can be stored in that buffer. Then, the i-th output letter can be produced with delay d, i.e., on the basis of the first $i+d$ input letters.

Conversely, in many realistic situations of distributed systems the transmission of data is deferred. This means that in some component a decision has to be taken at a point of time where the actions of the other components are not yet known. This captures the case that the output has to be produced with a certain shift.

Whereas Büchi's and Landweber's setting is limited to standard controllers, the result of Hosch and Landweber implies that for both the aforementioned scenarios one can decide whether a bounded lookahead into the future behavior of the adversary will help in satisfying a given regular specification. Therefore, Hosch's and Landweber's work extended the findings in [BL69].

In the second part of the thesis we further generalize the results obtained in [HL72], getting rid of the obstacle that the production of an output letter may involve only bounded knowledge about the future of the input. Our motivation is to introduce a notion of a strategy which captures the class of continuous operators. To do so, we consider strategies with *finite delay*, i.e., we allow one of the players to postpone each of his moves for an arbitrary finite number of steps. It will turn out that, for the class of regular omega-languages, the problem whether a given condition admits a finite-delay solution is decidable in time doubly exponential in the size of the representation of the specification. Moreover, we show that each solution can be reduced to one of doubly exponential bounded delay. Apart from treating a more general scenario, our proof is considerably simpler and gives lower complexity bounds than the one in [HL72].

We use our results to derive a *generalized determinacy* of regular conditions. More precisely, we show the following extension of the Büchi-Landweber Theorem: let L be a regular condition which is *not* solvable by a 0-shift operator.

Then, L is not solvable by any continuous operator either, or there exists a minimal constant d_L (uniquely determined by L) such that L has a solution with bounded delay d_L. The problem which one of these two cases applies is decidable, and the constant d_L is computable.

Organization of Part II. The second part of the thesis comprises Chapter 4, Chapter 5 and Chapter 6. Chapter 4 serves as an excursus into the field of topology. In particular, it points out the connection between continuous operators and games with finite delay. Chapter 5 is devoted to the proof that, for the class of regular languages, the problem whether a given condition is solvable with finite delay is decidable. In Chapter 6 we extend our results to a concurrent setting.

Most of the results presented in the second part of the thesis were obtained in collaborations with Łukasz Kaiser, Frank Pöttgen and Wolfgang Thomas. They are published in [HKT10, Pöt10].

Chapter 1

Preliminaries

In this chapter we fix the notation used throughout this thesis. We introduce basic notions on formal languages and automata both over finite and infinite words. Moreover, we define the two game-theoretic settings which are going to be considered.

1.1 Basic Notation

For any set M, we write 2^M for the *power set* of M, i.e., the set of all subsets of M.

An *alphabet* is a finite set; each of its elements is called a *letter*. Usually, we write Σ for an alphabet and $a, b, \ldots$ for its letters. The *Boolean* alphabet $\{0, 1\}$ is denoted $\mathbb{B}$. The set of natural numbers is denoted $\mathbb{N}$, and $\mathbb{N}_+$ is the set of natural numbers without zero, i.e., $\mathbb{N}_+ := \mathbb{N} \setminus \{0\}$. The set of integers is denoted $\mathbb{Z}$ and the set of positive integers is $\mathbb{Z}_+ := \{i \in \mathbb{Z} \mid i \geq 1\}$; analogously, the set of negative integers is $\mathbb{Z}_-$. We write $\mathbb{R}$ for the set of real numbers.

1.2 Words and Formal Languages

We write Σ^* and Σ^ω for the set of finite and infinite *words*, respectively, over Σ. Usually, finite words are denoted $u, v, \ldots$ whereas $\alpha, \beta, \ldots$ are infinite words.

For $u \in \Sigma^*$, we write $|u|$ for the *length* of u, i.e., the number of letters of u, and $\Sigma^n := \{u \mid |u| = n\}$ is the set of all words of length n. The *empty word* has length 0 and is written as ε. The set $\Sigma^* \setminus \{\varepsilon\}$ of all non-empty words is denoted Σ^+. For $u \in \Sigma^*$ and $0 \leq i, j \leq |u - 1|$, we write $u[i, j]$ for the infix of u from position i to position j and $u[i] := u[i, i]$; if $j < i$ then $u[i, j] := \varepsilon$. Infixes of ω-words are written analogously. For $n_1, n_2 \in \mathbb{N}$ with $n_1 < n_2$, we write $\Sigma^{[n_1, n_2]}$ for $\bigcup_{n_1 \leq n \leq n_2} \Sigma^n$.

A *language* is a subset of either Σ^* or Σ^ω. In the first case we call it a $*$-language, or finitary language, and in the second case we call it an ω-language, or infinitary language. Usually, we use the capital letter L to denote a language. Given the words $u, v \in \Sigma^*$ and $\alpha \in \Sigma^\omega$, let uv be the finite word resulting from the concatenation of u and v. Analogously, the infinite word $u\alpha$ results from the concatenation of u and α. Extending to languages, the concatenation LL' of a finitary language L and a finitary or infinitary language L' is the set of all possible concatenations of a word in L and a word in L'. This means, we have $LL' := \{uv \mid u \in L, v \in L'\}$. L^+ denotes the set of all (non-empty) finite concatenations of L with itself, and $L^* := L^+ \cup \{\varepsilon\}$; analogously, we write L^ω for the set of all infinite concatenations.

For an alphabet Σ, the set of *regular expressions* over Σ is inductively defined. The atomic expressions are $\varnothing$, ε and each $a \in \Sigma$ defining the languages $L(\varnothing) := \varnothing$, $L(\varepsilon) := \{\varepsilon\}$ and $L(a) := \{a\}$, respectively. If r_1, r_2 are regular expressions, then so are $r_1 + r_2$, $r_1 r_2$ and r_1^*. The expression $r_1 + r_2$ defines the union, meaning $L(r_1 + r_2) := L(r_1) \cup L(r_2)$. Analogously, $r_1 r_2$ defines the concatenation $L(r_1)L(r_2)$, and finally, the expression r_1^* defines the language $L(r_1)^*$. We say that a $*$-language $L \subseteq \Sigma^*$ is *regular* if $L = L(r)$ for some regular expression r over Σ.

Regular expressions are defined analogously for infinitary languages. An *ω-regular expression* is of the form $r_1 s_1^\omega + \cdots + r_n s_n^\omega$, where n is a natural number, $r_1, \ldots, r_n, s_1, \ldots, s_n$ are standard regular expressions, and $L(s_i^\omega) := L(s_i)^\omega$ (for $1 \leq i \leq n$). We say that an ω-language $L \subseteq \Sigma^\omega$ is *ω-regular* if $L = L(r)$ for some ω-regular expression r over Σ.

1.3 Automata

In this section we introduce the different types of automata that we are going to use in this work.

1.3.1 Automata on Finite Words

A *nondeterministic finite automaton* (NFA) over Σ is a tuple $\mathcal{A} = (Q, q_0, \Delta, F)$ where Q is a finite (non-empty) set of *states*, $q_0 \in Q$ is the *initial state*, $\Delta \subseteq Q \times \Sigma \times Q$ is the *transition relation*, and $F \subseteq Q$ is the set of *final states*. A *run* ρ_u of $\mathcal{A}$ on a word $u := u_0 \cdots u_{n-1}$ is a finite sequence $\rho_u(0) \cdots \rho_u(n)$ with $\rho_u(0) = q_0$ and $(\rho_u(i), u_i, \rho_u(i+1)) \in \Delta$ for all $i = 0, \ldots, n-1$. We define ρ_u to be *accepting* if $\rho_u(n) \in F$, and $\mathcal{A}$ *accepts* u if there exists an accepting run of $\mathcal{A}$ on u. The set of all words accepted by $\mathcal{A}$ is the $*$-language of $\mathcal{A}$ and denoted $L_*(\mathcal{A})$.

The automaton $\mathcal{A}$ is called *deterministic* (DFA) if for all $q \in Q, a \in \Sigma$ there exists exactly one $q' \in Q$ such that $(q, a, q') \in \Delta$. In this case, we can rewrite Δ as a function $\delta : Q \times \Sigma \to Q$. Note that a DFA has exactly one run on each possible input word.

Theorem 1.1. *Let Σ be an alphabet and $L \subseteq \Sigma^*$. Then, the following are equivalent:*

1. *L is a regular $*$-language.*
2. *There exists a NFA $\mathcal{A}$ with $L_*(\mathcal{A}) = L$.*
3. *There exists a DFA $\mathcal{A}'$ with $L_*(\mathcal{A}') = L$.*

For a given transition function δ, we denote δ^* the extension of δ from input letters to finite input words. We obtain $\delta^* : Q \times \Sigma^* \to Q$ with $\delta^*(q, \varepsilon) := q$ and $\delta^*(q, wa) := \delta(\delta^*(q, w), a)$, for all $q \in Q, w \in \Sigma^*, a \in \Sigma$. In Chapter 5 we need the following basic property of deterministic finite automata.

Lemma 1.2. *Let $\mathcal{A}$ be a DFA with n states and $|L_*(\mathcal{A})| = \infty$. Then, for all $i \in \mathbb{N}$, $\mathcal{A}$ accepts a word u_i of length $i \leq |u_i| \leq i{+}n$.*

Proof. Let $\mathcal{A}$ be a DFA with n states and $|L_*(\mathcal{A})| = \infty$. Since $L_*(\mathcal{A})$ is infinite it must be possible, for each $i \in \mathbb{N}$, to read a word u of length i such that from $\delta^*(q_0, u)$ a final state is reachable. Otherwise, the length of words accepted by $\mathcal{A}$ is bounded by i, which is a contradiction to the fact that $\mathcal{A}$ accepts infinitely many words. Then, from $\delta^*(q_0, u)$ we can reach a final state by a word u' of length at most n. The word uu' is accepted by $\mathcal{A}$ and is of length between i and $i{+}n$. $\square$

1.3.2 Automata on Infinite Words

We assume that the reader is familiar with the theory of ω-automata (see for example [Tho97]). Here, we restrict only to the basic notations.

A *nondeterministic Büchi automaton* (NBA) over Σ is a tuple $\mathcal{A} = (Q, q_0, \Delta, F)$ where Q, q_0, Δ and F are defined as for NFA. A run of a NBA is the natural extension of a run of a NFA to infinite words, i.e., it is an infinite sequence $\rho_\alpha = \rho_\alpha(0)\rho_\alpha(1)\rho_\alpha(2)\cdots$ such that $(\rho_\alpha(i), \alpha_i, \rho_\alpha(i+1)) \in \Delta$ for all $i \in \mathbb{N}$, where $\alpha = \alpha_0\alpha_1\alpha_2\cdots \in \Sigma^\omega$ is some infinite input to $\mathcal{A}$. Let $\mathrm{Inf}(\rho_\alpha)$ be the *infinity set* of ρ_α, i.e., the set of states visited infinitely often in run ρ_α. We define ρ_α to be accepting if $\mathrm{Inf}(\rho_\alpha) \cap F \neq \varnothing$, i.e., at least one final state is visited infinitely often. $\mathcal{A}$ accepts α if there exists an accepting run of $\mathcal{A}$ on α. The set of all words accepted by $\mathcal{A}$ is the ω-language of $\mathcal{A}$ and denoted $L_\omega(\mathcal{A})$. If a given

Büchi automaton has a deterministic transition structure then, analogously to DFA, we call it a DBA and rewrite Δ as δ.

A *deterministic*[1] *parity automaton* (DPA) over Σ is a tuple $\mathcal{A} = (Q, q_0, \delta, c)$ where Q, q_0 and δ are defined as for DFA, and c is a *coloring*, i.e., a function $c : Q \to \{0, \dots, m\}$ ($m \in \mathbb{N}$) from the set of states into a finite set of *colors* (or *priorities*). For $R \subseteq Q$, we denote $c(R)$ the set of colors assigned to any state in R, i.e., $c(R) := \{c(q) \mid q \in R\}$. A run of a DPA is defined analogously to a run of a NBA. Let α be an input to the DPA $\mathcal{A}$ and let $\mathrm{Inf}(c(\rho_\alpha))$ be the set of colors seen infinitely often in run ρ_α. We define the parity automaton $\mathcal{A}$ to accept α if $\max(\mathrm{Inf}(c(\rho_\alpha)))$, i.e., the maximal color seen infinitely often in the run of $\mathcal{A}$ on α, is even. Accordingly, the acceptance condition of $\mathcal{A}$ is called a max-*parity condition*.[2] As for Büchi automata, the set of all words accepted by $\mathcal{A}$ is the ω-language of $\mathcal{A}$.

A *deterministic weak parity automaton* has the same format as a deterministic parity automaton. The only difference is that it has a weaker acceptance condition: a run ρ is accepting if $\max(\mathrm{Occ}(c(\rho)))$, i.e., the maximal color occurring in ρ, is even.

The following theorem is a fundamental result on the expressive power of automata on infinite words (see for example [GTW02]).

Theorem 1.3. *Let Σ be an alphabet and $L \subseteq \Sigma^\omega$. Then, the following are equivalent:*

1. *L is a regular ω-language.*
2. *There exists a NBA $\mathcal{A}$ with $L_\omega(\mathcal{A}) = L$.*
3. *There exists a DPA $\mathcal{A}'$ with $L_\omega(\mathcal{A}') = L$.*

In the sequel, we simply use the terms "language" and "regular", if it is clear from the context whether we consider finitary or infinitary languages; moreover, we write $L(\mathcal{A})$ instead of $L_*(\mathcal{A})$ or $L_\omega(\mathcal{A})$, if it is clear what kind of automaton $\mathcal{A}$ is. Moreover, for a state q of any given automaton $\mathcal{A}$, we denote $\mathcal{A}_q$ the automaton $\mathcal{A}$ with initial state q.

1.4 Infinite Games

In this section we introduce two versions of infinite games, and the notion of a strategy. In Section 1.4.1 we deal with infinite games on finite arenas, where a play is considered as an infinite path through a finite graph. This notion of

[1]We consider only the deterministic version of parity automata.

[2]A min-parity condition is defined analogously, with the minimal color seen infinitely often.

game is used in Part I of the thesis. In Section 1.4.3 we define the more general setting of Gale-Stewart games [GS53, Mos80], which we need for Part II.

1.4.1 Games on Finite Graphs

An *infinite game* on a graph is a tuple $\Gamma = (G, \varphi)$, where G is a *game arena* and φ is a *winning condition*. The arena is a finite, directed graph $G = (V, V_0, V_1, E)$, where V is a set of *vertices* and $E \subseteq V \times V$ is a set of *edges* between vertices. The set V is partitioned into the two disjoint sets V_0, V_1, i.e., it holds $V = V_0 \uplus V_1$. We assume that each vertex has at least one successor, i.e., for all $v \in V$ the set $vE := \{v' \mid (v, v') \in E\}$ is non-empty. The game is played by two players, called *Player 0* and *Player 1*, where the set V_0 contains the vertices belonging to Player 0, and V_1 is the set of vertices belonging to Player 1; for brevity, we usually write only $G = (V, E)$.

The game Γ is played as follows. A token is placed on an initial vertex and moved through the graph by the players, for an infinite number of rounds. If the current vertex is owned by Player 0, then she makes the next move, and accordingly for Player 1. Clearly, moves are allowed only along edges. So, if the current vertex is v, then the respective player has to choose some v' such that $(v, v') \in E$, and the token is moved on from v to v'. This way, the players build up an infinite path through G, called a *play*. It is an infinite sequence $\varrho = \varrho(0)\varrho(1)\varrho(2)\cdots \in V^\omega$ such that $(\varrho(i), \varrho(i+1)) \in E$, for all $i \in \mathbb{N}$.

The *winning condition* (or *objective*) φ is the set of all plays which are *winning* for Player 0. That means, it is a set of infinite paths through G or, more formally, $\varphi \subseteq V^\omega$. A play is winning for Player 1 if it is not winning for Player 0.

1.4.2 Strategies and Memory

A *strategy* for Player 0 is a (partial) function $f : V^*V_0 \to V$ defining for each *game position* of Player 0 her next move. By a game position of Player 0 we mean a finite play prefix ending in some vertex $v \in V_0$. That means, for any finite play prefix $v_0 \cdots v_i$ with $v_i \in V_0$ it must hold $(v_i, f(v_0 \cdots v_i)) \in E$. A play $\varrho = \varrho(0)\varrho(1)\varrho(2)\cdots$ is played *according to f* if for every $i \in \mathbb{N}$ with $\varrho(i) \in V_0$ we have $\varrho(i+1) = f(\varrho(0)\cdots\varrho(i))$. For each $v \in V$, the function f is called a *winning strategy from v* for Player 0 if every play starting in v that is played according to f is winning for Player 0, i.e., it is contained in φ; we say that a player *wins from v* if he has a winning strategy from v. The *winning region* W_0 of Player 0 is the set of all vertices from where Player 0 wins. Winning strategies for Player 1 are defined analogously.

One of the main interests in the field of infinite games is to determine which player has a winning strategy. By a *game solution* we mean the winning regions W_0, W_1 of both players and corresponding winning strategies. Usually, we do not consider a particular initial vertex but ask for the whole winning region of each player. It is not hard to see that from a given vertex at most one of the players can have a winning strategy.

Remark 1.4. For each vertex v, at most one of the players wins from v, i.e., the set $W_0 \cap W_1$ is empty.

Proof. We prove Remark 1.4 by contradiction. Assume both players have a winning strategy from v, say f_0 for Player 0 and g_0 for Player 1. Consider the unique play ϱ_{f_0,g_0} that is built up when Player 0 and Player 1 play according to f_0 and g_0, respectively. Since f_0 is winning for Player 0 we have $\varrho_{f_0,g} \in \varphi$, for all strategies g of Player 1, in particular for g_0. Moreover, since g_0 is winning for Player 1, we have $\varrho_{f,g_0} \in \overline{\varphi} := V^\omega \setminus \varphi$, for all strategies f of Player 0, in particular for f_0. We get $\varrho_{f_0,g_0} \in \varphi \cap \overline{\varphi}$. A contradiction. □

The following notion is needed to speak about the existence of winning strategies.

Definition 1.5. Let $\Gamma = (G, \varphi)$ be an infinite game on a graph. We say that Γ is *determined* if, for each vertex v, one of the players has a winning strategy from v.

One of the most fundamental results in the algorithmic theory of infinite games is due to Büchi and Landweber, stating that each infinite game on a finite graph with a regular winning condition is determined [BL69]. An introduction into the winning conditions we are going to deal with in Part I of this thesis can be found in Chapter 2. All of the games presented there are determined.

1.4.2.1 Finite-state Strategies

Throughout the first part of the thesis we are concerned with the amount of *memory* that is needed to implement (winning) strategies. In the case of regular winning conditions it is sufficient to consider strategies which require only finite memory [BL69].

To get clearer what is meant by memory we introduce strategy automata, which are a special kind of DFA. Let $G = (V, E)$ be a game graph, where V_0 is the set of vertices belonging to Player 0. A *strategy automaton* (over V) for Player 0 is a tuple $\mathcal{A} = (S, s_0, \sigma, \tau)$ where S is a finite (non-empty) set of states, called *memory*, s_0 is the *initial memory content* and $\sigma : S \times V \to S$ is the *memory update function*, i.e., the transition function of $\mathcal{A}$. The major difference to a normal DFA is that a strategy automaton has a *transition choice function* $\tau : S \times V_0 \to V$, instead of a set of final states. The strategy automaton $\mathcal{A}$ defines the strategy $f_{\mathcal{A}}$ for Player 0 as follows: as soon as a move in G is made the automaton reads the new vertex and updates its state according to σ. If the new vertex is contained in V_0, then the transition choice function τ tells Player 0 where to move next. The strategy defined by $\mathcal{A}$ is the function $f_{\mathcal{A}} : V^*V_0 \to V$ where $f_{\mathcal{A}}(v_0 \cdots v_i) := \tau(\sigma^*(s_0, v_0 \cdots v_{i-1}), v_i)$, for $v_i \in V_0$.

A strategy f is called an *automaton strategy* (or *finite-state*) if there exists a strategy automaton $\mathcal{A}$ with $f_{\mathcal{A}} = f$. By the *size* of a strategy we mean the minimal number of states among all automata implementing this strategy. We say that a strategy is *positional* if it has size one; such a strategy depends only on the current vertex of the play. That means, if f is a positional strategy for Player 0, then $f(wv) = f(v)$ for all game positions wv of Player 0 ($w \in V^*, v \in V_0$). Note that a positional strategy f can be represented by a set of edges, namely by the set $E_f := \{(v, f(v)) \mid v \in V_0\}$.

We say that an infinite game is *positionally determined* if both players have positional winning strategies from their respective winning regions.

1.4.3 Gale-Stewart Games

In the second part of this thesis we use a more general notion of infinite game, called *Gale-Stewart game* [GS53]. It is played by two players, Player Input and Player Output, or short Player I and Player O. The game proceeds in rounds, where one round is played as follows: first Player I chooses a letter from a finite *input* alphabet Σ_{I}, then Player O chooses a letter from a finite *output* alphabet Σ_{O}. This way, the players build up two ω-words; Player I builds up $\alpha = a_0a_1a_2 \cdots \in \Sigma_{\mathrm{I}}^{\omega}$, and Player O builds up $\beta = b_0b_1b_2 \cdots \in \Sigma_{\mathrm{O}}^{\omega}$. The corresponding play is winning for Player O if the word $\alpha^\wedge\beta := \binom{a_0}{b_0}\binom{a_1}{b_1}\binom{a_2}{b_2} \cdots$ satisfies the winning condition, i.e., it is contained in a given ω-language L over $\Sigma := \Sigma_{\mathrm{I}} \times \Sigma_{\mathrm{O}}$. Otherwise, the play is winning for Player I. For $L \subseteq \Sigma^{\omega}$, the Gale-Stewart game with winning condition L is denoted $\Gamma(L)$. We say that L is *solvable* if Player O has a winning strategy in $\Gamma(L)$.

It is important to note that, for ω-regular L, the Gale-Stewart game $\Gamma(L)$ is positionally determined. This follows from the fact that it can be modeled by a parity game on a finite graph with finitely many colors (see Section 2.1.4 for more details). To this end, let $\mathcal{A} = (Q, q_0, \delta, c)$ over Σ be a DPA recognizing L. Each play starts in q_0 and proceeds as follows: if $q \in Q$ is the current state, then Player I chooses $a \in \Sigma_{\mathrm{I}}$, moving on to (q, a). Afterwards, Player O chooses $b \in \Sigma_{\mathrm{O}}$ and thereby moves to $\delta(q, \binom{a}{b})$. The game graph built up this way is of size linear in $|\mathcal{A}|$.

In Part II of this work we introduce a generalization of Gale-Stewart games, where one of the players, or even both of them, may have to choose several bits in one round. Whereas in the turn-based setting of Chapter 5 we obtain a game which is still determined, in Chapter 6 we deal with concurrent games; both players make their choices simultaneously, and determinacy is no longer guaranteed.

Chapter 2

Synopsis of Winning Conditions

A fundamental difference between various games is the complexity to solve them, with respect to both the time required to compute winning strategies and the space needed to implement these winning strategies. Whereas some winning conditions allow for positional winning strategies, others require memory, often of size exponential in the representation of the game graph or the winning condition.

The present chapter predominantly serves as an introduction to the first part of the thesis, where we deal with the problem of memory reduction, i.e., the problem of computing a winning strategy which has a memory as small as possible. In the following sections, we formally define the winning conditions we are going to consider later on; moreover, we present fundamental results from the field of infinite games which are relevant for us (see for example [GTW02]). All the types of infinite games considered in this chapter have ω-regular objectives and are played on a finite arena, if not stated differently. Hence, we can rely on determinacy [BL69].

In Section 1.4.1 we have introduced an infinite game as a tuple $\Gamma = (G, \varphi)$ where G is a finite graph and φ is a set of infinite sequences over the set V of vertices of G. In fact, we intend to deal only with winning conditions which can be represented by finite objects. More precisely, there are four distinct ways of writing down the winning conditions we consider.

- Reachability and Büchi objectives are defined in Sections 2.1.1 and 2.1.3, respectively. They are given by a subset F of V: $F \subseteq V$.

- Staiger-Wagner and Muller objectives are introduced in Sections 2.3.1 and 2.3.3, respectively. They are given by a family $\mathcal{F}$ of subsets of V: $\mathcal{F} = \{F_1, \dots, F_k\}$ with $F_i \subseteq V$.

- Request-Response and Streett objectives are presented in Sections 2.3.2 and 2.3.4, respectively. They are given by a family Ω of pairs of subsets of V: $\Omega = \{(F_1, F_1'), \dots, (F_k, F_k')\}$ with $F_i, F_i' \subseteq V$.

- Weak parity and parity objectives are defined in Sections 2.1.2 and 2.1.4, respectively. They are given by a mapping c from V into a finite set of colors: $c : V \to \{0, \dots, m\}$, for some $m \in \mathbb{N}$.

This chapter is organized as follows. Section 2.1 introduces several types of winning conditions for infinite games, all of which are solvable by positional winning strategies. In Section 2.2 we present the concept of game simulation, i.e., a technique to solve games requiring non-positional winning strategies. In Section 2.3 we define four winning conditions for which the corresponding games are usually not solvable by positional winning strategies; to each of them we apply the approach of game simulation.

2.1 Games with Positional Winning Strategies

In this section we introduce several types of infinite games which allow for positional winning strategies. The constructions presented are of particular relevance for Part I of this work. Section 2.1.4 cites some results on parity games which are also relevant for the second part of the thesis.

2.1.1 Reachability Games and Attractor Strategies

Attractor strategies are the most fundamental concept needed for solving infinite games. In their purest form, they are used to solve games with reachability objectives, and winning strategies for more complex winning conditions are usually composed of several attractor strategies. The name "attractor" strategy comes from the fact that it is defined in such a way that the play is attracted towards a particular part of the game arena, as quickly as possible. An attractor strategy is a positional strategy, which is computable by a backwards breadth-first search through the game graph.

In a *reachability game* we are given a game graph G and a designated subset $F \subseteq V$. A play ϱ is winning for Player 0 if it visits a vertex in F at least once. To solve a reachability game we inductively compute the set of vertices from where Player 0 can force the play into F. This set is called the *0-attractor of F* and denoted $\mathrm{Attr}_0(F)$; the subscript 0 indicates that we refer to Player 0. We show that the 0-attractor coincides with Player 0's winning region W_0. Then, since reachability games are determined, the complement of the 0-attractor is Player 1's winning region. To compute $\mathrm{Attr}_0(F)$ we introduce a superscript i, denoting the set of vertices from where Player 0 can force the play into F in at most i moves. It holds $\mathrm{Attr}_0^0(F) = F$ and

$$\begin{aligned}\mathrm{Attr}_0^{i+1}(F) = \mathrm{Attr}_0^{i}(F) &\cup \{v \in V_0 \mid \exists (v,v') \in E : v' \in \mathrm{Attr}_0^{i}(F)\} \\ &\cup \{v \in V_1 \mid \forall (v,v') \in E : v' \in \mathrm{Attr}_0^{i}(F)\}.\end{aligned}$$

Clearly, we have $\mathrm{Attr}_0^i \subseteq \mathrm{Attr}_0^{i+1}$ and $\mathrm{Attr}_0(F) = \bigcup \mathrm{Attr}_0^i(F)$. Since G is finite, the sequence $(\mathrm{Attr}_0^i)_{i \in \mathbb{N}}$ becomes stationary after at most $|V|$ steps, i.e., it holds $\mathrm{Attr}_0(F) = \mathrm{Attr}_0^{|V|}(F)$.

An *attractor strategy* is a strategy which reduces the distance to F in each step. By the distance of a vertex v to F we mean the minimal i such that $v \in \mathrm{Attr}_0^i(F)$. Either $v \in V_0$, which means that Player 0 can choose a successor v' of v which has shorter distance to F than v, or $v \in V_1$, which means that all successors v' of v have shorter distance to F than v. These observations yield the following winning strategy for Player 0:

from a vertex in $\mathrm{Attr}_0^{i+1}(F)$ move to a vertex in $\mathrm{Attr}_0^i(F)$.

Any strategy satisfying the above condition is winning for Player 0 in the reachability game. Algorithm 2.1 is a linear time procedure to compute the 0-attractor; it performs a backwards breadth-first search starting from F.

Algorithm 2.1 (Attractor)

Input: Reachability game $\Gamma = (G, F)$ with $G = (V, V_0, V_1, E)$ and $F \subseteq V$

Output: $\mathrm{Attr}_0(F)$ and positional winning strategy for Player 0

```
 1: $A \leftarrow F$; $M \leftarrow E$; $E_0 \leftarrow \varnothing$
 2: for all $v \in V_1 \setminus F$ do {Count number of outgoing edges}
 3:     $\mathrm{out}(v) \leftarrow |\{v' \mid (v,v') \in E\}|$
 4: end for
 5: while there exists $e = (v,v') \in M$ with $v \notin A, v' \in A$ do
 6:     $M \leftarrow M \setminus \{e\}$
 7:     if $v \in V_0$ then {From $v$, Player 0 can move into $\mathrm{Attr}_0(F)$}
 8:         $A \leftarrow A \cup \{v\}$
 9:         $E_0 \leftarrow E_0 \cup \{e\}$
10:     else
11:         $\mathrm{out}(v) \leftarrow \mathrm{out}(v) - 1$
12:         if $\mathrm{out}(v) = 0$ then {From $v$, Player 1 must move into $\mathrm{Attr}_0(F)$}
13:             $A \leftarrow A \cup \{v\}$
14:         end if
15:     end if
16: end while
17: return $(A, E_0)$
```

Note that each edge is traversed at most twice. Hence, the running time of Algorithm 2.1 is linear in $|G|$, where we set $|G| := |V| + |E|$. After termination,

A is exactly the set $\mathrm{Attr}_0(F)$ and E_0 (extended by any positional strategy from V_0-vertices in $V \setminus \mathrm{Attr}_0(F)$) is an attractor winning strategy for Player 0 from $\mathrm{Attr}_0(F)$. This yields the following observation.

Corollary 2.1. *Let $\Gamma = (G, F)$ be a reachability game. Then we can compute an attractor winning strategy for Player 0 in time $\mathcal{O}(|G|)$.*

To see how we can compute a winning strategy for Player 1, we consider Γ from his perspective. This leads us to the objective of avoiding the set F, i.e., keeping the play inside $V \setminus F$. Such a condition is called a *safety* condition, the complement of a reachability condition. Safety conditions can be characterized in terms of "bad" prefixes (see for example [BK08]); from Player 1's point of view the bad prefixes are those containing a vertex from the set F.

From each vertex $v \in V \setminus \mathrm{Attr}_0(F)$, Player 1 has a strategy to avoid the set F, i.e., Player 0 cannot force the play into F. Either it holds $v \in V_0$, which means that each outgoing edge must lead to $V \setminus \mathrm{Attr}_0(F)$, or it holds $v \in V_1$, which means that at least one outgoing edge leads to $V \setminus \mathrm{Attr}_0(F)$. Once we have computed $\mathrm{Attr}_0(F)$, we can compute a winning strategy for Player 1 from $V \setminus \mathrm{Attr}_0(F)$ in linear time. For each vertex $v \in V_1 \setminus \mathrm{Attr}_0(F)$, we have to consider each outgoing edge at most once to find a successor v' of v with $v' \in V \setminus \mathrm{Attr}_0(F)$. The following theorem summarizes the above results.

Theorem 2.2. *Let $\Gamma = (G, F)$ be a reachability game. Then, the winning region for Player 0 is $W_0 = \mathrm{Attr}_0(F)$ and the winning region for Player 1 is $W_1 = V \setminus \mathrm{Attr}_0(F)$. Both players have positional winning strategies computable in time linear in $|G|$.*

2.1.2 Weak Parity Games

In this section we introduce *weak parity games*. A weak parity condition is given by a coloring, i.e., a mapping $c : V \to \{0, \dots, m\}$ with $m \in \mathbb{N}$. We can assume w.l.o.g. that m is even. Recall that, for a play ϱ, we denote $\mathrm{Occ}(\varrho)$ the *occurrence set* of ϱ, i.e., the set of vertices occurring in ϱ at least once. A play $\varrho = v_0 v_1 v_2 \cdots$ is winning for Player 0 if the maximal color seen in ϱ is even. That means

$$\varrho \in \varphi \iff \max(\mathrm{Occ}(c(\varrho))) \text{ is even},$$

where $c(\varrho)$ is the sequence $c(v_0)c(v_1)c(v_2)\cdots$ of colors seen in the play ϱ.

The following theorem states that weak parity games can be solved in polynomial time by positional winning strategies for both players.

Theorem 2.3 ([Cha06]). *Let $\Gamma = (G, c)$ be a weak parity game. Then, the winning regions of both players and corresponding positional winning strategies can be computed in time $\mathcal{O}(m)$, where m is the number of edges of G.*

Proof. We proceed via a refined attractor construction; still, each edge has to be traversed only once. The idea is to compute for each color k (in decreasing order, starting with the maximal color m) the set A_k of vertices from where one of the players wins such that he can force a visit to a vertex of color at least k. In the first iteration we are interested in the set of all vertices from where Player 0 can reach the even color m, i.e., the set $A_m := \mathrm{Attr}_0(C_m)$, where $C_k := c^{-1}(k)$ for all $0 \leq k \leq m$. Afterwards, the set C_m can be deleted from the game graph, obtaining the subgraph G^{m-1}. (Note that the graph G^{m-1} is still a valid game graph, since it is a *trap* for Player 0 [Zie98].) On the game graph G^{m-1} we continue analogously, computing the set $A_{m-1} := \mathrm{Attr}_1^{m-1}(C_{m-1})$ for Player 1. (The superscript $m-1$ indicates that the computation is carried out on the graph G^{m-1}; note that it holds $G^m = G$.) Afterwards, we delete $\mathrm{Attr}_1^{m-1}(C_{m-1})$ from G^{m-1}, obtaining the graph G^{m-2}. On this graph we compute the set $A_{m-2} := \mathrm{Attr}_0^{m-2}(C_{m-2})$ for Player 0, and so on. In [Cha06] it is shown that it holds

$$W_0 = \bigcup_{k \text{ even}} A_k \text{ and } W_1 = \bigcup_{k \text{ odd}} A_k,$$

and that the respective attractor strategies computed in the iterations of the algorithm are positional winning strategies for the players. □

The particular procedure for solving weak parity games is fundamental for two results we are going to present later on (in Theorems 3.32 and 3.38).

2.1.3 Büchi Games

In a *Büchi game* $\Gamma = (G, F)$ we are given a designated subset $F \subseteq V$ of *final vertices*. Player 0 wins a play ϱ if a final vertex is seen again and again:

$$\varrho \in \varphi \iff \mathrm{Inf}(\varrho) \cap F \neq \varnothing$$

Büchi games are solved in two steps (cf. [Tho95, GTW02]). The first step is to compute the set of final vertices from where Player 0 can force infinitely many revisits to final vertices. We call this set the *recurrence region of F* for Player 0 and denote it $\mathrm{Recur}_0(F)$. It can be computed inductively as follows: for all i, we compute the set of final vertices from where Player 0 can force i revisits to F; each iteration requires time $\mathcal{O}(|G|)$. After at most $|V|$ iterations the sequence of computed sets becomes stationary, at $\mathrm{Recur}_0(F)$. The second step is a bit simpler. We compute the 0-attractor of $\mathrm{Recur}_0(F)$, i.e., the set $\mathrm{Attr}_0(\mathrm{Recur}_0(F))$. The latter set coincides with the winning region W_0 of Player 0; the union of the attractor strategy to reach $\mathrm{Recur}_0(F)$ and the strategy to revisit $\mathrm{Recur}_0(F)$ again and again is a positional winning strategy for Player 0 from W_0.

Computing a winning strategy for Player 1 is a bit more involved. Once we have computed W_0, we also have $W_1 = V \setminus W_0$ in our hands. However, Player 1 needs to ensure that a final vertex is visited only finitely often. Such an objective is called a *co-Büchi* condition, the complement of a Büchi condition. The problem is that W_1 may contain final vertices, and we have to avoid them from a certain point onwards. To get the solution, we first solve a subgame on W_1, where Player 0 has the reachability objective to reach F, and Player 1 has to fulfill the safety condition to stay inside $W_1 \setminus F$. Let W_1' be the winning region of Player 1 in this subgame. Due to the fact that Player 1 wins the Büchi game from W_1 (on G), the set W_1' must be non-empty, and he has a positional winning strategy that never leaves W_1' (cf. Theorem 2.2). Moreover, he must have an attractor strategy from all vertices in W_1 which forces a play to reach W_1' in at most $|V|$ moves. Putting together the winning strategy from W_1' for the subgame and an attractor strategy to reach W_1' (on G) we obtain a positional winning strategy for Player 1 from W_1 in the Büchi game.

Theorem 2.4. *Let $\Gamma = (G, F)$ be a Büchi game on a graph with n vertices and l edges. The winning regions are $W_0 = \mathrm{Attr}_0(\mathrm{Recur}_0(F))$ and $W_1 = V \setminus \mathrm{Attr}_0(\mathrm{Recur}_0(F))$, and corresponding positional winning strategies for both players can be computed in time $\mathcal{O}(n \cdot l)$.*

2.1.4 Parity Games

A *parity game* is an infinite game where each vertex of G is assigned one of finitely many colors. The winning condition is defined analogously to the acceptance condition of a DPA: a play ϱ is winning for Player 0 if $\max(\mathrm{Inf}(c(\varrho)))$, i.e., the maximal color seen infinitely often, is even.

Parity games on finite graphs with finitely many colors are determined with positional winning strategies for both players; this result was independently proven in [EJ91] and [Mos91]. In Part II of the present thesis we come across parity games on countably infinite graphs where the number of colors remains finite. For such games, the result of Emerson & Jutla and Mostowski holds as well. (In this work we do not consider parity games with infinitely many colors; the interested reader is referred to [GW06].)

Many efforts have been made to find algorithmic solutions for parity games (see for example [Jur00, VJ00]). Although it is known that the problem of deciding the winner in a parity game is in UP $\cap$ co-UP [Jur98], the search for a tight lower bound for the computational complexity has not been successful yet (see for example [Fri09]). In particular, it is not known whether parity games can be solved in polynomial time. The fastest deterministic algorithm presently known has a running time which is subexponential in the number of vertices of the game graph [JPZ06].

Theorem 2.5. *Let $\Gamma = (G, c)$ be a parity game on a finite or countably infinite graph, with finitely many colors. Each vertex belongs to either W_0 or W_1. Moreover, Player 0 and Player 1 have positional winning strategies from W_0 and W_1, respectively. If G is finite, then one can compute the winning regions and corresponding positional winning strategies in time at most $n^{\mathcal{O}(\sqrt{n})}$, where n is the number of vertices of the game graph.*

2.2 Game Simulation

The key idea of *game simulation* is to reduce the solution of a given game to the solution of a new one which has a larger game graph but a simpler winning condition [Tho02]. We have as input an infinite game $\Gamma = (G, \varphi)$ and obtain as output an infinite game $\Gamma' = (G', \varphi')$. The game Γ' is computed from Γ such that G' consists of finitely many copies of G. Each copy memorizes a particular history of a play in the original game and the new edge relation comprises the memory update rule. Whereas the expanded game graph G' is more complex than G, the winning condition φ' (for Player 0) is much easier than the given one. In many cases, φ requires memory (of exponential size), but φ' admits positional winning strategies. Therefore, the complexity of the given winning condition φ is diverted to the game graph G'.

Definition 2.6. Let $\Gamma = (G, \varphi)$ and $\Gamma' = (G', \varphi')$ be infinite games with game graphs $G = (V, V_0, V_1, E), G' = (V', V_0', V_1', E')$ and winning conditions φ, φ'. We say that Γ is *simulated* by Γ' (short: $\Gamma \leq \Gamma'$) if the following hold:

1. $V' = S \times V$ for a finite memory set S; for $i \in \{0, 1\}$ and all $s \in S, v \in V$ it holds $v \in V_i \iff (s, v) \in V_i'$
2. Every play ϱ of Γ is transformed into a unique play ϱ' of Γ' by
   a) $\exists s_0 \in S, \forall v \in V$: $\varrho(0) = v \Longrightarrow \varrho'(0) = (s_0, v)$
   b) Let $(s, v) \in V'$:
      i. $(v, v') \in E \Longrightarrow \exists s' \in S : ((s, v), (s', v')) \in E'$
      ii. $((s, v), (s_1, v_1)), ((s, v), (s_2, v_2)) \in E' \Longrightarrow s_1 = s_2$
   c) $((s, v), (s', v')) \in E' \Longrightarrow (v, v') \in E$
3. ϱ is winning for Player 0 in $\Gamma \iff \varrho'$ is winning for Player 0 in Γ'

What one should especially note in the above definition is item 2(b)ii: from a given vertex (s, v) all outgoing edges lead to vertices with the same memory component. This means that the memory update is uniquely determined by s and v and, thus, depends only on the source vertex of an edge. For this reason, we need the unique initial memory content s_0 (cf. item 2a). The above definition of game simulation is chosen due to particular technical reasons connected with the algorithms we are going to present later on.[1]

Once we have applied a game simulation, we can compute winning strategies for both players from their respective winning region in Γ. Let us explain this exemplarily for Player 0: we solve the game Γ' and obtain a finite-state winning strategy f' from Player 0's winning region W_0' in Γ'; the game simulations we are going to consider in Section 2.3 in fact yield games with positional winning strategies. From that we can construct a winning strategy f for Player 0 in Γ; it is implemented by the strategy automaton $\mathcal{A}_f$ constructed as follows: each state of $\mathcal{A}_f$ corresponds to one of the G-copies in G', and the transition function σ is obtained from the edge relation E' of G'. The output function τ of $\mathcal{A}_f$ is uniquely determined by the strategy f'. If f' is a winning strategy for Player 0 in Γ', then $\mathcal{A}_f$ implements a winning strategy for her in Γ.

Theorem 2.7. *Let $\Gamma = (G, \varphi)$ and $\Gamma' = (G', \varphi')$ be infinite games such that $\Gamma \leq \Gamma'$ according to Definition 2.6. If Player 0 wins Γ' from vertex (s_0, v) by a positional winning strategy f', then she wins Γ from vertex v by an automaton strategy f.*

Proof. We prove the above theorem by constructing the strategy automaton $\mathcal{A}_f = (S, s_0, \sigma, \tau)$ over V, where S, V and s_0 are already given by the definition of game simulation. We define σ and τ as follows:

$$\begin{array}{ll} \sigma : S \times V \to S & \sigma(s, v) := s' \text{ where } s' \text{ is the memory content uniquely} \\ & \text{determined by } s \text{ and } v \\ \tau : S \times V_0 \to V & \tau(s, v_0) := v' \text{ where } v' \text{ is the second component of} \\ & \text{the unique vertex } (s', v') \text{ with } ((s, v_0), (s', v')) \in E_{f'} \end{array}$$

$\mathcal{A}_f$ gets as input the vertices visited in G and changes states accordingly. For every game position $v_0 \cdots v_i$ of Player 0 it outputs some E-successor v_{i+1} of v_i. Finally, the choices of Player 1 induce a play ϱ on G which, together with the memory contents, forms a play ϱ' winning for Player 0 in Γ'. Since Γ is simulated by Γ' it follows from item 3 of Definition 2.6 that ϱ is winning for Player 0 in Γ. □

[1]There are other equivalent definitions of game simulation where the memory update also depends on the target vertex of an edge.

With the above technique we can solve games which do not allow for positional winning strategies in general. (The constructions in the proof of Theorem 2.7 work analogously for Player 1.) The winning regions W_0, W_1 in Γ can be read off directly from the winning regions W_0', W_1' in Γ'. For $i \in \{0, 1\}$ and $v \in V$, vertex v belongs to W_i if and only if vertex (s_0, v) belongs to W_i'.

2.3 Games with Non-Positional Winning Strategies

There are games which usually do not admit a solution by positional winning strategies. The purpose of this section is to introduce those to which we apply our memory reduction algorithm in Part I of the thesis.

The first type of games we shall consider are the ones induced by "weak" winning conditions, or *Staiger-Wagner* conditions, as we call them (cf. [SW74]). In the associated games the winner of a play is declared on the basis of the set of all vertices occurring in the play (at least once). The second class of conditions we are going to deal with are called *Request-Response* conditions (see [WHT03]). They do not depend on the set of occurring vertices but rather on their particular order. In addition, we present results on two standard forms for ω-regular objectives, namely *Muller* (see for example [GH82, DJW97]) and *Streett* (see for example [BLV96, Hor05]). Each of the two depends on the vertices visited infinitely often and, accordingly, is also referred to as a "strong" winning condition.

2.3.1 Staiger-Wagner Games

In a *Staiger-Wagner game* $\Gamma = (G, \mathcal{F})$ we are given a family $\mathcal{F} \subseteq 2^V$ where V is the set of vertices of the given game graph. A play ϱ is winning for Player 0 if $\mathrm{Occ}(\varrho) \in \mathcal{F}$ (cf. weak Muller games in [Tho02]). Note that a Staiger-Wagner condition is a Boolean combination of reachability and safety conditions.

A lower bound for the size of the memory required to win Staiger-Wagner games is exponential in the size of the game graph.

Theorem 2.8. *There is a family $\Gamma_n = (G_n, \mathcal{F}_n)$ of Staiger-Wagner games where the size of G_n is linear in n and each winning strategy from a particular vertex v_1 requires a memory of size at least 2^n.*

Proof. The game graph G_n is depicted in Figure 2.1. Circle vertices belong to Player 0 and square vertices belong to Player 1. The graph has $6n + 2 \in \mathcal{O}(n)$

vertices and $8n + 2 \in \mathcal{O}(n)$ edges. Thus, the size of G_n is linear in n. The winning condition $\mathcal{F}_n$ contains precisely the sets $F \subseteq V$ which satisfy

$$i \in F \iff i' \in F, \text{ for all } 1 \leq i \leq n.$$

Each play is divided in two phases: in the first phase (between vertices v_1 and x) Player 1 selects a subset $P \subseteq \{1, \dots, n\}$ of vertices. In the second phase (between vertices $v_{1'}$ and y) Player 0 selects a subset $R \subseteq \{1', \dots, n'\}$ of vertices, analogously. Player 0 wins if she mimics the behavior of Player 1, i.e., it holds $i \in P \iff i' \in R$ for all $i \in \{1, \dots, n\}$.

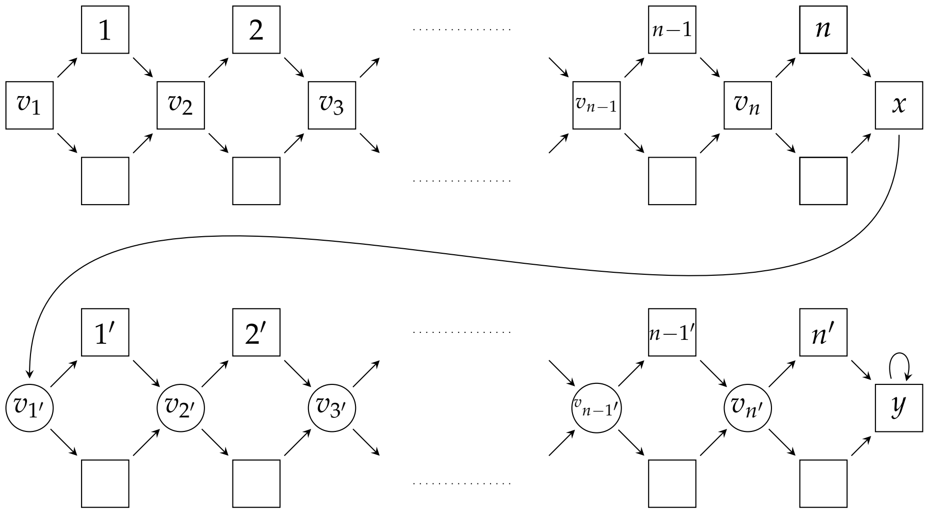

Figure 2.1: *Staiger-Wagner game graph G_n*

Clearly, Player 0 wins Γ_n by memorizing the vertices in $\{1, \dots, n\}$ which are visited in the first phase and by moving from $v_{1'}$ accordingly. Since the set $\{1, \dots, n\}$ has 2^n subsets, each winning strategy needs at least 2^n states to be implemented by a strategy automaton. A simple argument shows that no automaton with less than 2^n states implements a winning strategy for Player 0. □

We apply the technique of game simulation to solve Staiger-Wagner games. More precisely, we simulate them by weak parity games, using a memory of size 2^n where n is the number of vertices of the game graph.

To win Staiger-Wagner games, it suffices to store for each vertex of the game arena whether it has already been visited or not. Consequently, it is straightforward to use as memory the set of possible occurrence sets; we define the memory to be the set $S := 2^V$. While playing, we collect the visited states such that at any given point in a play, i.e., at some game position $\varrho(0)\cdots\varrho(i)$, the current memory content is $\mathrm{Occ}(\varrho(0)\cdots\varrho(i))$. The game graph $G' = (V', E')$ has as vertices the set $V' := 2^V \times V$. If v_0 is the initial vertex of Γ, then $(\varnothing, v_0)$ is the initial vertex of Γ'. Following Definition 2.6, we update the memory according to the source vertex of an edge; the new edge relation is $E' := \{((R,v),(R\cup\{v\},v')) \mid (v,v') \in E, R \subseteq V\}$. Thereby, we are done with items 1 and 2 of Definition 2.6. What is still missing is the winning condition φ'. For that we define the coloring c as follows:

$$c(R,v) := \begin{cases} 2\cdot|R\cup\{v\}|-1 & \text{, if } R\cup\{v\} \notin \mathcal{F} \\ 2\cdot|R\cup\{v\}| & \text{, if } R\cup\{v\} \in \mathcal{F} \end{cases}$$

Let us explain how a play ϱ of Γ is transformed into a play ϱ' of Γ'. During ϱ', the sequence of occurrence sets in the visited vertices is weakly increasing with respect to the subset relation $\subseteq$. This is due to the fact that we only add vertices to the R-component, but never delete some. Therefore, the sequence of visited colors is weakly increasing (with respect to $\leq$) and it becomes stationary as soon as the memory component has reached $\mathrm{Occ}(\varrho)$, i.e., no new V-vertices are visited afterwards. From that point onwards, the unique maximal color is being seen for the rest of the play and it is even if and only if $\mathrm{Occ}(\varrho) \in \mathcal{F}$. This means that ϱ is winning for Player 0 in Γ if and only if ϱ' is winning for Player 0 in Γ', verifying item 3 of Definition 2.6.

Theorem 2.9. *Let $\Gamma = (G, \mathcal{F})$ be a Staiger-Wagner game where G has n vertices. Then there exists a weak parity game $\Gamma' = (G', c)$ such that G' has $2^n \cdot n$ vertices and $\Gamma \leq \Gamma'$.*

Note that Staiger-Wagner games are symmetric, i.e., both players have to satisfy Staiger-Wagner conditions in order to win. Accordingly, winning strategies for both players require memory.

By Theorems 2.9 and 2.3, Staiger-Wagner games are solvable in time exponential in the number of vertices of the game graph.

Corollary 2.10. *Let $\Gamma = (G, \mathcal{F})$ be a Staiger-Wagner game. The game Γ is determined. One can compute the winning regions W_0, W_1 and automaton winning strategies for both players in time exponential in $|G|$.*

2.3.2 Request-Response Games

In a *Request-Response game* the winning condition is given by a family Ω of pairs of subsets of V: $\Omega = \{(P_1, R_1), \ldots, (P_k, R_k)\}$, where $P_j, R_j \subseteq V$ for $1 \leq j \leq k$, with $k \in \mathbb{N}$. Each set P_j is called "request"-set and each set R_j is called "response"-set. Player 0 wins a play ϱ if, for all j, each visit to the set P_j is eventually followed by a visit to the set R_j.

A lower bound for the memory required to solve Request-Response games is exponential in the number k of request-response pairs. For a proof of this result we refer to [WHT03].

Theorem 2.11 ([WHT03])**.** *There is a family $\Gamma_k = (G_k, \Omega_k)$ of Request-Response games such that the following hold:*

1. *Both the size of G_k and the number of request-response pairs are linear in k.*
2. *Player 0 has a winning strategy from a particular vertex v_0, and every winning strategy from v_0 requires a memory of size at least $2^k \cdot k$.*

The rest of this section is devoted to a game simulation of Request-Response games by Büchi games. The idea is to memorize the set of active request-response pairs. We say that a pair (P_j, R_j) is *active* at a given point in a play, if the set P_j has been visited before and the latest visit to P_j has not yet been responded to. We cyclically[2] test whether all requests are eventually responded to. If the test is successful for a particular pair (P_j, R_j), i.e., it is either not active or it has been active but was responded to in the latest move, then we proceed to the next pair. If the test for the last pair (P_k, R_k) is successful, then a final vertex is visited and we start over again with the first pair (P_1, R_1). If some request is never responded to, i.e., the respective pair remains active from a certain point onwards, then no further final vertex is seen. Let $G = (V, E)$ and $\Omega = \{(P_1, R_1), \ldots, (P_k, R_k)\}$. We define $G' := (V', E')$ as follows:

- $V' := 2^{\{1,\ldots,k\}} \times \{1, \ldots, k\} \times \mathbb{B} \times V$
- $((P, j, b, v), (P', j', b', v')) \in E' :\Longleftrightarrow$
  - $P' = (P \cup \{j \mid v \in P_j\}) \setminus \{j \mid v \in R_j\}$
  - $j' = \begin{cases} j & \text{, if } j \in P' \\ (j \bmod k) + 1 & \text{, otherwise} \end{cases}$
  - $b' = \begin{cases} 1 & \text{, if } j = k \text{ and } j' = 1 \\ 0 & \text{, otherwise} \end{cases}$
  - $(v, v') \in E$

[2]If there is only one request-response pair in Ω, then we need to add a dummy pair $(P_2, R_2) = (\varnothing, \varnothing)$ to detect a completed cycle.

- $F := 2^{\{1,\ldots,k\}} \times \{1,\ldots,k\} \times \{1\} \times V$

As initial memory content we choose $s_0 := (\varnothing, 1, 0)$ (cf. item 2a of Definition 2.6). The winning condition φ' is uniquely determined by the set F. The reader may verify that the above definitions of G' and F satisfy the properties of game simulation.

Theorem 2.12. *([WHT03]) Let $\Gamma = (G, \Omega)$ be a Request-Response game where G has n vertices and Ω has k request-response pairs. Then there exists a Büchi game $\Gamma' = (G', F)$ such that G' has $2^{k+1} \cdot k \cdot n$ vertices and $\Gamma \leq \Gamma'$.*

It is not hard to see that Player 1 has a positional winning strategy from his winning region. If he plays a winning strategy from some vertex $v \in W_1$, then there must exist $1 \leq j \leq k$ such that eventually the pair (P_j, R_j) is activated, but never responded to afterwards. Note that the choice of j may be influenced by Player 0. For the activation of (P_j, R_j) Player 1 can play an attractor strategy, and for keeping (P_j, R_j) active he has to avoid R_j, i.e., satisfy a safety condition.

By Theorems 2.12 and 2.4, Request-Response games are solvable in time exponential in the number of request-response pairs of the winning condition.

Corollary 2.13. *Let $\Gamma = (G, \Omega)$ be a Request-Response game. The game Γ is determined. One can compute the winning regions W_0, W_1 and automaton winning strategies for both players in time exponential in $|\Omega|$ and polynomial in $|G|$.*

2.3.3 Muller Games

In a *Muller game* we are given a family $\mathcal{F} = \{F_1, \ldots, F_k\}$ of subsets of V: $F_j \subseteq V$ for all $1 \leq j \leq k$ where $k \in \mathbb{N}$. A play ϱ is winning for Player 0 if the set of vertices visited infinitely often, i.e., the set $\mathrm{Inf}(\varrho)$, is one of the sets in $\mathcal{F}$. Note that a Muller condition is a Boolean combination of Büchi and co-Büchi conditions.

A lower bound for the memory needed to implement winning strategies for Muller games is $n!$, if $\mathcal{O}(n)$ is the size of the game graph.

Theorem 2.14 ([DJW97])**.** *There is a family $\Gamma_n = (G_n, \mathcal{F}_n)$ of Muller games such that the number of vertices of G_n is linear in n and each winning strategy for Player 0 requires a memory of size at least $n!$.*

Proof. The graph G_n is depicted in Figure 2.2. The set of vertices of the game graph is $\{v_1, \ldots, v_n, v'_1, \ldots, v'_n\}$, $V_0 := \{v_1, \ldots, v_n\}$ and $V_1 := \{v'_1, \ldots, v'_n\}$ are the sets of vertices belonging to Player 0 and Player 1, respectively, and the set of edges is $\{(v_j, v'_l), (v'_l, v_j) \mid 1 \leq j, l \leq n\}$. The graph G_n has $2n \in \mathcal{O}(n)$ vertices. The players move in alternation where in each move the current player is free to move to any of the vertices belonging to the opponent. The Muller winning condition φ_n is induced by $\mathcal{F}_n$ with

$$\mathcal{F}_n := \{F \subseteq V \mid |F \cap V_0| = \max\{j \mid v'_j \in F \cap V_1\}\}.$$

The definition of the winning condition means that the number of V_0-vertices visited infinitely often is equal to the maximal index j such that V_1-vertex v'_j is visited infinitely often.

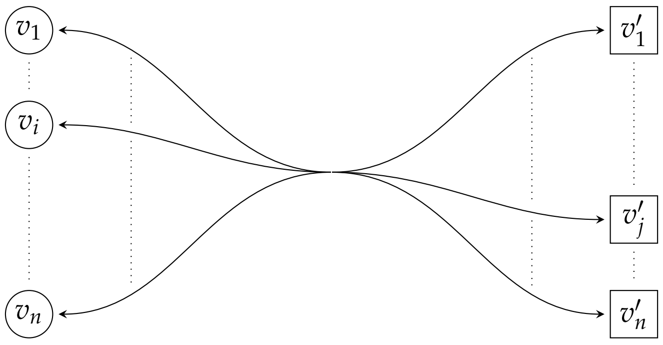

Figure 2.2: *Muller game graph G_n*

Dziembowski et al. have shown that Player 0 wins from each vertex of G_n and that each winning strategy requires a memory of size at least $n!$. The proof is accomplished by induction on n and can be found in [DJW97]. □

We present a game simulation of Muller games by parity games (see for example [GTW02]). As a start, we introduce the *Latest Appearance Record* (LAR), which we are going to use as memory (see for example [McN65, GH82]). If $V = \{v_1, \dots, v_n\}$ is the set of vertices of the game graph, then we define $\mathrm{LAR}(V) := \mathcal{S}_n \times \{1, \dots, n\}$ where $\mathcal{S}_n$ denotes the symmetric group of all permutations of the set $\{1, \dots, n\}$. We use the LAR to memorize the order of latest visits to the vertices of the game graph. A play ϱ of the Muller game Γ is transformed into a play ϱ' of the parity game Γ' as follows. As initial memory content we choose $s_0 := (1 \cdots n, 1)$. Let $r = (j_1 \cdots j_n, h)$ be the current memory content and v_{j_l} the current vertex $(1 \le l \le n)$; we call h the *hit value* of r. If a move from v_{j_l} to v_{j_m} is made, then j_l is shifted to the first position and the hit value is set to l.

We define the parity game graph $G' = (V', E')$ as follows:

- $V' := \mathrm{LAR}(V) \times V$
- For all $(j_1 \cdots j_n, h) \in \mathrm{LAR}(V), l, m \in \{1, \dots, n\}$ with $(v_{j_l}, v_{j_m}) \in E$ there is an edge $(((j_1 \cdots j_n, h), v_{j_l}), ((j_l j_1 \cdots j_{l-1} j_{l+1} \cdots j_n, l), v_{j_m})) \in E'$.

The parity winning condition is induced by the following coloring:

$$c((j_1 \cdots j_n, h), v_{j_l}) := \begin{cases} 2h - 1 & \text{, if } \{v_{j_1}, \ldots, v_{j_h}\} \notin \mathcal{F} \\ 2h & \text{, if } \{v_{j_1}, \ldots, v_{j_h}\} \in \mathcal{F} \end{cases}$$

The above game simulation of Muller games by parity games is correct. In other words, a play ϱ in Γ is winning for Player 0 if and only if the corresponding play ϱ' in Γ' is winning for Player 0. For a full proof we refer to [GTW02].

Theorem 2.15. *Let $\Gamma = (G, \mathcal{F})$ be a Muller game where G has n vertices. Then there exists a parity game $\Gamma' = (G', c)$ such that G' has $n! \cdot n^2$ vertices and $\Gamma \leq \Gamma'$.*

From Theorem 2.14 and the fact that Muller games are symmetric it follows that both players require memory (of possibly factorial size) in order to win. By Theorems 2.15 and 2.5, a solution to a Muller game can be computed.

Corollary 2.16. *Let $\Gamma = (G, \mathcal{F})$ be a Muller game. The game Γ is determined. One can compute the winning regions W_0, W_1 and automaton winning strategies for both players.*

So far we have spared to mention the computational complexity of Muller games. This issue is caught up in the following paragraph.

Excursus on the Complexity of Muller Games. The time needed to solve an infinite game is measured in the size of its representation. For Muller games, the literature has suggested a number of different ways for writing down the winning condition, each of which has a different degree of succinctness. In this work, we choose the *explicit* representation, i.e., the sets in $\mathcal{F}$ are all listed. For this type, Horn has shown that the winning regions of a Muller game can be computed in polynomial time [Hor08].

A more succinct formalism is due to Zielonka [Zie98]; he introduced the notion of a *split tree*, mostly referred to in the literature as *Zielonka tree*. It is a rooted tree in which each node is labeled by a subset of vertices of the game graph such that the node is owned by the player for whom the label is winning, and each child of a node is labeled by a subset (of the parent-label) winning for the other player. Zielonka trees are a commonly used memory structure for the implementation of winning strategies in Muller games (see for example [DJW97]). In [HD05], Hunter and Dawar have introduced *Zielonka DAGs* which are Zielonka trees with nodes labeled by the same set merged. Whereas Zielonka trees allow for a solution of the corresponding Muller game in NP $\cap$ co-NP [DJW97], Zielonka DAGs are more succinct such that deciding the winner becomes Pspace-complete.

Other ways of writing down Muller winning conditions, for which [HD05] has established Pspace-completeness for deciding the winner, are *win-sets* and their variant *colorings*, and *Emerson & Lei* formulas; for detailed definitions of these conditions, the reader is referred to [McN93, EL85].

2.3.4 Streett Games

Streett games capture the notion of *strong fairness* in distributed systems. For example, if a process is repeatedly requesting a particular resource, then it should eventually be granted access to that resource. This means that infinitely many requests are fulfilled infinitely often; finitely many requests can be ignored.

A Streett condition is a family $\Omega = \{(E_1, F_1), \ldots, (E_k, F_k)\}$ of pairs of subsets $E_j, F_j \subseteq V$ for $1 \leq j \leq k$, with $k \in \mathbb{N}$. A play ϱ is winning for Player 0 if, for each $1 \leq j \leq k$ such that the set F_j is visited infinitely often, the set E_j is also visited infinitely often:

$$\forall j (\mathrm{Inf}(\varrho) \cap F_j \neq \varnothing \Longrightarrow \mathrm{Inf}(\varrho) \cap E_j \neq \varnothing)$$

A Streett condition is a special kind of Muller condition. If we express Ω by means of an equivalent family $\mathcal{F}$ of subsets of V, then $\mathcal{F}$ is closed under union (see [Zie98]).

Winning strategies in Streett games may require a memory of size at least $k!$, i.e., factorial in the number of Streett pairs, if both the game graph and the winning condition are of size quadratic in k.

Theorem 2.17 ([Hor05]). *There is a family $\Gamma_k = (G_k, \Omega_k)$ of Streett games such that, for $k \geq 2$, both the size of G_k and the number of pairs in Ω_k are quadratic in k, and each winning strategy for Player 0 needs a memory of size at least $k!$.*

Proof. The game graph G_k is depicted in Figure 2.3. It has the set V_k of vertices with $V_k := \{v, w\} \cup \{i, \{i, j\} \mid 1 \leq i \neq j \leq k\}$, and the set

$$E_k := \{(v, \{i, j\}), (\{i, j\}, i), (i, w) \mid 1 \leq i \neq j \leq k\} \cup \{(w, v)\}$$

of edges.[3] That means there are $\mathcal{O}(k^2)$ vertices and $\mathcal{O}(k^2)$ edges. The winning condition contains $k^2 - k \in \mathcal{O}(k^2)$ pairs; it is defined as

$$\Omega = \{(\{i\}, \{\{i, j\}\}) \mid 1 \leq i \neq j \leq k\}.$$

[3] Note that $(\{i, j\}, i)$ represents two edges.

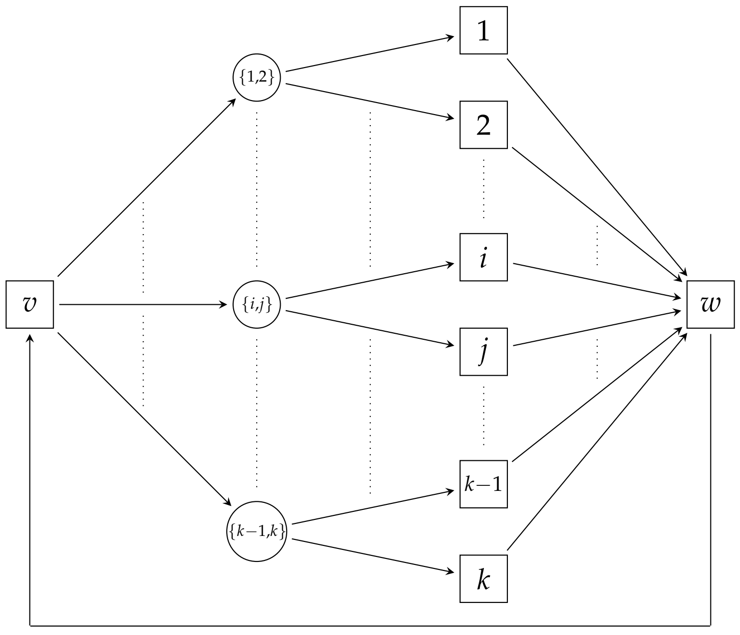

Figure 2.3: *Streett game graph G_k*

The game starts at vertex v and proceeds in rounds as follows: Player 1 chooses a vertex $\{i, j\}$, thereby activating the two conditions i and j. Afterwards, Player 0 answers precisely one of them, moving to either vertex i or vertex j. She has the following natural winning strategy of size $k!$: each memory content is a permutation $\pi = (\pi_1 \cdots \pi_k) \in \mathcal{S}_k$, where $\mathcal{S}_k$ denotes the symmetric group of all permutations of the set $\{1, \ldots, k\}$; each π_0 can be taken as initial memory content. If Player 1 moves to vertex $\{i, j\}$, then Player 0 answers by moving to the vertex which appears first in π and shifts it to the very last position. That means, π is a priority queue: if $\pi_r = i$ and $\pi_s = j$ and $r < s$, then Player 0 moves to vertex i and the new memory content is $(\pi_1 \cdots \pi_{r-1}\pi_{r+1} \cdots \pi_k\pi_r)$. It is not difficult to see that this strategy is winning for Player 0. If a condition is requested but not immediately answered, then it is shifted in π by one position to the front. After at most k unanswered requests it reaches the first position of π, which means that it is answered next time it is requested. Thus, if it is requested infinitely often, then it answered infinitely often.

In fact, the aforementioned winning strategy for Player 0 is optimal in the sense that there exists no winning strategy for her with less memory. The proof is accomplished by induction on k and can be found in [Hor05]. □

We present a game simulation of Streett games by parity games (see [MS95, BLV96]), using almost the same idea as for Muller games. We use the *Index Appearance Record* (IAR) to memorize the order of latest visits to the sets E_j ($j = 1, \ldots, k$). Let $G = (V, E)$ be a game graph and $\Omega = \{(E_1, F_1), \ldots, (E_k, F_k)\}$ a family of pairs of subsets of V. As memory S we define:

$$S := \mathrm{IAR}(\Omega) = \{(j_1 \cdots j_k, e, f) \mid (j_1 \cdots j_k) \in \mathcal{S}_k, 1 \leq e, f \leq k\}$$

As initial memory content we choose $s_0 := (1 \cdots k, 1, 1)$. The edge relation E' of the parity game graph $G' = (V', E')$ is uniquely determined by the edge relation E and Ω. We define:
$(((j_1 \cdots j_k, e, f), v), ((j'_1 \cdots j'_k, e', f'), v')) \in E' :\Longleftrightarrow$

1. $(v, v') \in E$
2. $(j'_1 \cdots j'_k)$ is obtained from $(j_1 \cdots j_k)$ by shifting all j_l with $v \in E_{j_l}$ to the left, $l \in \{1, \ldots, k\}$
3. e' is the maximal[4] $l \in \{1, \ldots, k\}$ such that $v \in E_{j_l}$
4. f' is the maximal $m \in \{1, \ldots, k\}$ such that $v \in F_{j'_m}$

Let the coloring $c : S \times V \to \{1, \ldots, 2k\}$ be defined by:

$$c((j_1 \cdots j_k, e, f), v) := \begin{cases} 2e & \text{, if } e \geq f \\ 2f - 1 & \text{, if } e < f \end{cases}$$

Correctness of the above game simulation is established in [MS95].

Theorem 2.18. *Let $\Gamma = (G, \Omega)$ be a Streett game where G has n vertices and Ω has k pairs. Then there exists a parity game $\Gamma' = (G', c)$ such that G' has $k! \cdot k^2 \cdot n$ vertices and $\Gamma \leq \Gamma'$.*

In a Streett game, a play ϱ is winning for Player 1 if it violates the given Streett condition, i.e., it satisfies the following condition:

$$\exists j (\mathrm{Inf}(\varrho) \cap E_j = \varnothing \wedge \mathrm{Inf}(\varrho) \cap F_j \neq \varnothing)$$

This kind of condition is called *Rabin* condition. It can always be fulfilled by a positional strategy, though, such a strategy is not obtained when solving the game by game simulation via IAR. In [Hor05], Horn gives an algorithm for solving Streett games which computes a positional winning strategy for the Rabin player and has a running time exponential in k.

[4]We assume w.l.o.g. that $E_k = F_k = V$ to have the pointers e' and f' well-defined.

Corollary 2.19. *Let $\Gamma = (G, \Omega)$ be a Streett game. The game Γ is determined. One can compute the winning regions W_0, W_1 and both an automaton winning strategy for Player 0 and a positional winning strategy for Player 1 in time exponential in $|\Omega|$ and polynomial in $|G|$.*

Part I

Memory Reduction for Strategies in Infinite Games

Chapter 3

An Algorithm for Memory Reduction

The purpose of this chapter is to propose an algorithm which reduces the memory necessary for the implementation of winning strategies in infinite games.

The design and analysis of algorithms for the computation of winning strategies is a central task in the field of controller synthesis. The basic result in this area is the Büchi-Landweber Theorem, which says that each regular infinite game on a finite arena is determined and a finite-state winning strategy for the winner can be constructed [BL69]. Besides studying the algorithmic complexity of solving such games (see for example [HD05]), many results in the past decades have revealed both exponential lower and upper bounds for the size of the memory required to implement winning strategies [DJW97, Zie98, GTW02, Hor05]. The aim of the present chapter is not the analysis or extension of the aforementioned findings. Rather, we deal with the problem of algorithmic reduction of the used memory, for multiple (sub)classes of regular conditions.

A finite-state strategy can be reduced by minimizing a strategy automaton implementing it. Even though this can be done very efficiently, there are two strong arguments why this technique is inappropriate for memory reduction. The first one is that minimization of an automaton reduces only the representation of a strategy, but it disregards other possible winning strategies which may require less memory. The second reason is that the resulting automaton is minimal with respect to the output function as a whole, but the implemented strategy is only a partial function. Clearly, the size of existing winning strategies depends only on the underlying game graph and the winning condition. Therefore, we intend to establish a method for memory reduction which is independent of automata minimization or the representation of particular strategies.

The starting point of our new algorithm is the concept of *game simulation*, which has been presented in Section 2.2 (see also [Tho02]). Recall that the key idea of game simulation is to reduce the solution to a given game to the solution of a new one which has an extended game graph and an easier winning condition. The new game graph contains enough memory to solve the original game. The basic idea of our memory reduction algorithm is to simplify the extended game graph before the computation of winning strategies for the two players. This is accomplished by state space reduction of an equivalent ω-automaton. In this context, equivalence means that the automaton accepts precisely the language of all plays winning for Player 0 in the given game. The reduction is carried out such that the properties of game simulation are preserved. Our algorithm computes an equivalence relation on the set of vertices of the extended game graph and from that infers equivalent memory contents.

This chapter is organized as follows. We start Section 3.1 by giving a review of a minimization technique for finite automata with output. Moreover, we explain the shortcomings of using this approach for memory reduction of finite-state strategies. In Section 3.2 we present our new algorithm and formally prove its correctness. In Sections 3.3 through 3.5 we apply our method to four types of winning conditions: Staiger-Wagner, Request-Response, Muller and Streett. Section 3.6 serves as a discussion of the advantages and disadvantages of our approach; it also comprises computational results on an implementation of our technique.

3.1 Retrospection: Mealy Automata

The minimization problem for DFA is the problem, for a given DFA $\mathcal{A}$, to compute a DFA $\mathcal{A}'$ such that it holds $L(\mathcal{A}') = L(\mathcal{A})$ and $\mathcal{A}'$ is the minimal automaton with this property. This problem is well-understood and known to be solvable in time $\mathcal{O}(n \cdot \log n)$, if n is the number of states of $\mathcal{A}$ and $|\Sigma|$ is assumed constant [Hop71, Gri73]. The automaton $\mathcal{A}'$ is uniquely determined (up to isomorphism).

In this section we deal with minimization of *Mealy automata* [Mea55], i.e., deterministic finite automata with output. Mealy automata are very similar to usual DFA. Instead of a set F of final states, they are equipped with an output function τ assigning to each transition one letter of a given output alphabet. The output for a particular input word w is the concatenation of the output letters assigned to the transitions taken by $\mathcal{A}$ during its run on w. Accordingly, two states are declared equivalent if from both the two states the automaton computes the same output, for all possible input words. Two

Mealy automata $\mathcal{A}, \mathcal{A}'$ are said to be equivalent if they compute the same output from their initial states. To get this clearer, let us first formally define a Mealy automaton.

Definition 3.1. A *Mealy automaton* over Σ, Σ_O has the form $\mathcal{A} = (Q, q_0, \delta, \tau)$ where Q, Σ, q_0 and δ are defined as for usual DFA, Σ_O is an *output alphabet* and $\tau : Q \times \Sigma \to \Sigma_O$ is called *output function*. The *output* $f_\mathcal{A} : \Sigma^* \to \Sigma_O^*$ *computed by* $\mathcal{A}$ is inductively defined as $f_\mathcal{A}(\varepsilon) := \varepsilon$, and for all $w \in \Sigma^*, a \in \Sigma$ as

$$f_\mathcal{A}(wa) := f_\mathcal{A}(w)\tau(\delta^*(q_0, w), a).$$

The upcoming theorem shows that minimization of Mealy automata can be done analogously to minimization of standard DFA. The algorithm is a refined block partitioning approach. The partition is initialized such that two states q, q' are in distinct blocks if and only if there is a letter a such that $\tau(q, a) \neq \tau(q', a)$. The blocks are refined iteratively until there are no more two states in the same block from which some letter $a \in \Sigma$ leads into different blocks. To get the same asymptotic running time as in minimization of standard DFA, we need to make the additional assumption that $|\Sigma_O|$ is taken as a constant; then, we can minimize a given Mealy automaton in time $\mathcal{O}(n \cdot \log n)$, obtaining an equivalent one which is uniquely determined (up to isomorphism). The algorithm and both the proof of correctness and complexity work analogously to the one for DFA [Gri73, Bra84].

Theorem 3.2. *Let $\mathcal{A}$ be a Mealy automaton with n states. Then, $\mathcal{A}$ can be transformed into an equivalent minimal Mealy automaton $\mathcal{A}'$ in time $\mathcal{O}(n \cdot \log n)$, and $\mathcal{A}'$ is uniquely determined (up to isomorphism).*

We intend to use Mealy automata as strategy automata for infinite games on graphs. That means, the automaton reads the vertices visited in a play and changes its state accordingly, and the output function determines the moves of one player. If we are given a strategy automaton for Player 0 and the current vertex is owned by her, then the automaton additionally outputs the vertex to which she is supposed to move to next; if the current vertex belongs to Player 1, then τ outputs a dummy letter $\bullet \notin V$. (In a strategy automaton for Player 1, the dummy letter is output when Player 0 moves.) This extension of τ is necessary to guarantee correctness[1] of the minimization algorithm described above. Since in our case input letters are vertices in V and output letters are vertices

[1]We require length preservation in $\mathcal{A}$: for two words w, w' with $|w| < |w'|$ it must hold $\tau^*(q_0, w) < \tau^*(q_0, w')$, where τ^* is the extension of τ from input letters to input words, in the natural way.

in V or the dummy letter $\bullet$, we abolish the particular output alphabet Σ_O. Moreover, from now on, we use only the term "strategy automaton", i.e., we mean an automaton of the format $\mathcal{A} = (S, s_0, \sigma, \tau)$ over the input alphabet V (cf. page 16), and the output alphabet is $V \cup \{\bullet\}$.

In the following section, we explain two reasons why minimization of strategy automata is mostly unreasonable for memory reduction.

3.1.1 Disadvantages of Strategy Automata

We have already indicated how strategy automata can be minimized algorithmically. That means, if we are given an automaton $\mathcal{A}$ computing the output function f, then we can compute the minimal automaton implementing f. However, this does not hold for the strategy implemented by $\mathcal{A}$, say g. This is due to the fact that a strategy is a partial function, and it needs only be defined for the set of possible play prefixes on the underlying game graph. Hence, some transitions of $\mathcal{A}$ can be useless for the definition of g. If we are given a minimal automaton implementing f and redirect the irrelevant transitions, then the strategy implemented by the new automaton is still g, but it needs no longer be minimal.

Example 3.3. Consider the Staiger-Wagner game depicted in Figure 3.1 with the winning condition $\mathcal{F} = \{\{0,1\},\{1,2\}\}$ for Player 0. In this game Player 0's winning region is $W_0 = \{0,1\}$, and she has the following winning strategy of size two: if the play starts in vertex 0, then move to vertex 1 and stay there; if it starts in vertex 1, then move to vertex 2.

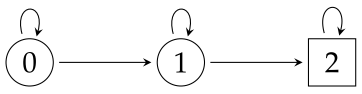

Figure 3.1: *Staiger-Wagner game*

Consider the automata $\mathcal{A}$ and $\mathcal{A}'$ in Figure 3.2, where input letters are written to the left of a vertical line and output letters are written to the right of that line. Both $\mathcal{A}$ and $\mathcal{A}'$ are minimal with respect to the output function they compute, and each of them implements the aforementioned strategy.

If $\mathcal{A}$ changes its state from s_0 to s_2, then Player 0 moves from vertex 1 to vertex 2 and, hence, letter 0 can never be read again. Thus, the transition from s_2 to s_2 reading input letter 0 is irrelevant, and we can redirect it from s_2 to s_1 without changing the implemented strategy. However, this yields a non-minimal automaton, say $\mathcal{A}^*$, where states s_0 and s_2 are equivalent with respect

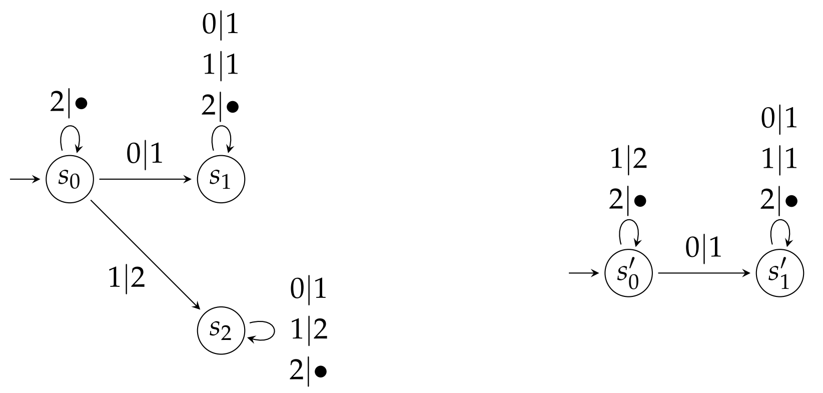

Figure 3.2: *Strategy automata $\mathcal{A}$ (on the left) and $\mathcal{A}'$ (on the right)*

to the computed output function. Minimizing $\mathcal{A}^*$ we obtain precisely $\mathcal{A}'$ (up to isomorphism).

More generally speaking, there may exist infinitely many automata, each of them implementing the same strategy, such that each automaton is minimal (with respect to the output function it computes). For example, assume we are given a game $\Gamma = (G, \varphi)$ and simulate it by a new game $\Gamma' = (G', \varphi')$ to construct a winning strategy. Then, we may consider only the reachable part of G' and thereby obtain a partially defined automaton containing only the relevant transitions. However, to be able to apply the minimization algorithm we need to specify all the missing transitions as well. Clearly, there is an exponential number of possible completions and we know of no efficient procedure to find the one which allows for the best minimization. If we allow to add states, then there are even infinitely many possibilities. Note that this yields arbitrary large minimal automata. For a further discussion on minimization of partially defined Mealy automata we refer to [Koh70].

The above considerations show that minimization of a given strategy is heavily influenced by the particular representation of the strategy. However, the major shortcoming we want to address here is needless complexity of the strategy itself. Even if we have a winning strategy automaton with the irrelevant transitions defined such that minimization yields an optimal (with respect to the implemented strategy) number of states, then still there can exist other winning strategies which require less memory.

From now on, let f denote a strategy rather than a whole output function. Consider the Staiger-Wagner game depicted in Figure 3.3 with the winning condition $\mathcal{F} = \{\{0,1\}, \{0,2\}, \{0,1,2,3\}\}$ for Player 0. In this game Player 0's winning region is $W_0 = \{0,1\}$, and she has the following winning strategy f of size two: if the play has visited vertex 1, then move from vertex 2 to vertex 3, otherwise stay at vertex 2. It can easily be shown that this strategy is optimal in the sense that Player 0 has no positional winning strategy.

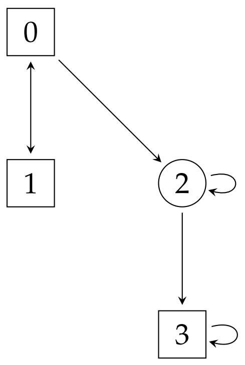

Figure 3.3: *Staiger-Wagner game*

Consider the automaton $\mathcal{A}_n$ depicted in Figure 3.4, where we omit the output letter $\bullet$. It has $n + 2$ states and implements a variant of the strategy f, say f_n: if vertex 1 has been visited, then stay at vertex 2 for exactly n times and afterwards move on to vertex 3; otherwise stay at vertex 2. (Note that it holds $f = f_0$.)

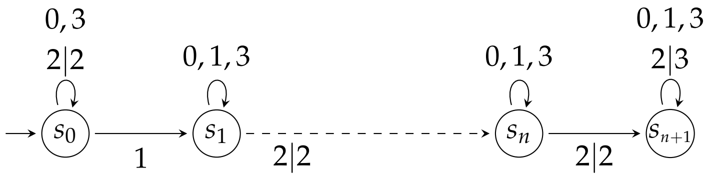

Figure 3.4: *Strategy automaton $\mathcal{A}_n$*

Clearly, for each $n \in \mathbb{N}$, the strategy f_n is a winning strategy for Player 0. Moreover, the automaton $\mathcal{A}_n$ is minimal with respect to f_n because we need the states $s_1, \ldots, s_n$ to keep track of the exact number of revisits to vertex 2. If n is arbitrary, then we get an arbitrary large strategy automaton; this is very unsatisfying.

Even though the algorithms from Section 2.3 guarantee upper bounds for the size of the used memory, we cannot assume that they yield a winning strategy of acceptable size, i.e., at most polynomial in the size of the optimal winning strategy. We come back to this problem in Section 3.6. There, we show that there exist games where standard algorithms yield winning strategies of size exponentially larger than the size of optimal ones.

Subsuming the previous observations shows that we should use an algorithm for memory reduction which is independent of any particular strategy. Rather, it should make use of properties of the game itself and use this to find a winning strategy which is as simple as possible. In the following section we present an algorithm which overcomes the disadvantages of the minimization algorithm for strategy automata.

3.2 Reduction of Game Graphs

In this section we present our algorithm which reduces the memory necessary for the implementation of winning strategies in infinite games. The basic idea is to apply a game simulation and then to simplify the expanded game graph, *before* the computation of winning strategies. The simplification is accomplished via state space reduction of an ω-automaton. We show the correctness of our approach and apply it to four classes of winning conditions, namely Staiger-Wagner, Request-Response, Muller and Streett[2] conditions. The main advantage of our algorithm over the technique of automata minimization (see Section 3.1) is that the actual memory reduction is independent of any particular strategy.

Let us explain our technique in more detail; it is illustrated in Figure 3.5. We are given an infinite game Γ which is to be solved. Our algorithm realizes a modification of the output of the game simulation, i.e., the dashed arrow from Γ' to Γ''. The infinite game Γ'' admits easier winning strategies than Γ'. This is achieved by the sequence of solid arrows from Γ' to Γ'', via the automata $\mathcal{A}$ and $\mathcal{B}$. The arrow from Γ to Γ' is not really part of our algorithm, but just a standard technique used for introducing memory. Analogously, computing a solution to the game Γ'' (or to Γ') is independent of the technique of game simulation. Therefore, we measure the complexity of our approach in the size of Γ', but not in the size of Γ, and we consider the time needed for solving Γ'', separately.

We reduce the size of the game graph G' and get a new game graph G''. For formal reasons, the actual reduction is carried out on an ω-automaton $\mathcal{A}$ which is equivalent to the considered game, in the sense that it holds $L(\mathcal{A}) = \varphi$. The state set and the acceptance condition of $\mathcal{A}$ are the set of vertices and the winning condition of Γ', respectively. As indicated in Figure 3.5, the properties of

[2]For Muller and Streett conditions our algorithm coincides, except for the game simulation in the first step.

game simulation shall be preserved when Γ'' is computed from Γ'. To guarantee this, we need to maintain the structural properties of G', proceeding as follows: we compute a language-preserving equivalence relation $\approx$ on the set $S \times V$. From $\approx$ we compute a refined relation $\approx_S$ on S and use this relation to reduce only the set S; the new memory is the set $S' := S/_{\approx_S}$. The correctness of our algorithm, i.e., the fact that $\Gamma \leq \Gamma''$ holds (see the leftmost dashed arrow in Figure 3.5), is solely derived from the *compatibility* of the relation $\approx$ (see Definition 3.6).

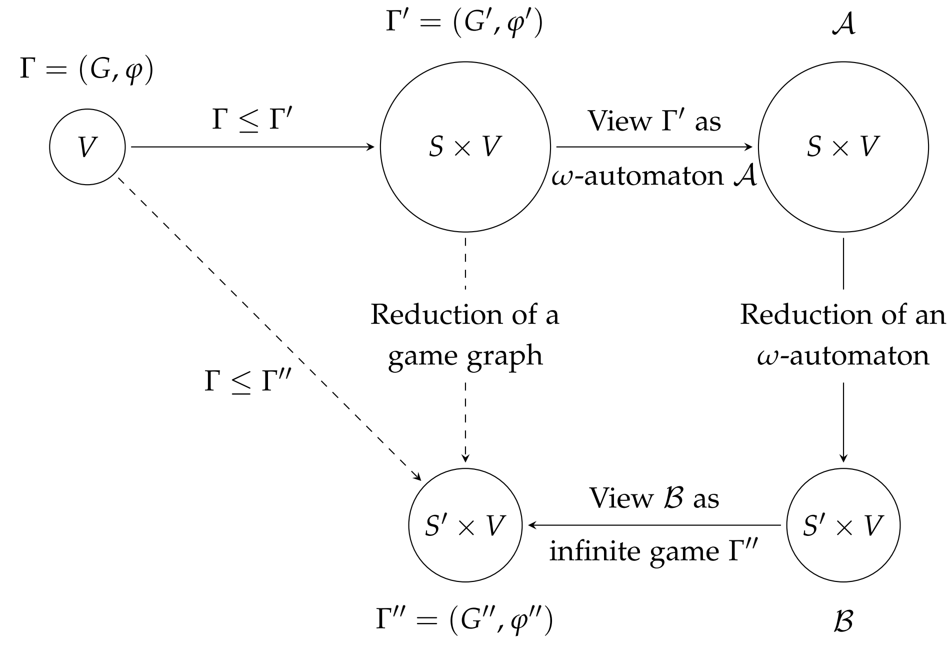

Figure 3.5: *Algorithm for memory reduction*

In the sequel of this section, we explain the single steps of our algorithm, i.e., the solid arrows in Figure 3.5, where the concept of game simulation has already been introduced in Section 2.2. Hence, let us directly proceed to a formal definition of the automaton $\mathcal{A}$.

Definition 3.4. Let $\Gamma = (G, \varphi), \Gamma' = (G', \varphi')$ be two infinite games with game graphs $G = (V, V_0, V_1, E), G' = (S \times V, S \times V_0, S \times V_1, E')$ such that $\Gamma \leq \Gamma'$. We define the (det.) *game automaton* $\mathcal{A}(\Gamma') := ((S \times V) \uplus \{q_0, q_{\text{sink}}\}, q_0, \delta, \psi, V_0)$ over V. The state set consists of the set of all vertices of G' plus two auxiliary states: q_0 is the initial state and q_{sink} is a sink state to intercept inputs which are not valid plays on G. The transition function δ is mainly adopted from E' such that a transition is labeled by the V-component of its target state: if $((s_1, v_1), (s_2, v_2)) \in E'$ then $\delta((s_1, v_1), v_2) := (s_2, v_2)$.[3] For $v' \in V$, we set $\delta(q_0, v') := (s_0, v')$, where s_0 is the initial memory content given by the game simulation, and $\delta(q_{\text{sink}}, v') := q_{\text{sink}}$. For $s \in S, v, v' \in V$ with $(v, v') \notin E$, we set $\delta((s, v), v') := q_{\text{sink}}$.

The acceptance condition ψ is defined on an abstract level: a run $q_0 \varrho'$ of $\mathcal{A}$ is defined to be accepting if ϱ' is a play winning for Player 0 in Γ'.[4]

For retranslation of the automaton $\mathcal{B}$ into the infinite game Γ'' (see Figure 3.5), we use a construction reverse to that of a game automaton from an infinite game. For retrieval of the partition of the vertices, we need to keep the information V_0 in the signature of $\mathcal{A}$. We call this an *automaton game*, denoted $\Gamma(\mathcal{A})$.

The reader may verify that the game automaton $\mathcal{A}$ is deterministic and that it is equivalent to both Γ and Γ', in the sense that plays winning for Player 0 map to accepting runs, and vice versa.

Remark 3.5. Let Γ, Γ' be infinite games such that $\Gamma \leq \Gamma'$ and $\mathcal{A} := \mathcal{A}(\Gamma')$. Then, the language recognized by $\mathcal{A}$ is the set of all plays winning for Player 0 in Γ, i.e., it holds $L(\mathcal{A}) = \varphi$.

3.2.1 Reduction of Game Automata

In this section we explain the technical details for the reduction of game graphs via game automata. There are two major concerns we have to cope with:

1. The minimization problem for ω-automata is known to be Pspace-hard, already for deterministic Büchi automata [GJ79]. To overcome this obstacle a variety of simulation relations, which guarantee language preservation and admit efficient computation, have been studied (see for example [HHK95, HKR97, EWS05]). The particular relations used in this work are presented in Sections 3.3 through 3.5.

2. We want to reduce only the set S of memory contents, but not the complete state space. Therefore, any equivalence relation on the set of states needs to be refined, appropriately.

As indicated above, we intend to compute a language-preserving equivalence relation $\approx$ on the state space $S \times V$ of the game automaton $\mathcal{A}$, which is refined to an equivalence relation $\approx_S$ on S, afterwards. Moreover, the quotient

[3]The transition labels are redundant information, but they are necessary to define the language accepted by $\mathcal{A}$.

[4]In particular, the run $q_0 \varrho'$ must mimic a play on G', i.e., ϱ' does not visit q_{sink}.

automaton with respect to $\approx_S$ shall have a deterministic transition structure. These requirements are subsumed in the following definition.

Definition 3.6. Let $\mathcal{A}$ be a game automaton and let $\approx$ be an equivalence relation on $S \times V$. We say that $\approx$ is *compatible* with $\mathcal{A}$ if the following hold:

1. For all $s_1, s_2 \in S, v, v' \in V$:
   $(s_1, v) \approx (s_2, v) \Longrightarrow \delta((s_1, v), v') \approx \delta((s_2, v), v')$
2. Let ρ and ρ' be two runs of $\mathcal{A}$ (starting in arbitrary states) such that $\rho(i) \approx \rho'(i)$ for all $i \geq 0$. Then ρ is accepting if and only if ρ' is accepting.

Our idea for the definition of the refinement $\approx_S$ is as follows: if it holds the equivalence $(s_1, v) \approx (s_2, v)$, then from these two states exactly the same inputs are accepted. This means, if in Γ' a player plays the same strategy from (s_1, v) and (s_2, v), then the plays built up are both either winning or losing, irrespective of what the opponent does. Accordingly, if the equivalence $(s_1, v) \approx (s_2, v)$ holds for all $v \in V$, then s_1 and s_2 need not be distinguished.

Definition 3.7. Let $\mathcal{A}$ be a game automaton and let $\approx$ be a compatible equivalence relation on $S \times V$. The equivalence relation $\approx_S$ on S is defined as follows:

$$s_1 \approx_S s_2 :\Longleftrightarrow \forall v \in V : (s_1, v) \approx (s_2, v)$$

The equivalence class of s with respect to $\approx_S$ is denoted $[s]$. Note that property 1 of Definition 3.6 does not translate from $\approx$ to $\approx_S$. That means, if we are given $s_1, s_2 \in S$ with $s_1 \approx_S s_2$, $(v, v') \in E$ and $(s'_i, v') := \delta((s_i, v), v')$ for $i = 1, 2$, then it holds $(s'_1, v') \approx (s'_2, v')$, but s'_1, s'_2 need not be $\approx_S$-equivalent. Hence, to get the v'-successor of $([s_1], v)$ well-defined, we choose a fixed total order $\prec_S$ on S, i.e., $s_1 \prec_S s_2 \prec_S s_3 \prec_S \ldots$, and use the $\prec_S$-minimal element among all successor memory contents. This is clarified by the following definition.

Definition 3.8. Let $\mathcal{A}$ be a game automaton, $\approx$ a compatible equivalence relation on $S \times V$ and $\approx_S$ derived from $\approx$ as above. We define the deterministic *quotient game automaton* $\mathcal{A}/_{\approx_S} = ((S \times V) \cup \{q_0, q_{\text{sink}}\}, q_0, \delta/_{\approx_S}, \psi/_{\approx_S}, V_0)$ over V. Given $([s], v) \in S/_{\approx_S} \times V$ and $(v, v') \in E$, we define

$$\delta/_{\approx_S}(([s], v), v') := ([s_{\min}(s, v)], v'),$$

where

$$s_{\min}(s, v) := \min\{\hat{s}' \mid \exists \hat{s} \exists v' : \hat{s} \approx_S s \text{ such that } \delta((\hat{s}, v), v') = (\hat{s}', v')\}.$$

Note that $\hat{s}'$ depends only on $\hat{s}$ and v, due to uniqueness of the memory update; hence $s_{\min}(s, v)$ is well-defined. The rest of $\delta/_{\approx_S}$ is defined analogously:

for each $v \in V$, we have a transition from q_0 to $([s_0], v)$, where s_0 is the initial memory content; for $s_1 \approx_S s_2, (v, v') \notin E$, the transitions $\delta((s_1, v), v') = \delta((s_2, v), v') = q_{\text{sink}}$ in $\mathcal{A}$ are merged such that $\delta/_{\approx_S}(([s_1], v), v') := q_{\text{sink}}$. The transitions from q_{sink} to q_{sink} are adopted from $\mathcal{A}$.

The acceptance condition $\psi/_{\approx_S}$ of $\mathcal{A}/_{\approx_S}$ is defined in terms of accepting runs: let $\rho = q_0([s_0], v_0)([s_1], v_1)([s_2], v_2)\cdots$ be the run of $\mathcal{A}/_{\approx_S}$ on $v_0v_1v_2\cdots$ and let $\rho' = q_0(s'_0, v_0)(s'_1, v_1)(s'_2, v_2)\cdots$ be the corresponding run of $\mathcal{A}$, i.e., it holds $s'_i \in [s_i]$ for all $i \in \mathbb{N}$. We define ρ to be accepting if and only if ρ' is accepting.

Note that the run ρ' is uniquely determined by the run ρ which, in turn, is uniquely given by the word $v_0v_1\cdots$, because both $\mathcal{A}$ and $\mathcal{A}/_{\approx_S}$ are deterministic. Then, the acceptance condition of $\mathcal{A}/_{\approx_S}$ immediately implies $L(\mathcal{A}) = L(\mathcal{A}/_{\approx_S})$, because accepting and non-accepting runs in $\mathcal{A}$ correspond to accepting and non-accepting runs in $\mathcal{A}/_{\approx_S}$, respectively.

Remark 3.9. Let $\mathcal{A}$ be a game automaton, $\approx$ a compatible equivalence relation on $S \times V$ and $\approx_S$ as given by Definition 3.7. Then, $\mathcal{A}$ and $\mathcal{A}/_{\approx_S}$ are equivalent, i.e., it holds $L(\mathcal{A}) = L(\mathcal{A}/_{\approx_S})$.

In the forthcoming sections we present game automata with either weak parity, Büchi or parity acceptance condition. For each of the three we prove existence of a compatible equivalence relation and use this to apply our memory reduction algorithm. Before that, we close the present section by giving a formal correctness proof of our technique. In the following theorem we show that the automaton game Γ'' of $\mathcal{A}/_{\approx_S}$ has the same structural properties as Γ', i.e., the game simulation relation is preserved. This means that it holds $\Gamma \leq \Gamma''$, as indicated by the leftmost dashed arrow in Figure 3.5.

Theorem 3.10. *Let $\Gamma = (G, \varphi)$ and $\Gamma' = (G', \varphi')$ be infinite games such that Γ is simulated by Γ'. Let $\mathcal{A}$ be the game automaton of Γ' and $\approx$ a compatible equivalence relation on $S \times V$. Then, Γ is simulated by the unique automaton game Γ'' of $\mathcal{A}/_{\approx_S}$.*

Proof. From Definitions 3.6 and 3.8 it follows that $\mathcal{A}/_{\approx_S}$ has a state set of the form $(S' \times V) \uplus \{q_0, q_{\text{sink}}\}$ for a finite set S', and a deterministic transition structure. Hence, $\mathcal{A}/_{\approx_S}$ is a game automaton and we can transform it into the unique automaton game $\Gamma'' = (G'', \varphi'')$ with $G'' = (V'', E'')$, where the owner of a vertex $(s', v) \in V''$ is given by the owner of $v \in V$. It remains to show that Γ is simulated by Γ''. To do so, we verify items 1 through 3 of Definition 2.6; item 1 is already clear by the previous remarks.

For item 2a, we choose $[s_0]$ as initial memory content and obtain as possible initial vertices on G'' the set $\{([s_0], v) \mid v \in V\}$. To check the edge relation E'' of G'', consider a vertex $([s], v) \in V''$; obviously, it holds $(s, v) \in S \times V$.

Let $v' \in V$ be such that $(v, v') \in E$. Since $\Gamma \leq \Gamma'$, there exists $s' \in S$ such that $((s, v), (s', v')) \in E'$ and, hence, there exists (a unique) $s_{\min}(s, v) \in S$ such that $\delta/_{\approx_S}(([s], v), v') = ([s_{\min}(s, v)], v')$ (cf. Definition 3.8). This yields the edge $(([s], v), ([s_{\min}(s, v)], v')) \in E''$, validating item 2(b)i. For the reverse direction, i.e., item 2c, we argue the other way round. However, note that $(([s], v), ([s_{\min}(s, v)], v')) \in E''$ does not imply $((s, v), (s_{\min}(s, v), v')) \in E'$. But, by Definition 3.8, there must exist $\hat{s} \in [s]$ such that $((\hat{s}, v), (s_{\min}(s, v), v')) \in E'$. Thereby, the existence of (v, v') in E is verified.

To complete the proof of item 2 from Definition 2.6, we show uniqueness of the memory update as follows: for $s, s_1, s_2 \in S$ and $v, v_1, v_2 \in V$, we assume the edges $(([s], v), ([s_1], v_1)), (([s], v), ([s_2], v_2)) \in E''$. By Definition 3.4, this means $\delta/_{\approx_S}(([s], v), v_1) = ([s_1], v_1)$ and $\delta/_{\approx_S}(([s], v), v_2) = ([s_2], v_2)$. Towards a contradiction, assume $[s_1] \neq [s_2]$ and consider the following implication:

$$\begin{aligned}
[s_1] \neq [s_2] \overset{\text{Def. 3.8}}{\Longrightarrow} & [\min\{\hat{s}' \mid \exists \hat{s} : \hat{s} \approx_S s \text{ such that } \delta((\hat{s}, v), v_1) = (\hat{s}', v_1)\}] \neq \\
& [\min\{\hat{s}' \mid \exists \hat{s} : \hat{s} \approx_S s \text{ such that } \delta((\hat{s}, v), v_2) = (\hat{s}', v_2)\}] \\
\Longrightarrow\ & \min\{\hat{s}' \mid \exists \hat{s} : \hat{s} \approx_S s \text{ such that } \delta((\hat{s}, v), v_1) = (\hat{s}', v_1)\} \neq \\
& \min\{\hat{s}' \mid \exists \hat{s} : \hat{s} \approx_S s \text{ such that } \delta((\hat{s}, v), v_2) = (\hat{s}', v_2)\} \\
\Longrightarrow\ & \{\hat{s}' \mid \exists \hat{s} : \hat{s} \approx_S s \text{ such that } \delta((\hat{s}, v), v_1) = (\hat{s}', v_1)\} \neq \\
& \{\hat{s}' \mid \exists \hat{s} : \hat{s} \approx_S s \text{ such that } \delta((\hat{s}, v), v_2) = (\hat{s}', v_2)\} \\
\overset{\text{Def. 3.4}}{\Longrightarrow}\ & \{\hat{s}' \mid \exists \hat{s} : \hat{s} \approx_S s \text{ such that } ((\hat{s}, v), (\hat{s}', v_1)) \in E'\} \neq \\
& \{\hat{s}' \mid \exists \hat{s} : \hat{s} \approx_S s \text{ such that } ((\hat{s}, v), (\hat{s}', v_2)) \in E'\} \\
\overset{\text{Def. 2.6}}{\Longrightarrow}\ & \lightning
\end{aligned}$$

The last inequality yields the desired contradiction as follows: for every single $\hat{s} \in S$ with $\hat{s} \approx_S s$, the memory update $\hat{s}'$ from $(\hat{s}, v)$ is unique. If we iterate this argument, then we get that the above sets of memory updates must coincide. Thus, we have deduced the uniqueness of the memory update in G'' from the uniqueness of the memory update in G'.

What remains to be shown is that plays winning for Player 0 in Γ map to plays winning for her in Γ'', and vice versa (cf. item 3 of Definition 2.6). This basically follows from the fact that $\mathcal{A}$ and $\mathcal{A}/_{\approx_S}$ are equivalent. Let $\varrho = v_0 v_1 v_2 \cdots$ be a play in Γ, and let $\varrho' = (s_0, v_0)(s_1, v_1)(s_2, v_2) \cdots$ and $\varrho'' = ([s_0'], v_0)([s_1'], v_1)([s_2'], v_2) \cdots$ be the corresponding plays in Γ' and Γ'', respectively. Then, we get the following equivalence:

$$
\begin{aligned}
\varrho \in \varphi &\overset{\Gamma \leq \Gamma'}{\iff} \varrho' \in \varphi' \\
&\overset{\text{Def. 3.4}}{\iff} \text{the unique run } q_0\varrho' \text{ of } \mathcal{A} \text{ on } \varrho \text{ is accepting} \\
&\iff \varrho \in L(\mathcal{A}) \\
&\overset{\text{Rem. 3.9}}{\iff} \varrho \in L(\mathcal{A}/_{\approx_S}) \\
&\iff \text{the unique run } q_0\varrho'' \text{ of } \mathcal{A}/_{\approx_S} \text{ on } \varrho \text{ is accepting} \\
&\overset{\text{Def. 3.4}}{\iff} \varrho'' \in \varphi''
\end{aligned}
$$

□

The essence of Theorem 3.10 is that from a solution to Γ'' we can construct winning strategies for both players in Γ which are implemented with less memory than winning strategies constructed by solving Γ'. More precisely, we have reduced the upper bound for the size of the used memory from $|S|$ to $|S/_{\approx_S}|$.

Let us formulate the full memory reduction algorithm schematically depicted in Figure 3.5 (cf. page 48).

Algorithm 3.1 (Memory Reduction)

Input: Infinite game $\Gamma = (G, \varphi)$
Output: Strategy automaton $\mathcal{A}_f$ for Player 0 from W_0

```
1: Establish a game simulation of $\Gamma$ by a new game $\Gamma'$ in which Player 0 has a
   positional winning strategy.
2: View $\Gamma'$ as $\omega$-automaton $\mathcal{A}$.
3: Reduce $\mathcal{A}$: use a compatible equivalence relation $\approx$ on $S \times V$ to com-
   pute $\approx_S$ on $S$ and construct the corresponding quotient game automa-
   ton $\mathcal{A}/_{\approx_S}$.
4: Transform $\mathcal{A}/_{\approx_S}$ into the unique automaton game $\Gamma''$.
5: Compute a positional winning strategy for Player 0 in $\Gamma''$ and from it con-
   struct the strategy automaton $\mathcal{A}_f$.
```

Note that Algorithm 3.1 does not depend on the actual winning condition φ; we only need a compatible equivalence relation $\approx$ for the execution of step 3. Moreover, our technique works correctly even if Γ' or Γ'' do not admit positional winning strategies. In this case we need to apply a product construction, which gets as input the memory update structure of the reduced game, i.e., the graph G'', and a strategy automaton implementing a winning strategy for Player 0 in Γ''.

Note that, once the strategy automaton $\mathcal{A}_f$ is computed, the classical minimization approach for strategy automata can be applied; this observation holds independently of the procedure the automaton $\mathcal{A}_f$ is obtained from.

In the upcoming sections we present techniques to reduce the size of game automata with particular acceptance conditions. We show that the equivalence relations used are compatible and that it is possible to define a quotient game automaton (cf. Definition 3.8) with the same type of acceptance condition.

3.3 Staiger-Wagner Conditions

In this section we apply Algorithm 3.1 to Staiger-Wagner games. For the game simulation in step 1 we use the algorithm presented on page 28, obtaining a weak parity game automaton. To reduce its state space we transform it into a DWA, i.e., a deterministic *weak* Büchi automaton. DWA are a special kind of deterministic Büchi automata and can be efficiently minimized with more or less the same algorithm as for minimizing automata over finite words [Löd01]. More precisely, we do some precomputations to the DWA and afterwards compute the state equivalence known from minimization of standard DFA. We show that the obtained relation is compatible with the DWA and from it compute the equivalence relation $\approx_S$ according to Definition 3.7. The quotient automaton induced by $\approx_S$ is again a DWA, and it is transformed back into an equivalent weak parity game automaton. This automaton is then used to execute the last two steps of Algorithm 3.1.

In Section 3.6.1, we present a family of Staiger-Wagner games where our algorithm yields an exponential reduction of the used memory, as the classical approach, i.e., minimization of a strategy automaton, yields exponential memory. As a start, let us introduce deterministic weak Büchi automata.

3.3.1 Deterministic Weak Büchi Automata

DWA form a special subclass of deterministic Büchi automata (DBA). They recognize the class of regular languages which are both recognizable by deterministic Büchi and deterministic co-Büchi automata. This means that they are weaker than nondeterministic Büchi automata (NBA) and also weaker than DBA. (We introduce DWA here, because we only use them to reduce weak parity game automata.)

An important property of a weak parity game automaton is that in every run the sequence of seen colors is weakly increasing (cf. page 29). This immediately implies that all states within the same strongly connected component have the same color. A *strongly connected component* (short: SCC) is a maximal subset $C \subseteq Q$ such that each state in C is reachable from every other state in C. If the color of some SCC C is even, then a run remaining in C until infinity is accepting, no matter which particular states are visited. This is a certain kind of weakness which similarly can be found in DWA.

Definition 3.11. Let $\mathcal{A}{=}(Q, q_0, \delta, F)$ be a deterministic Büchi automaton over Σ. The DBA $\mathcal{A}$ is called a deterministic *weak* Büchi automaton if every SCC of $\mathcal{A}$ contains either only final states or only non-final states. According to this we call a SCC final or non-final.

We call a state $q \in Q$ *recurrent* if there is a word $w \in \Sigma^+$ such that $\delta^*(q, w) = q$. Otherwise q is called *transient*. A SCC is called transient if it has only one state and this state is transient. Otherwise a SCC is called recurrent.

The first step towards a minimization algorithm for DWA is to express the acceptance condition by means of a coloring.

Definition 3.12. Let $\mathcal{A} = (Q, q_0, \delta, F)$ be a DWA and $k \in \mathbb{N}$. A function $c : Q \to \{0, \dots, k\}$ is called $\mathcal{A}$*-coloring* if the following hold:

1. $c(q)$ is even for every recurrent state $q \in F$
2. $c(q)$ is odd for every recurrent state $q \notin F$
3. $c(q) \leq c(r)$ for all $q, r \in Q$ with $\delta(q, a) = r$ for some $a \in \Sigma$

A coloring c is called *k-maximal* if the following hold:

1. $c'(q) \leq c(q)$ for every $\mathcal{A}$-coloring $c' : Q \to \{0, \dots, k\}$ and all $q \in Q$
2. $c(q) \leq k$ for every $q \in Q$

A coloring is called *maximal* if it is k-maximal for some $k \in \mathbb{N}$.

The following remark (deduced from [Löd01]) shows that an $\mathcal{A}$-coloring can be used to express the acceptance condition of $\mathcal{A}$ in terms of a weak parity condition. However, note that the notion of coloring from Definition 3.12 is more restrictive than the one introduced on page 14.

Remark 3.13. Let $\mathcal{A}$ be a deterministic weak Büchi game automaton over V and let c be an $\mathcal{A}$-coloring. Let further $\mathcal{A}'$ be the weak parity game automaton determined by $\mathcal{A}$ and acceptance condition c. Then, $\mathcal{A}$ and $\mathcal{A}'$ are equivalent.

Vice versa, given a weak parity game automaton we can construct an equivalent DWA, declaring final all states with even color. Altogether, we obtain that weak parity game automata and deterministic weak Büchi game automata can be transformed into each other. Note that both directions take only linear time.

Let us consider a technique for minimization of (general) DWA [Löd01]. Basically, the algorithm works as for standard DFA. The only difference is that, before computing equivalent states, the DWA has to be normalized (see Definition 3.14). As is shown below, the normalization can be done in time $\mathcal{O}(n)$, where n is the number of states of the given DWA. Applying for example block partitioning [Hop71], one finally obtains a minimal equivalent DWA uniquely determined (up to isomorphism) in time $\mathcal{O}(n \cdot \log n)$. We do not want to give the complete correctness proof for this approach, but we mention one fundamental part of it and refer to [Löd01] for the remaining details. That is, we explain how a DWA is normalized. For that we need the following definition.

Definition 3.14. Let $\mathcal{A} = (Q, q_0, \delta, F)$ be a DWA over Σ and c an $\mathcal{A}$-coloring. Let F_c be the set of states with even color: $F_c := \{q \mid c(q) \text{ is even}\}$. We say that $\mathcal{A}$ is in *normal form* if $F = F_c$ for some k-maximal $\mathcal{A}$-coloring $c : Q \to \{0, \ldots, k\}$ with k even.

Theorem 3.15 ([Löd01])**.** *Let $\mathcal{A} = (Q, q_0, \delta, F)$ be a DWA over Σ with n states. Then there exists a set $F' \subseteq Q$ of final states computable in time $\mathcal{O}(n)$ such that the DWA $\mathcal{A}' = (Q, q_0, \delta, F')$ over Σ is in normal form and equivalent to $\mathcal{A}$.*

Proof. The asymptotic running time of each graph search algorithm used in this proof is linear in $n + |\Sigma|n$, i.e., the size of $\mathcal{A}$. Since we consider $|\Sigma|$ a constant we obtain an overall running time in $\mathcal{O}(n)$ (see [Löd01, Sed91] for further details).

The main part of the proof is to find a k-maximal $\mathcal{A}$-coloring, for some even $k \in \mathbb{N}$. From item 3 of Definition 3.12 we immediately deduce that all states within the same SCC get the same color. Hence, we only have to find a maximal coloring for the SCC-graph G of $\mathcal{A}$. This graph is of the form $G = (V, E)$ with $V = \{Q_1, \ldots, Q_m\}$, where $Q_1, \ldots, Q_m$ are the SCCs of $\mathcal{A}$. There is an edge $(Q_i, Q_j) \in E$ if and only if $i \neq j$ and there are states $q_i \in Q_i, q_j \in Q_j$ such that $\delta(q_i, a) = q_j$ for some $a \in \Sigma$. Since G is acyclic the set V is partially ordered by E and, hence, we can assume that V is topologically sorted, i.e., $(Q_i, Q_j) \in E$ implies $i < j$. Moreover, we can assume that all SCCs are marked whether they are transient (or recurrent) and whether they are final (or non-final). If $\mathcal{A}$ has m SCCs, then we compute a m'-maximal coloring d for G, where m' is the smallest even number greater than or equal to m, i.e., $m' := m$ if m is even or $m' := m{+}1$ if m is odd. Algorithm 3.2 computes the coloring d as follows: we start with all SCCs that have no successors in G, assigning to them the highest possible color (either m' or $m'{-}1$). From that we inductively obtain the d-values for predecessor-SCCs.

Algorithm 3.2 (Maximal Coloring)

Input: Topologically sorted SCC-graph $G = (V, E)$ with $V = \{Q_1, \ldots, Q_m\}$
Output: Maximal coloring for G

```
 1: for $i = m$ downto 1 do
 2:     if $succ(Q_i) = \varnothing$ then {Initialize $d(Q_i)$ with maximal possible value}
 3:         if $Q_i$ is final then
 4:             $d(Q_i) := m'$
 5:         else
 6:             $d(Q_i) := m'-1$
 7:         end if
 8:     else {Compute $d(Q_i)$ with help of successor values}
 9:         $l := \min\{d(Q_j) \mid Q_j \in succ(Q_i)\}$
10:         if $Q_i$ is transient then
11:             $d(Q_i) := l$
12:         else if $l$ is even and $Q_i$ is final then
13:             $d(Q_i) := l$
14:         else if $l$ is odd and $Q_i$ is non-final then
15:             $d(Q_i) := l$
16:         else
17:             $d(Q_i) := l-1$
18:         end if
19:     end if
20: end for
21: return $d$
```

Since V is topologically sorted, the d-values of all successors of a SCC Q_i are computed before $d(Q_i)$ is computed. The values of the SCCs with no successors are maximal (either m' or $m'-1$), and so are the d-values of all other SCCs. Note that each transition of $\mathcal{A}$ is considered at most once, namely in line 9, which takes time at most $|\Sigma|n$. Let F' be the set of states with even d-color:

$$F' := \bigcup_{d(Q_i) \text{ is even}} Q_i$$

For $\alpha \in \Sigma^\omega$, the run of $\mathcal{A}$ on α and the run of $\mathcal{A}'$ on α coincide. Moreover, for each recurrent SCC Q_i, it holds that Q_i is final in $\mathcal{A}$ if and only if Q_i is final in $\mathcal{A}'$. There may be SCCs which are final in $\mathcal{A}$ and non-final in $\mathcal{A}'$, or vice versa, but this can only be the case for transient SCCs, which have no effect on acceptance. Hence, $\mathcal{A}$ and $\mathcal{A}'$ are equivalent. ☐

In [Löd01] it is shown that a normalized DWA $\mathcal{A}$ can be minimized in the same way as the DFA $\mathcal{A}$. To do so, we compute the state equivalence relation $\approx_{\mathcal{A}}$ (see for example [Hop71]); for two states q, q' of a DFA $\mathcal{A}$ we define

$$q \approx_{\mathcal{A}} q' :\iff \forall w \in \Sigma^* : \delta^*(q, w) \in F \iff \delta^*(q', w) \in F.$$

If we merge all $\approx_{\mathcal{A}}$-equivalent states in a normalized DWA, then we obtain an equivalent minimal DWA, which only depends on the accepted ω-language.

Theorem 3.16 ([Löd01])**.** *Let $\mathcal{A} = (Q, q_0, \delta, F)$ be a DWA over Σ with n states. Then, one can compute an equivalent minimal DWA $\mathcal{A}^{\min}$ in time $\mathcal{O}(n \cdot \log n)$, where $\mathcal{A}^{\min}$ is uniquely determined (up to isomorphism).*

3.3.2 Reduction of Weak Parity Game Automata

By the results of the previous section, a weak parity game automaton can easily be transformed into a deterministic weak Büchi game automaton. In the DWA, a state is declared final if it has an even color in the weak parity game automaton. Moreover, the initial state q_0 is declared final and the sink state q_{sink} is declared non-final. This does not change the accepted language and the automaton is still a DWA.

We show that the state equivalence relation $\approx_{\mathcal{A}}$ known from minimization of standard DFA is compatible with the DWA $\mathcal{A}$. (For a definition of $\approx_{\mathcal{A}}$ see at the end of Section 3.3.1.) Moreover, we define the quotient automaton $\mathcal{A}/_{\approx_S}$, where $\approx_S$ is computed from $\approx_{\mathcal{A}}$ according to Definition 3.7.

Lemma 3.17. *Let $\mathcal{A}$ be a deterministic weak Büchi game automaton. Then, the relation $\approx_{\mathcal{A}}$ is compatible with $\mathcal{A}$.*

Proof. We show item 1 of Definition 3.6 by contraposition. Consider two states $(s, v), (s', v') \in S \times V$ and a letter $v_0 \in V$ such that $\delta((s, v), v_0) \not\approx_{\mathcal{A}} \delta((s', v'), v_0)$. Then, there exists $w \in V^*$ such that

$$\delta^*(\delta((s, v), v_0), w) \in F \iff \delta^*(\delta((s', v'), v_0), w) \notin F.$$

Hence, the word $v_0 w$ separates (s, v) and (s', v'). For item 2, let ρ, ρ' be two runs such that $\rho(i) \approx_{\mathcal{A}} \rho'(i)$, for all $i \in \mathbb{N}$. Note that two $\approx_{\mathcal{A}}$-equivalent states are either both final or both non-final. From this it follows that ρ is accepting if and only if ρ' is accepting. □

We define equivalent memory contents as follows: for $s_1, s_2 \in S$, let $s_1 \approx_S s_2$ if for all $v \in V$ it holds $(s_1, v) \approx_{\mathcal{A}} (s_2, v)$. We compute the quotient automaton induced by $\approx_S$ according to Definition 3.8. It has the following set $F/_{\approx_S}$ of final states: for $[s] \in S/_{\approx_S}, v \in V$, let

$$([s], v) \in F/_{\approx_S} :\iff (s, v) \in F.$$

If $s_1 \approx_S s_2$ then by definition of $\approx_S$ it holds $(s_1, v) \approx_{\mathcal{A}} (s_2, v)$, for all $v \in V$. This implies the equivalence $(s_1, v) \in F \iff (s_2, v) \in F$, which means that $F/_{\approx_S}$ is well-defined. From the above results we deduce that a run ρ of $\mathcal{A}$ is accepting if and only if the corresponding run ρ' of $\mathcal{A}/_{\approx_S}$ is accepting. This means that Definition 3.8 is satisfied and $\mathcal{A}/_{\approx_S}$ is a deterministic weak Büchi game automaton equivalent to $\mathcal{A}$ (cf. Remark 3.9).

The automaton $\mathcal{A}/_{\approx_S}$ can also be considered as a weak parity game automaton, because only states with the same color can be merged. To see this, we go back to general DWA.

Lemma 3.18. *Let $\mathcal{A} = (Q, q_0, \delta, F)$ be a DWA over Σ and let c be a maximal $\mathcal{A}$-coloring. Let further $p, q \in Q$ such that $c(p) \neq c(q)$. Then, it holds $p \not\approx_{\mathcal{A}} q$.*

Proof. Let $p, q \in Q$ such that $c(p) \neq c(q)$. We use a lemma from [Löd01] to deduce that $L_\omega(\mathcal{A}_p) \neq L_\omega(\mathcal{A}_q)$. Hence, there exists $\alpha \in \Sigma^\omega$ such that w.l.o.g. it holds $\alpha \in L_\omega(\mathcal{A}_p)$ and $\alpha \notin L_\omega(\mathcal{A}_q)$. Since $\mathcal{A}_q$ assumes only finitely many final states in its run on α, there exists a finite prefix w of α such that $\delta^*(p, w) \in F$ and $\delta^*(q, w) \notin F$. Thus, w separates p and q. □

The above lemma shows that two states which are to be merged must have assigned the same color. This means that for each maximal coloring d (computed by Algorithm 3.2) the coloring $d/_{\approx_S}$ given by

$$d/_{\approx_S}([s], v) := d(s, v)$$

is well-defined. Hence, the DWA $\mathcal{A}/_{\approx_S}$ can be seen as a weak parity game automaton with coloring $d/_{\approx_S}$.

Let us summarize the results of Sections 3.3.1 and 3.3.2 by reformulating Algorithm 3.1 for Staiger-Wagner games.

3.3.3 A Memory Reduction Algorithm for Staiger-Wagner Games

Given a Staiger-Wagner game $\Gamma = (G, \mathcal{F})$, we apply the game simulation from page 28, obtaining a weak parity game $\Gamma' = (G', c)$. We consider Γ' as a weak parity game automaton which subsequently is transformed into an equivalent DWA. We apply Algorithm 3.2, computing a maximal coloring d, and obtain the normalized deterministic weak Büchi game automaton $\mathcal{A}$ with the set F of final states.

To $\mathcal{A}$ we apply a minimization algorithm for standard DFA, computing the equivalence relation $\approx_{\mathcal{A}}$. From $\approx_{\mathcal{A}}$ we compute $\approx_S$ according to Definition 3.7. By our previous results, the relation $\approx_{\mathcal{A}}$ is compatible with $\mathcal{A}$ and, accordingly, $\mathcal{A}/_{\approx_S}$ is a deterministic weak Büchi game automaton equivalent to $\mathcal{A}$. Using Remark 3.13, we view $\mathcal{A}/_{\approx_S}$ as weak parity game automaton with coloring $d/_{\approx_S}$. We get that Γ is simulated by the automaton game Γ'' of $\mathcal{A}/_{\approx_S}$.

Theorem 3.19. *Let Γ be a Staiger-Wagner game and Γ' the expanded weak parity game according to page 28. Further, let $\mathcal{A}$ be a normalized deterministic weak Büchi game automaton for Γ' with maximal coloring d, and let $\approx_{\mathcal{A}}$ be defined as on page 58. Then Γ is simulated by the weak parity automaton game Γ'' of $\mathcal{A}/_{\approx_S}$ with coloring $d/_{\approx_S}$.*

Algorithm 3.3 (Memory Reduction for Staiger-Wagner games)

```
Input: Staiger-Wagner game $\Gamma = (G, \mathcal{F})$
Output: Strategy automaton $\mathcal{A}_f$ for Player 0 from $W_0$
1: Establish a game simulation of $\Gamma$ by a weak parity game $\Gamma'$.
2: Transform $\Gamma'$ into a deterministic weak Büchi game automaton, via the
   weak parity game automaton of $\Gamma'$.
3: Apply Algorithm 3.2 to obtain a normalized DWA $\mathcal{A}$, say with maximal
   coloring $d$.
4: Consider $\mathcal{A}$ as a DFA and compute the state equivalence relation $\approx_{\mathcal{A}}$; re-
   fine $\approx_{\mathcal{A}}$ to obtain $\approx_S$ and construct the quotient DWA $\mathcal{A}/_{\approx_S}$.
5: Transform $\mathcal{A}/_{\approx_S}$ into an equivalent weak parity game automaton $\mathcal{B}$ with
   the coloring $d/_{\approx_S}$.
6: View $\mathcal{B}$ as weak parity automaton game $\Gamma''$.
7: Compute a positional winning strategy for Player 0 in $\Gamma''$ and from it con-
   struct $\mathcal{A}_f$.
```

We measure the running time of our algorithm in the size of Γ'. By the results of Section 3.3.1, transformation of a weak parity game automaton into a deterministic weak Büchi game automaton (or vice versa) takes time $\mathcal{O}(n)$, where n is the number of states of the given game automaton. Normalization of the deterministic weak Büchi game automaton $\mathcal{A}$ takes time linear in the number of transitions, which is $\mathcal{O}(n \cdot \log n)$; the factor $\log n$ comes from the input alphabet V, which is of size logarithmic in n. For the same reason, the computation of $\approx_{\mathcal{A}}$ takes time $\mathcal{O}(n \cdot (\log n)^2)$. The weak parity game Γ'' can be solved in time $\mathcal{O}(n \cdot \log n)$. Thus, the overall running time of Algorithm 3.3 is $\mathcal{O}(n \cdot (\log n)^2)$, i.e., polynomial in $|\Gamma'|$.

3.4 Request-Response Conditions

In this section, we apply our approach for memory reduction to Request-Response games. The first step is a game simulation by a Büchi game (cf. Theorem 2.12). Afterwards, we reduce the state space of the obtained Büchi game automaton. Several types of simulation relations have been proposed for automata with Büchi acceptance condition, for example *direct* and *fair simulation* (see for example [EH00, HKR97]). The problem with these notions is that they are inadequate for state space reduction. Direct simulation is too strong because in an accepting run of a Büchi automaton the particular positions where final states occur are not relevant; quotienting with respect to fair simulation does not preserve the recognized language.

To overcome this, we use the notion of *delayed simulation* (introduced in [EWS05]), which is an intermediate between direct and fair simulation, and it preserves the recognized language. Delayed simulation is defined by means of a simulation game between two players, called Spoiler and Duplicator. Duplicator wins if, for each visit to a final state in the run built up by Spoiler, there is a visit to a final state in the run built up by Duplicator, sometime later.

In [EWS05] it is shown that, for a (nondeterministic) Büchi automaton, the delayed simulation relation can be computed in polynomial time (by solving a parity game with constantly many colors). In our setting we deal with deterministic Büchi automata, which makes the situation much simpler. More precisely, we show that for Büchi game automata the computation of delayed simulation can be reduced to minimization of standard DFA, after some preprocessing.

3.4.1 Delayed Simulation for Büchi Automata

First of all, let us introduce the delayed simulation game for Büchi automata.

We assume that $\mathcal{A} = (Q, q_\mathrm{I}, \Delta, F)$ is a (nondeterministic) Büchi automaton over Σ with no dead ends, i.e., for each state q there exists a state q' and a letter a such that $(q, a, q') \in \Delta$. For $q_0, q_0' \in Q$, we define the *delayed simulation game* $\mathcal{G}_{de}(q_0, q_0')$. (We assume $\mathcal{A}$ implicitly and leave it out in our notation.) It is played by two players, called *Spoiler* and *Duplicator*. The players move in turn, Spoiler starting from q_0 and Duplicator starting from q_0'. In each round $i = 0, 1, 2, \ldots$, Spoiler chooses a transition $(q_i, a_i, q_{i+1}) \in \Delta$ and moves a pebble from q_i to q_{i+1} and, afterwards, Duplicator does the same with a transition $(q_i', a_i', q_{i+1}') \in \Delta$ with the same labeling ($a_i = a_i'$). Thereby, Spoiler and Duplicator build up the runs $\rho = q_0 q_1 q_2 \cdots$ and $\rho' = q_0' q_1' q_2' \cdots$, respectively.

Duplicator wins if and only if she can mimic each of Spoiler's visits to final states, in a delayed fashion: the play (ρ, ρ') is winning for Duplicator if it holds

$$\forall i (q_i \in F \Longrightarrow \exists j \geq i : q'_j \in F).$$

Spoiler wins if either the above condition is not satisfied or the play comes to a halt, i.e., Duplicator cannot answer by a transition labeled with the same letter previously chosen by Spoiler. We say that q'_0 *delayed simulates* q_0 if Duplicator has a winning strategy in $\mathcal{G}_{de}(q_0, q'_0)$, and we denote this $q_0 \preceq_{de} q'_0$. Accordingly, q_0, q'_0 delayed simulate each other if and only if $q_0 \preceq_{de} q'_0$ and $q'_0 \preceq_{de} q_0$, denoted $q_0 \simeq_{de} q'_0$.

As is shown in [EWS05], the relation $\preceq_{de}$ is a preorder, i.e., it is reflexive and transitive. This immediately implies that $\simeq_{de}$ is an equivalence relation. Moreover, it is shown that quotienting with respect to $\simeq_{de}$ preserves the recognized language, which is the basic observation needed to use delayed simulation for state space reduction of Büchi automata. The $\simeq_{de}$-quotient of a (nondeterministic) Büchi automaton is defined in the natural way: it has the state set $Q/_{\simeq_{de}}$ with initial state $[q_\mathrm{I}]$ (q_I is the initial state of $\mathcal{A}$) and

$$([q], a, [q']) \in \Delta/_{\simeq_{de}} :\Longleftrightarrow \exists q_0 \in [q], q'_0 \in [q'] \text{ such that } (q_0, a, q'_0) \in \Delta,$$

and the set $F/_{\simeq_{de}} := \{[q] \mid [q] \cap F \neq \varnothing\}$ of final states.

Lemma 3.20 ([EWS05]). *Let $\mathcal{A}$ be a Büchi automaton. Then $L(\mathcal{A}) = L(\mathcal{A}/_{\simeq_{de}})$ holds.*

We show that for deterministic Büchi automata delayed simulation can be computed by a reduction to the minimization problem for standard DFA. To do so, we first introduce the *delayed bisimulation game* $\mathcal{G}^{bi}_{de}$. It is defined similarly to $\mathcal{G}_{de}$, except for the following two differences: in each round of the play, Spoiler can choose freely which of the two pebbles he wants to move. Duplicator must choose the other pebble, afterwards. Moreover, the winning condition for Duplicator is modified. If a final state is seen at some position i in either run, then there must be a position $j \geq i$ such that a final state is seen at position j in the other run. If Duplicator has a winning strategy in $\mathcal{G}^{bi}_{de}(q_0, q'_0)$, then we say that q_0, q'_0 are *delayed bisimilar*, and we denote this $q_0 \approx^{bi}_{de} q'_0$.

We first prove that, for a deterministic Büchi automaton, two states delayed simulate each other if and only if they are delayed bisimilar. The major reason for this is the fact that the chosen transitions are uniquely determined by Spoiler's decisions. Hence, it makes no difference whether he moves in ρ or ρ', and Duplicator has only one possible move to answer with.

Lemma 3.21. *Let $\mathcal{A} = (Q, q_\mathrm{I}, \delta, F)$ be a deterministic Büchi automaton. Then, for all states $q_0, q_0' \in Q$ it holds*

$$q_0 \simeq_{de} q_0' \iff q_0 \approx_{de}^{bi} q_0'.$$

Proof. By the above remarks, $\mathcal{G}_{de}(q_0, q_0'), \mathcal{G}_{de}(q_0', q_0)$ and $\mathcal{G}_{de}^{bi}(q_0, q_0')$ can all be modeled on the *same* game graph G, which has as set of vertices the Cartesian product $Q \times Q$, and all vertices belong to Spoiler. Hence, it suffices to show that each path in G satisfies the winning condition of $\mathcal{G}_{de}(q_0, q_0')$ and $\mathcal{G}_{de}(q_0', q_0)$ if and only if it satisfies the winning condition of $\mathcal{G}_{de}^{bi}(q_0, q_0')$. For the implication from left to right, assume that for each i with $q_i \in F$ there exists $j \geq i$ such that $q_j' \in F$, and for each i with $q_i' \in F$ there exists $j \geq i$ such that $q_j \in F$. This is exactly the winning condition of $\mathcal{G}_{de}^{bi}(q_0, q_0')$. The reverse implication is shown analogously. $\square$

In [EWS05] it is proven that, even for nondeterministic Büchi automata, delayed bisimulation coincides with direct bisimulation after some preprocessing. The *direct bisimulation game* $\mathcal{G}_{di}^{bi}$ differs from the delayed version in that Duplicator has to precisely mimic Spoiler's visits to final vertices: for all i, it must hold that $q_i \in F$ if and only if $q_i' \in F$. If $q_0, q_0' \in Q$ are direct bisimilar, then we write $q_0 \approx_{di}^{bi} q_0'$.

To show how we can compute delayed bisimulation, let us first define the closure of a Büchi automaton.

Definition 3.22. Let $\mathcal{A} = (Q, q_\mathrm{I}, \Delta, F)$ be a Büchi automaton over Σ. The *closure* of $\mathcal{A}$, denoted $\mathrm{cl}(\mathcal{A})$, is the Büchi automaton $\mathcal{A} = (Q, q_\mathrm{I}, \Delta, F')$ over Σ, where we initially set $F' := F$ and then iterate the following until a fixed point is reached:

If there exists $q \notin F'$ such that all successors of q are in F', then put q in F'.

Note that $\mathrm{cl}(\mathcal{A})$ can be computed from $\mathcal{A}$ in time $\mathcal{O}(|\Sigma| \cdot n)$, if n is the number of states of $\mathcal{A}$, and that $\mathcal{A}$ and $\mathrm{cl}(\mathcal{A})$ accept the same language. The inclusion $L(\mathcal{A}) \subseteq L(\mathrm{cl}(\mathcal{A}))$ is immediately clear, because $F \subseteq F'$. For the reverse inclusion, note that from each state in $F' \setminus F$ a state in F must be seen in every run of $\mathcal{A}$, after traversing at most $|Q|$ transitions. Thus, each accepting run in $\mathrm{cl}(\mathcal{A})$ has an accepting counterpart in $\mathcal{A}$.

The following result yields the correspondence between delayed bisimulation and direct bisimulation.

Lemma 3.23 ([EWS05])**.** *Let $\mathcal{A}$ be a Büchi automaton and q_0, q_0' states of $\mathcal{A}$. Then, it holds*

$$q_0 \approx_{de}^{bi} q_0' \text{ in } \mathcal{A} \iff q_0 \approx_{di}^{bi} q_0' \text{ in } \mathrm{cl}(\mathcal{A}).$$

Proof. We first show that each winning strategy f for Duplicator in the delayed bisimulation game on $\mathcal{A}$ is a winning strategy for her in the direct bisimulation game on $\mathrm{cl}(\mathcal{A})$, i.e., $q_i \in F'$ if and only if $q'_i \in F'$, for all $i \in \mathbb{N}$. For a contradiction, assume w.l.o.g. that $q_j \in F'$ and $q'_j \notin F'$, for some j. Since $q'_j \notin F'$ there exists an infinite path from q'_j without a state in F' on it; since $F' \supseteq F$, this path neither has a state in F on it. Pick a strategy g for Spoiler (in the delayed bisimulation game on $\mathcal{A}$) playing exactly this path. Since $q_j \in F'$ there must be a state in F on each infinite path from q_j. This means, if Spoiler plays the strategy g in the delayed bisimulation game on $\mathcal{A}$, then from q_j a state in F must be reached after a finite number of transitions, whereas from q'_j no state in F is ever reached. This means that Spoiler wins the delayed bisimulation game on $\mathcal{A}$; a contradiction.

Conversely, let f be a winning strategy for Duplicator in the direct bisimulation game on $\mathrm{cl}(\mathcal{A})$ from (q_0, q'_0). To show that f is winning for her in the delayed bisimulation game on $\mathcal{A}$, we prove, for each $i \in \mathbb{N}$, that $q_i \in F$ implies the existence of $j \geq i$ such that $q'_j \in F$. By definition, if $q_i \in F$ then $q_i \in F'$; since f is a winning strategy for Duplicator in the direct bisimulation game on $\mathrm{cl}(\mathcal{A})$, this implies $q'_i \in F'$. We distinguish two cases: if $q'_i \in F$, then we set $j := i$, i.e., we take q'_i as the state we were looking for. Otherwise, it holds $q'_i \in F' \setminus F$. This means that q'_i must have been added to F' in some iteration of the computation of F', say in the l-th iteration, and all successors of q'_i must have been in F' after iteration $l-1$. This yields a situation for the successors of q'_i and iteration $l-1$ analogous to that of q'_i and iteration l. We repeat this argument at most l times, so we eventually leave $F' \setminus F$ and reach a state in F. Hence, on each infinite path from q'_i we reach a candidate for q'_j. $\square$

Note that, for a deterministic Büchi automaton $\mathcal{A}$, the computation of the direct bisimulation relation can be done in the same way as block partitioning for the DFA $\mathcal{A}$. To this end, let $\approx_{\mathcal{A}}$ be defined as on page 58 and observe the following: if there exists $w \in \Sigma^*$ separating q and q', then Spoiler wins $\mathcal{G}^{bi}_{di}$ by choosing the sequence of transitions labeled with $w\alpha$, for any $\alpha \in \Sigma^\omega$. Conversely, if $q \approx_{\mathcal{A}} q'$, then in all possible plays of $\mathcal{G}^{bi}_{di}$ the states visited in a round are either both final or non-final; hence, Duplicator wins.

The construction of $\mathrm{cl}(\mathcal{A})$ can be done in time $\mathcal{O}(n)$ and determining the relation $\approx_{\mathrm{cl}(\mathcal{A})}$ takes time $\mathcal{O}(n \cdot \log n)$.[5]

Theorem 3.24. *Let $\mathcal{A}$ be a deterministic Büchi automaton with n states and $\simeq_{de}$ the delayed simulation relation on the state space of $\mathcal{A}$. Then, $\simeq_{de}$ can be computed in time $\mathcal{O}(n \cdot \log n)$.*

[5]Recall that $|\Sigma|$ is assumed constant.

3.4.2 Reduction of Büchi Game Automata

By Lemmas 3.21 and 3.23 and the above explanations, we have proven that, for a Büchi game automaton $\mathcal{A}$, the computation of $\simeq_{de}$ in $\mathcal{A}$ can be reduced to the computation of $\approx_{di}^{bi}$ in $\mathrm{cl}(\mathcal{A})$, which is the same as $\approx_{\mathrm{cl}(\mathcal{A})}$ in the DFA $\mathrm{cl}(\mathcal{A})$. Since $\mathcal{A}$ and $\mathrm{cl}(\mathcal{A})$ are equivalent and $\simeq_{de}$ in $\mathcal{A}$ coincides with $\approx_{\mathrm{cl}(\mathcal{A})}$ in $\mathrm{cl}(\mathcal{A})$, we conclude that $\mathcal{A}/_{\simeq_{de}}$ is equivalent to $\mathrm{cl}(\mathcal{A})/_{\approx_{\mathrm{cl}(\mathcal{A})}}$. Hence, we can proceed by considering only $\mathrm{cl}(\mathcal{A})$ instead of $\mathcal{A}$.

In order to use these findings for our memory reduction algorithm, we verify that $\approx_{\mathrm{cl}(\mathcal{A})}$ is compatible with (the Büchi automaton) $\mathrm{cl}(\mathcal{A})$: item 1 of Definition 3.6 follows from the proof of Lemma 3.17. Item 2 holds as well, because equivalent states are either both final or both non-final. Accordingly, if $\rho = \rho(0)\rho(1)\rho(2)\cdots$ and $\rho' = \rho'(0)\rho'(1)\rho'(2)\cdots$ are two runs in $\mathrm{cl}(\mathcal{A})$ such that for all $i \in \mathbb{N}$ it holds $\rho(i) \approx_{\mathrm{cl}(\mathcal{A})} \rho'(i)$ in the DFA $\mathrm{cl}(\mathcal{A})$, then ρ is accepting if and only if ρ' is accepting.

Lemma 3.25. *Let $\mathcal{A}$ be a Büchi game automaton and $\approx_{\mathrm{cl}(\mathcal{A})}$ the state equivalence relation of the DFA* $\mathrm{cl}(\mathcal{A})$*. Then $\approx_{\mathrm{cl}(\mathcal{A})}$ is compatible with* $\mathrm{cl}(\mathcal{A})$.

For two memory contents $s_1, s_2 \in S$, we declare $s_1 \approx_S s_2$ if for all $v \in V$ it holds $(s_1, v) \approx_{\mathrm{cl}(\mathcal{A})} (s_2, v)$. We compute the quotient automaton of $\mathrm{cl}(\mathcal{A})$ induced by $\approx_S$, according to Definition 3.8. Since the equivalence $s_1 \approx_S s_2$ implies $(s_1, v) \approx_{\mathrm{cl}(\mathcal{A})} (s_2, v)$ for all $v \in V$, both $(s_1, v), (s_2, v)$ are either final or non-final. Let F be the set of final states in $\mathrm{cl}(\mathcal{A})$. Then, we can choose the following set $F/_{\approx_S}$ of final states in $\mathrm{cl}(\mathcal{A})/_{\approx_S}$: for $[s] \in S/_{\approx_S}, v \in V$, let

$$([s], v) \in F/_{\approx_S} :\Longleftrightarrow (s, v) \in F.$$

From the above results we deduce that a run of a Büchi game automaton $\mathcal{A}$ is accepting if and only if the corresponding run of $\mathrm{cl}(\mathcal{A})/_{\approx_S}$ is accepting. This means that Definition 3.8 is satisfied and $\mathrm{cl}(\mathcal{A})/_{\approx_S}$ is a Büchi game automaton equivalent to $\mathcal{A}$ (cf. Remark 3.9).

3.4.3 A Memory Reduction Algorithm for Request-Response Games

Given a Request-Response game $\Gamma = (G, \Omega)$, we apply the game simulation from page 30, obtaining a Büchi game Γ'. Subsequently, we transform Γ' into the Büchi game automaton $\mathcal{A}$ and compute $\mathrm{cl}(\mathcal{A})$. In $\mathrm{cl}(\mathcal{A})$ we compute the relation $\approx_{\mathrm{cl}(\mathcal{A})}$ of the DFA $\mathrm{cl}(\mathcal{A})$. From $\approx_{\mathrm{cl}(\mathcal{A})}$ we compute $\approx_S$ and the quotient automaton $\mathrm{cl}(\mathcal{A})/_{\approx_S}$. Subsuming our results, we can reformulate Theorem 3.10 as follows.

Theorem 3.26. *Let Γ be a Request-Response game and Γ' the expanded Büchi game according to page 30. Let $\mathcal{A}$ be the game automaton of Γ' and $\approx_{\mathrm{cl}(\mathcal{A})}$ the state equivalence relation in the DFA $\mathrm{cl}(\mathcal{A})$. Then Γ is simulated by the automaton game Γ'' of $\mathcal{A}/_{\approx_S}$.*

In fact, the above theorem reveals that our technique for memory reduction can also be applied to other games which can be simulated by Büchi games, for example generalized Büchi or upwards-closed Muller games.

Algorithm 3.4 (Memory Reduction for Request-Response games)

Input: Request-Response game $\Gamma = (G, \Omega)$
Output: Strategy automaton $\mathcal{A}_f$ for Player 0 from W_0

```
1: Establish a game simulation of $\Gamma$ by a Büchi game $\Gamma'$.
2: Transform $\Gamma'$ into the Büchi game automaton $\mathcal{A}$.
3: Compute $\mathrm{cl}(\mathcal{A})$, the closure of $\mathcal{A}$, say with the set $F$ of final vertices.
4: Consider $\mathrm{cl}(\mathcal{A})$ as a DFA and compute the state equivalence relation $\approx_{\mathrm{cl}(\mathcal{A})}$;
   refine $\approx_{\mathrm{cl}(\mathcal{A})}$ to obtain $\approx_S$ and construct the quotient Büchi game automa-
   ton $\mathcal{B} := \mathrm{cl}(\mathcal{A})/_{\approx_S}$ with the set $F/_{\approx_S}$ of final vertices.
5: View $\mathcal{B}$ as Büchi automaton game $\Gamma''$.
6: Compute a positional winning strategy for Player 0 in $\Gamma''$ and from it con-
   struct $\mathcal{A}_f$.
```

Note that we measure the running time of our algorithm in the size of Γ'. In Section 3.4.1, we have shown that the delayed simulation relation of a deterministic Büchi automaton can be computed in time $\mathcal{O}(n \cdot \log n)$, where n is the number of states and $|\Sigma|$ is assumed constant. Here, we get a complexity of $\mathcal{O}(n \cdot (\log n)^2)$, because we have $\mathcal{O}(\log n)$ input letters. The Büchi game Γ'' can be solved in time $\mathcal{O}(n^2 \cdot \log n)$. Hence, Algorithm 3.4 runs in time $\mathcal{O}(n^2 \cdot \log n)$, i.e., polynomial in $|\Gamma'|$, and it asymptotically requires no more time than solving Γ'.

3.5 Muller and Streett Conditions

In this section, we present our approach to memory reduction for both Muller and Streett conditions. It differs only in step 1 (of Algorithm 3.1), where we apply game simulations via LAR for Muller games and via IAR for Streett games, but we obtain a parity game in both cases. Accordingly, steps 2 through 5 are then dealt with at once.

We use the *right-hand delayed simulation* for alternating parity automata introduced in [FW06] to reduce parity game automata. Delayed simulation for the parity condition takes into account not only whether visited vertices are colored even or odd, but also the actual colors. Whereas for Büchi game automata the problem of computing delayed simulation can be reduced to the minimization problem for standard DFA (cf. Section 3.4.1), for parity automata the situation is much more involved: the simulation game has to be solved explicitly. It is equipped with a *priority memory* keeping track of the "pending obligations" for Duplicator. If she visits a color which, for satisfying the parity condition, is at least as good as the current obligation imposed by Spoiler, then the priority memory is reset and a final state is visited; accordingly, the simulation game has a Büchi winning condition (for Duplicator).

In our setting, we can use a simplification of the game described in [FW06]. First of all, a parity game automaton is not alternating which means that the first move of each round is made by Spoiler in the simulated automaton and the second move is made by Duplicator in the simulating automaton. Due to this fixed policy of moving the pebbles, we need less vertices in the simulation game graph. Secondly, a game automaton is deterministic. This implies that the positions of the two pebbles and the update of the priority memory are uniquely determined by a letter chosen by Spoiler. Accordingly, this letter needs not be stored and all vertices in the simulation game graph belong to Spoiler.

In the next section we formally introduce the delayed simulation game. For brevity, we only present the simplified version for parity game automata.

3.5.1 Delayed Simulation for Parity Game Automata

We are given a parity game automaton $\mathcal{A} = ((S \times V) \,\dot{\cup}\, \{q_0, q_{\text{sink}}\}, q_0, \delta, c, V_0)$ over V where a run ρ of $\mathcal{A}$ is accepting if the maximal color seen infinitely often in ρ is even.

We construct the *right-hand*[6] *delayed simulation game* $\mathcal{G}^{rh}_{de} := (G^{rh}_{de}, \varphi^{rh}_{de})$ as follows: the game graph $\mathcal{G}^{rh}_{de} := (V^{rh}_{de}, E^{rh}_{de})$ has the set of vertices

$$V^{rh}_{de} := (S \times V) \times (S \times V) \times (c(S \times V) \,\dot{\cup}\, \{\checkmark\});$$

the third component of a vertex is called *priority memory*. We set $V_{\text{Sp}} := V^{rh}_{de}$

[6]The notion of delayed simulation presented in [FW06] is called *right-hand* delayed simulation in [Fri05]. The intuition behind "right-hand" is that the simulating automaton, i.e., the automaton on the right-hand side of the relation symbol, triggers a reset of the priority memory. In [Fri05], other definitions of delayed simulation for the parity condition are given as well, but quotienting with respect to these relations does not preserve the recognized language. However, *right-hand* delayed simulation preserves the recognized language. For further explanations and formal proofs we refer to [Fri05, FW06].

(and $V_{\mathrm{Du}} := \varnothing$). The edge relation $E^{rh}_{de} \subseteq V^{rh}_{de} \times V^{rh}_{de}$ is defined as follows: $(((s_1, v_1), (s_2, v_2), k), ((s'_1, v'_1), (s'_2, v'_2), k')) \in E^{rh}_{de} :\Longleftrightarrow$

- $\delta((s_i, v_i), v'_i) = (s'_i, v'_i)$, for $i = 1, 2$
- $v'_1 = v'_2$
- $k' = \mathrm{pm}(c(s'_1, v'_1), c(s'_2, v'_2), k)$

In the work of Fritz and Wilke a min-parity condition is assumed. Since we deal only with max-parity games the notions introduced below are adjusted appropriately. First, let us define the priority memory update $\mathrm{pm} : \mathbb{N} \times \mathbb{N} \times (\mathbb{N} \uplus \{\checkmark\}) \to \mathbb{N} \uplus \{\checkmark\}$ as follows:

i. $\mathrm{pm}(i, j, \checkmark) = \max\{i, j\}$, if $j \prec i$

ii. $\mathrm{pm}(i, j, \checkmark) = \checkmark$, if $i \preceq j$

iii. $\mathrm{pm}(i, j, k) = \max\{i, j, k\}$, if $j \prec i$

iv. $\mathrm{pm}(i, j, k) = k$, if $i \preceq j$, i is odd and $k \leq i$, and (j is odd or $j < k$)

v. $\mathrm{pm}(i, j, k) = \checkmark$, if $i \preceq j$, j is even and $k \leq j$, and (i is even or $i < k$)

vi. $\mathrm{pm}(i, j, k) = \checkmark$, if i is odd, j is even, and both $k \leq i$ and $k \leq j$

vii. else $\mathrm{pm}(i, j, k) = k$

The binary relation $\prec$ is the *reward order* on $\mathbb{N}$. We define $n \preceq m$ if

- m is even and n is odd, or
- m and n are both even and $n \leq m$, or
- m and n are both odd and $m \leq n$.

This yields $\ldots \prec 5 \prec 3 \prec 1 \prec 0 \prec 2 \prec 4 \prec \ldots$; if $n \prec m$ then we say that m is *better* than n, whereas terms like *maximum* and *greater* refer to the standard relation $<$ on $\mathbb{N}$. We leave it up to the reader to verify that case vii of the definition of pm applies if and only if $i \preceq j, i < k$ and $j < k$.

The set φ^{rh}_{de} is determined by a Büchi winning condition. A play ϱ of $\mathcal{G}^{rh}_{de}$ is winning for Duplicator if the set

$$F := (S \times V) \times (S \times V) \times \{\checkmark\}$$

is visited infinitely often. This means Spoiler wins if he can avoid $\checkmark$ from a certain point onwards. The idea behind the Büchi winning condition is as follows:

assume ✓ disappears from the priority memory at some given point. Then, it can only be restored if in the simulating run we see a color which is both even and greater than (or equal to) the maximal color seen in the simulated run since the previous ✓. As a consequence, if the simulated run is accepting, then the simulating one is accepting as well.

Duplicator's winning region W_{Du} in $\mathcal{G}_{de}^{rh}$ determines the relation $\leq_{de}^{rh}$. Generally speaking, we write $(s_1, v_1) \leq_{de}^{rh} (s_2, v_2)$ and say that (s_2, v_2) *right-hand delayed simulates* (s_1, v_1) (short: "(s_2, v_2) delayed simulates (s_1, v_1)") if Duplicator has a winning strategy in $\mathcal{G}_{de}^{rh}$ from the initial game position $\mathrm{p_I}$ defined below. Let $i := c(s_1, v_1)$ and $j := c(s_2, v_2)$:

$$\mathrm{p_I}((s_1,v_1),(s_2,v_2)) := \begin{cases} ((s_1,v_1),(s_2,v_2),\max\{i,j\}) & \text{, if } j \prec i \\ ((s_1,v_1),(s_2,v_2),\checkmark) & \text{, otherwise} \end{cases}$$

Then, we set $(s_1, v_1) \leq_{de}^{rh} (s_2, v_2) :\iff \mathrm{p_I}((s_1,v_1),(s_2,v_2)) \in W_{\mathrm{Du}}$ and the corresponding equivalence relation[7] is defined as

$$(s_1,v_1) \approx_{de}^{rh} (s_2,v_2) :\iff (s_1,v_1) \leq_{de}^{rh} (s_2,v_2) \text{ and } (s_2,v_2) \leq_{de}^{rh} (s_1,v_1).$$

In our setting, we are only interested in pairs of states which have the same V-component. More precisely, we only consider initial game positions where it holds $v_1 = v_2$. We have to distinguish three cases depending on the colors of the considered states. Case (i) holds if $c(s_2, v) \prec c(s_1, v)$, case (ii) symmetrically holds if $c(s_1, v) \prec c(s_2, v)$. In the remaining case (iii) we have equal colors, i.e., it holds $c(s_1, v) = c(s_2, v)$. The following formalizes the above definition of $\mathrm{p_I}$ for all three cases:

$$\begin{aligned}
(s_1,v) \approx_{de}^{rh} (s_2,v) :\iff{} & (s_1,v) \leq_{de}^{rh} (s_2,v) \text{ and } (s_2,v) \leq_{de}^{rh} (s_1,v) \\
\underset{\text{case (i)}}{\iff}{} & ((s_1,v),(s_2,v),\max\{c(s_1,v),c(s_2,v)\}) \in W_{\mathrm{Du}} \text{ and} \\
& ((s_2,v),(s_1,v),\checkmark) \in W_{\mathrm{Du}} \\
\underset{\text{case (ii)}}{\text{or}}{} & ((s_1,v),(s_2,v),\checkmark) \in W_{\mathrm{Du}} \text{ and} \\
& ((s_2,v),(s_1,v),\max\{c(s_1,v),c(s_2,v)\}) \in W_{\mathrm{Du}} \\
\underset{\text{case (iii)}}{\text{or}}{} & ((s_1,v),(s_2,v),\checkmark) \in W_{\mathrm{Du}} \text{ and} \\
& ((s_2,v),(s_1,v),\checkmark) \in W_{\mathrm{Du}}
\end{aligned}$$

3.5.2 Quotienting

We show that $\approx_{de}^{rh}$ is compatible with the underlying parity game automaton. Item 1 of Definition 3.6 is verified by the upcoming lemma; item 2 follows

[7]The relation $\leq_{de}^{rh}$ is a preorder, i.e., transitive and reflexive [FW06].

from the fact that quotienting with respect to $\approx_{de}^{rh}$ is language-preserving, as is proven in [Fri05].

Lemma 3.27. *Let $\mathcal{A}$ be a parity game automaton and $\approx_{de}^{rh}$ be defined as in Section 3.5.1. Then, for all $s_1, s_2 \in S, v_1, v_2, v' \in V$ it holds:*

$$(s_1, v_1) \approx_{de}^{rh} (s_2, v_2) \Longrightarrow \delta((s_1, v_1), v') \approx_{de}^{rh} \delta((s_2, v_2), v')$$

Proof. We show the equivalent

$$\delta((s_1, v_1), v') \not\approx_{de}^{rh} \delta((s_2, v_2), v') \Longrightarrow (s_1, v_1) \not\approx_{de}^{rh} (s_2, v_2).$$

By symmetry, it suffices to show the following implication $(*)$:

$$\delta((s_1, v_1), v') \not\leq_{de}^{rh} \delta((s_2, v_2), v') \Longrightarrow (s_1, v_1) \not\leq_{de}^{rh} (s_2, v_2)$$

By definition, the premise of $(*)$ means that Spoiler has a winning strategy in $\mathcal{G}_{de}^{rh}$ from the initial game position $p_1 := \mathrm{p_I}(\delta((s_1, v_1), v'), \delta((s_2, v_2), v'))$. This implies that from p_1 there exists a word α such that on the path through G_{de}^{rh} (uniquely determined by α) the priority memory content $\checkmark$ is avoided from a certain point onwards.[8] We claim that choosing the word $v'\alpha$ is a winning strategy for Spoiler in $\mathcal{G}_{de}^{rh}$ from $p_2 := \mathrm{p_I}((s_1, v_1), (s_2, v_2))$.

Let $\varrho = \varrho(1)\varrho(2)\cdots$ be the play from p_1 and $\varrho' = \varrho'(0)\varrho'(1)\varrho'(2)\cdots$ the play from p_2. We get that $\mathrm{pr}_{1,2}(\varrho(s)) = \mathrm{pr}_{1,2}(\varrho'(s))$, for all $s \geq 1$, where $\mathrm{pr}_{1,2}$ is the projection on the first two components. In the proof we only mention the third components, i.e., the priority memory. The premise of $(*)$ can be reformulated as:

In ϱ the symbol $\checkmark$ is seen only finitely often.

If $\mathrm{pr}_3(\varrho(s)) = \mathrm{pr}_3(\varrho'(s))$ for some $s \geq 1$, then $\varrho(s) = \varrho'(s)$ which means that the plays ϱ and ϱ' coincide from position s onwards. Since ϱ is winning for Spoiler by assumption and we have a Büchi winning condition, this implies that ϱ' is winning for him as well, and we are done. We call this the trivial case.

Let r be the last position where $\checkmark$ appears in ϱ. By induction, we show that there exists no position $s > r$ where ϱ' has $\checkmark$ in the third component. Moreover, we prove the additional assertion that at each position $s > r$ the priority memory of $\varrho(s)$ is less than the priority memory of $\varrho'(s)$.

For $s \geq 1$, let $k_s := \mathrm{pr}_3(\varrho(s))$ denote the priority memory at position s in the play ϱ (k'_s analogously for ϱ') and let $i_s := c(\mathrm{pr}_1(\varrho(s))) = c(\mathrm{pr}_1(\varrho'(s)))$

[8]Note that all vertices in G_{de}^{rh} belong to Spoiler.

be the color of the state at position s in the simulated run, and analogously with j_s in the simulating run. Since $k_r = \checkmark$ and $k_r \neq k'_r$[9], we get $k'_r \in c(S \times V)$. By assumption, we also have $k_{r+1} \in c(S \times V)$. Generally speaking, for any two successive $\checkmark$, say at position s_1 and s_2, the sequence $(k_s)_{s_1<s<s_2}$ is weakly increasing (see the definition of pm). In our case, this means that the sequence $(k_s)_{r<s}$ eventually becomes stationary because $c(S \times V)$ is finite.

For the induction start, we show that $k'_{r+1} \in c(S \times V)$, where the exact value depends on i_{r+1}, j_{r+1} and k'_r. Since $k_r = \checkmark$ and $k_{r+1} \in c(S \times V)$, case i of the definition of pm must apply in ϱ, which means $j_{r+1} \prec i_{r+1}$ and $k_{r+1} := \max\{i_{r+1}, j_{r+1}\}$. This implies that in ϱ' case iii must apply because $k'_r \neq \checkmark$, by assumption. Hence, $k'_{r+1} = \max\{i_{r+1}, j_{r+1}, k'_r\} = \max\{k'_r, k_{r+1}\} \in c(S \times V)$. If $k_{r+1} \geq k'_r$ then we get the trivial case, whereas $k_{r+1} < k'_r$ implies $k'_{r+1} = k'_r$. This also proves the additional claim $k_{r+1} < k'_{r+1}$. For the induction step, we assume $k_s, k_{s+1}, k'_s \in c(S \times V), k_s < k'_s$ and distinguish the cases $k_s < k_{s+1}$ and $k_s = k_{s+1}$ (for $s > r$):

- $k_s < k_{s+1}$: the priority memory value can increase only in case iii (cf. definition of pm), which means $j_{s+1} \prec i_{s+1}$ and $k_{s+1} := \max\{i_{s+1}, j_{s+1}, k_s\}$. Since $k_s \neq k_{s+1}$, at least one of the values i_{s+1} or j_{s+1} must be strictly greater than k_s. Hence, it holds $k_{s+1} = \max\{i_{s+1}, j_{s+1}\}$. By the induction hypothesis, it holds $k'_s \in c(S \times V)$. Hence, case iii also applies in ϱ' and we get $k'_{s+1} := \max\{i_{s+1}, j_{s+1}, k'_s\}$. If $k'_s < k'_{s+1}$ then, for the same argument as above, we get $k'_{s+1} = \max\{i_{s+1}, j_{s+1}\}$, implying $k_{s+1} = k'_{s+1}$, i.e., the trivial case holds. Otherwise, we get $k'_{s+1} = k'_s \in c(S \times V)$. Since $k_s < k'_s$ we have $k_{s+1} = \max\{i_{s+1}, j_{s+1}, k_s\} \leq \max\{i_{s+1}, j_{s+1}, k'_s\} = k'_{s+1}$, and the assumption $k_{s+1} \neq k'_{s+1}$ yields the additional claim $k_{s+1} < k'_{s+1}$.
- $k_s = k_{s+1}$: exactly one of the cases iii,iv or vii of the definition of pm must apply in ϱ. We show that in each case it holds $k'_s = k'_{s+1}$. This also verifies the additional claim $k_{s+1} < k'_{s+1}$.
  - iii: We get $k_{s+1} := \max\{i_{s+1}, j_{s+1}, k_s\}$, which (together with the assumption $k_s = k_{s+1}$) implies $k_s \geq i_{s+1}$ and $k_s \geq j_{s+1}$. Furthermore, we have $j_{s+1} \prec i_{s+1}$ and since $k'_s \neq \checkmark$, case iii also applies in ϱ', which means $k'_{s+1} := \max\{i_{s+1}, j_{s+1}, k'_s\}$. The induction hypothesis $k'_s > k_s$ together with $k_s \geq i_{s+1}$ and $k_s \geq j_{s+1}$ yields $k'_s > i_{s+1}$ and $k'_s > j_{s+1}$, which implies $k'_{s+1} = k'_s$.
  - iv: In this case we have $i_{s+1} \preceq j_{s+1}$, which (together with the assumption $k'_s \neq \checkmark$) means that exactly one of the cases iv,v,vi or vii holds

[9]Otherwise, we have the trivial case and are done.

in ϱ'. We show that cases v and vi are impossible, which then implies $k'_{s+1} = k'_s$ (cf. cases iv and vii). To exclude case v, note that the conditions for that case imply that $j_{s+1} \geq k'_s$ and that j_{s+1} is even, which simultaneously with the conditions of case iv means that $k_s > j_{s+1}$. Altogether, this yields $k_s > j_{s+1} \geq k'_s$ contradicting the induction hypothesis $k_s < k'_s$. Since for case vi the conditions "$j_{s+1} \geq k'_s$" and "j_{s+1} is even" hold as well, we can deduce the same contradiction as for case v.

vii: By our remarks on case vii (cf. page 68), we have $i_{s+1} \preceq j_{s+1}$, and $k_s > i_{s+1}$ and $k_s > j_{s+1}$. As above, the cases iv,v,vi or vii are possible in ϱ'. In both the cases v and vi the condition $j_{s+1} \geq k'_s$ yields $k_s > k'_s$, a contradiction to the induction hypothesis.

We have shown that there exists no $s > r$ such that $k'_s = \checkmark$. Hence, the play ϱ' is winning for Spoiler, which means that he has a winning strategy from position p_2. Thereby, the implication $(*)$ is shown. □

Corollary 3.28. *Let $\mathcal{A}$ be a parity game automaton and $\approx^{rh}_{de}$ the delayed simulation relation for $\mathcal{A}$. Then $\approx^{rh}_{de}$ is compatible with $\mathcal{A}$.*

We compute $\approx_S$ from $\approx^{rh}_{de}$ (as in Definition 3.7) and express the acceptance condition $\psi/_{\approx_S}$ of $\mathcal{A}/_{\approx_S}$ in terms of a coloring $c/_{\approx_S}$. To this end, let $s \in S, v \in V$ and define

$$c/_{\approx_S}([s], v) := \max\{c(s', v') \mid (s', v') \approx^{rh}_{de} (s, v)\},$$

and let q_0, q_{sink} inherit their color from $\mathcal{A}$. Since $s_1 \approx_S s_2$ implies the equivalence $(s_1, v) \approx^{rh}_{de} (s_2, v)$ for all $v \in V$, the above definition of $c/_{\approx_S}$ is independent of representatives. Hence, $\mathcal{A}/_{\approx_S}$ is a parity game automaton.

In [Fri05], the $\approx^{rh}_{de}$-quotient $\mathcal{A}/_{\approx^{rh}_{de}}$ is defined in a natural way. For our setting, this means $\delta/_{\approx^{rh}_{de}}([(s,v)], v') := [\delta((s,v), v')]$ and

$$c/_{\approx^{rh}_{de}}([(s,v)]) := \max\{c(s', v') \mid (s', v') \approx^{rh}_{de} (s, v)\}.$$

The author shows that quotienting with respect to $\approx^{rh}_{de}$ preserves the recognized language, i.e., it holds $L(\mathcal{A}) = L(\mathcal{A}/_{\approx^{rh}_{de}})$ (see [Fri05, FW06] for details); we use this result in the proof of the following lemma.

Lemma 3.29. *Let $\mathcal{A}$ be a parity game automaton and $\mathcal{A}/_{\approx_S}$ the corresponding $\approx_S$-quotient with coloring $c/_{\approx_S}$ (defined as above). Then $\mathcal{A}$ and $\mathcal{A}/_{\approx_S}$ are equivalent.*

Proof. We have to show that it holds $L(\mathcal{A}) = L(\mathcal{A}/_{\approx_S})$, where it suffices to show that $L(\mathcal{A}/_{\approx^{rh}_{de}}) = L(\mathcal{A}/_{\approx_S})$ holds. By Corollary 3.28, automaton $\mathcal{A}/_{\approx^{rh}_{de}}$ is deterministic, and by Definition 3.8, automaton $\mathcal{A}/_{\approx_S}$ is deterministic. For $\alpha \in V^\omega$, let ρ be the run of $\mathcal{A}/_{\approx_S}$ on α and ρ' be the corresponding run of $\mathcal{A}/_{\approx^{rh}_{de}}$ on α. The run ρ' is uniquely determined by the run ρ, because both $\mathcal{A}/_{\approx^{rh}_{de}}$ and $\mathcal{A}/_{\approx_S}$ are deterministic and $\approx_S$ is a refinement of $\approx^{rh}_{de}$. Moreover, ρ is accepting if and only if ρ' is accepting, because both runs have the same sequence of colors. Since both $\mathcal{A}/_{\approx_S}$ and $\mathcal{A}/_{\approx^{rh}_{de}}$ are deterministic, there is no other run on α, neither for $\mathcal{A}/_{\approx_S}$ nor for $\mathcal{A}/_{\approx^{rh}_{de}}$. Thus, α is accepted by $\mathcal{A}/_{\approx_S}$ if and only if it is accepted by $\mathcal{A}/_{\approx^{rh}_{de}}$. □

3.5.3 A Memory Reduction Algorithm for Muller and Streett Games

The previous results show that our algorithm for memory reduction is applicable to both Muller and Streett games as follows: we simulate Γ by a parity game Γ' which is then transformed into a parity game automaton $\mathcal{A}$. For $\mathcal{A}$ we construct the right-hand delayed simulation game $\mathcal{G}^{rh}_{de}$ and solve it by standard techniques (see for example [GTW02]). Duplicator's winning region in $\mathcal{G}^{rh}_{de}$ and Definition 3.7 uniquely determine $\approx_S$. The corresponding quotient automaton $\mathcal{A}/_{\approx_S}$ is a parity game automaton equivalent to $\mathcal{A}$, and we can transform it into a unique parity automaton game Γ''. By Theorem 3.10, Γ is simulated by Γ''.

Theorem 3.30. *Let Γ be a Muller or Streett game and Γ' the corresponding parity game (cf. pages 32,36). Further, let $\mathcal{A}$ be the game automaton of Γ' and $\approx^{rh}_{de}$ the right-hand delayed simulation. Then Γ is simulated by the automaton game Γ'' of $\mathcal{A}/_{\approx_S}$.*

Algorithm 3.5 (Memory Reduction for Muller and Streett games)

Input: Muller game $\Gamma = (G, \mathcal{F})$ or Streett game $\Gamma = (G, \Omega)$
Output: Strategy automaton $\mathcal{A}_f$ for Player 0 from W_0

```
1: Establish a game simulation of $\Gamma$ by a parity game $\Gamma'$.
2: Transform $\Gamma'$ into the parity game automaton $\mathcal{A}$, say with coloring $c$.
3: Construct the delayed simulation game $\mathcal{G}^{rh}_{de}$ for $\mathcal{A}$.
4: From Duplicator's winning region compute $\approx^{rh}_{de}$; refine $\approx^{rh}_{de}$ to obtain $\approx_S$
   and construct the quotient parity game automaton $\mathcal{B} := \mathcal{A}/_{\approx_S}$ with the
   coloring $c/_{\approx_S}$.
5: View $\mathcal{B}$ as parity automaton game $\Gamma''$.
6: Compute a positional winning strategy for Player 0 in $\Gamma''$ and from it con-
   struct $\mathcal{A}_f$.
```

At this point, we mention an optional normalization of the coloring c done before the execution of step 4; it is efficiently computable and may make the relation $\approx^{rh}_{de}$ larger. For each SCC C of $\mathcal{A}$, we iterate the following: while there exists $(s, v) \in C$ such that $c(s, v) \geq 2$ and there exists no $(s', v') \in C$ such that $c(s', v') = c(s, v) - 1$, do $c(s, v) := c(s, v) - 2$. Clearly, this does not change the accepted language [FW06].

Recall that we measure the running time of our algorithm in the size of Γ'. For the computation of $\approx^{rh}_{de}$, we need to solve the simulation game $\mathcal{G}^{rh}_{de}$. It is a Büchi game with $\mathcal{O}(n^2 \cdot k)$ vertices where n is the number of states of the parity game automaton $\mathcal{A}$ and k is the number of colors assigned by the coloring c. Since Büchi games are solvable in polynomial time (measured in the size of the game graph), the running time of steps 2 through 5 of Algorithm 3.5 is polynomial in $|\Gamma'|$. However, note that step 6 requires time $n^{\mathcal{O}(\sqrt{n})}$ [JPZ06].

3.6 Discussion

In this section we analyze strengths and weaknesses of our approach to memory reduction.

On the positive side, we prove the existence of particular games for which our new algorithm yields winning strategies of at most constant size, whereas standard algorithms compute very complicated strategies of exponential size. For the class of Staiger-Wagner conditions, we present an example where our technique results in an exponential gain in the memory size, measured in the size of the game graph. Analogous results are obtained for Request-Response and Streett conditions, but there the benefit is measured in the number of pairs of the winning condition Ω.

On the negative side, we point out a particular weakness of our approach, which is that for determining equivalent memory contents the behavior of the two players is not taken into account. We give an example of a Staiger-Wagner game where the game graph is of size $\mathcal{O}(n)$ and our algorithm produces a memory of size at least 2^n. However, Player 0 has a positional winning strategy which is found if the game is solved via game simulation.

3.6.1 Exponential Gains

In this section we present examples of Staiger-Wagner, Request-Response and Streett games where our algorithm reduces the size of the needed memory from exponential to constant. In each of the three cases, we first apply a game simulation yielding a game which admits positional winning strategies for both players. We show that the winning strategy computed for Player 0 in the simulating game yields a very complicated finite-state winning strategy in the simulated game. However, if we apply our new technique, then we obtain a very simple winning strategy.

For both Staiger-Wagner and Request-Response games, it will turn out that the major argument in our proofs is that an attractor strategy chooses the shortest way into the set of final vertices. Moreover, in Theorem 3.32 we utilize the fact that, for solving weak parity games, high colors are preferred to low ones. Both these weaknesses are ruled out by our approach to memory reduction.

Additionally, we present a result for Streett games similar to the aforementioned ones. We compute a game simulation by a parity game and make reasonable assumptions of the properties of a winning strategy constructed for Player 0 in this parity game; as a consequence, we get a memory of size exponential in the number of Streett pairs, but our algorithm reduces the memory to size one.

Let us start with an example of a Staiger-Wagner game.

Example 3.31. Consider the Staiger-Wagner game $\Gamma_n = (G_n, \mathcal{F}_n)$ with the game graph G_n depicted in Figure 3.6 and the winning condition

$$\mathcal{F}_n = \{U \mid \{v\} \subseteq U \subseteq \{v, u_1, \ldots, u_n\}\} \cup \{R \mid \{x, y\} \subseteq R\}.$$

Player 1 owns only vertex v, which we consider to be the initial vertex of the game. Player 0 wins if the play remains within $\{v, u_1, \ldots, u_n\}$ or reaches both vertices x and y.

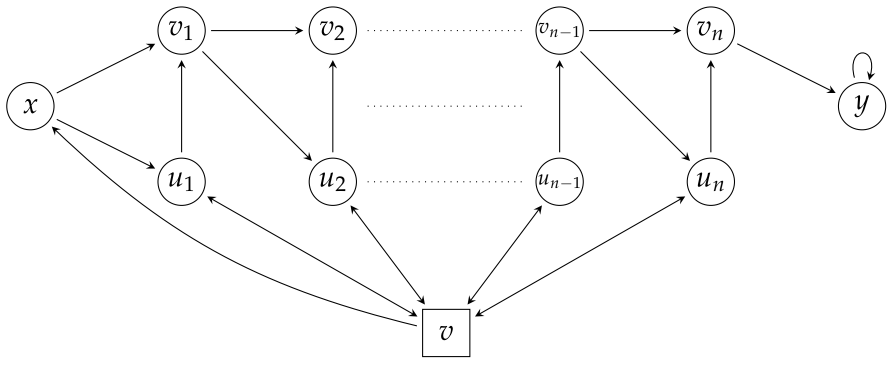

Figure 3.6: *Staiger-Wagner game graph G_n*

Simulating Γ_n by a weak parity game $\Gamma'_n = (G'_n, c_n)$, we obtain exponentially many reachable memory contents in G'_n. (A memory content s is *reachable* if there exist $v, v' \in V$ such that (s, v') is reachable from (s_0, v) via a finite path.)

If we solve Γ'_n with the algorithm from [Cha06], then this yields a winning strategy for Player 0 in Γ_n of size exponential in n. Note that the size of G_n is linear in n, while $\mathcal{F}_n$ contains exponentially many sets. If we use Algorithm 3.3 to reduce the size of G'_n before solving the game, then we obtain a winning strategy of constant size for Player 0 in Γ_n.

Theorem 3.32. *Let $\Gamma_n{=}(G_n, \mathcal{F}_n)$ be defined as in Example 3.31 and let $\Gamma'_n{=}(G'_n, c_n)$ be the expanded weak parity game obtained from Γ_n (according to page 28). Then, Player 0 wins Γ_n from vertex v such that the following hold:*

1. *The positional winning strategy for Player 0 in Γ'_n from $(\varnothing, v)$ computed by the algorithm from [Cha06] yields a winning strategy for Player 0 in Γ_n from v of size at least 2^n.*
2. *The equivalence relation $\approx_S$ computed from Γ_n by Algorithm 3.3 (see page 60) has five equivalence classes.*

Proof. The winning regions of Γ_n are the sets $W_0 = \{u_1, \ldots, u_n, v, x\}$ and $W_1 = \{v_1, \ldots, v_n, y\}$. We can assume that Player 1 eventually moves to vertex x; otherwise, the play stays within the set $\{u_1, \ldots, u_n, v\}$, which means that Player 0 wins. Thus, let a play be of the form $\varrho = v\varrho_1 x \varrho_2$. For abbreviation, we set $U_v := \{U \mid \{v\} \subseteq U \subseteq \{v, u_1, \ldots, u_n\}\}$.

If we apply the game simulation from page 28 to Γ_n, then we obtain a weak parity game Γ'_n where the game graph G'_n contains as memory the set of vertices visited in a play. For each color k, the algorithm for solving Γ'_n computes the set A_k from where a player can force a win by reaching a vertex of color at least k (cf. [Cha06] or Section 2.1.2). It starts with the highest color $m := 2(2n+3)$; the highest color is twice the number of vertices in G_n.

Note that from vertex x Player 0 has a strategy to move to vertex y such that all vertices in V_n are finally visited, namely via $u_1, v_1, \ldots, u_n, v_n$ (in the given order). As a consequence, the algorithm from [Cha06] will direct Player 0 to reach C_m, i.e., the vertices of color m, in Γ'_n. This is due to the fact that the set A_m is the first set to be computed (see the proof of Theorem 2.3). In particular, this means that the strategy computed for Γ_n in ϱ_2 visits each of the vertices u_i not visited in ϱ_1.

The important observation is how the strategy actually moves from vertex x to vertex y, namely by skipping all vertices u_i which have already been visited in ϱ_1. This is due to the fact that it searches the shortest way into C_k, because it is an *attractor* strategy. For example, if Player 1 moves from v directly to x without visiting any vertex u_i, then Player 0 visits each vertex u_i to reach C_k (in Γ'_n). This means that we get the play $\varrho = vxu_1v_1 \cdots u_nv_ny^\omega$. Contrariwise, if Player 1 visits each u_i before moving to x, then we get for example

$\varrho = vu_1v \cdots u_nvxv_1 \cdots v_ny^\omega$, because in this case the shortest way into C_k is to skip all vertices u_i.

Altogether, the set of vertices u_i visited in ϱ_1 uniquely determines the strategy for Player 0 from vertex x to vertex y. For each i, vertex u_i is visited in ϱ_2 if and only if it has not been visited in ϱ_1. Each of the 2^n sets in U_v yields a different strategy, each of which requires one state in the strategy automaton. Hence, this automaton is of size at least 2^n.

To see item **??** of the theorem, we have to determine the equivalence classes of $\approx_S$. First, we reduce the problem of computing $\approx$ for the weak parity game automaton $\mathcal{A}$ of Γ'_n to the problem of computing the standard equivalence relation $\approx_{\mathcal{A}'}$ for a DFA $\mathcal{A}'$ uniquely determined by $\mathcal{A}$ (cf. Section 3.3.2). This yields for two given states

$$(s_1, v) \approx (s_2, v) \iff L(\mathcal{A}_{(s_1,v)}) = L(\mathcal{A}_{(s_2,v)}),$$

which allows us to identify equivalence of states with winning the same plays from the corresponding vertices in the game. From $\approx$ we compute $\approx_S$ and obtain that the set $S/_{\approx_S}$ has five elements and that all sets in U_v are equivalent. Table 3.1 gives a review of the five equivalence classes. The sets from U_v are all in class three, which has $\{v, u_1\}$ as representative. Accordingly, Player 0 wins if the play stays in U_v or reaches both x and y; otherwise, Player 1 wins. Class five consists of all sets containing y, but not x. If a play in Γ'_n reaches the memory content $\{y\}$, then the corresponding play in Γ_n reaches vertex y. Thus, the only possible continuation of the play is y^ω and Player 0 loses.

Class	Representative	Plays winning for…	
		… Player 0	… Player 1
1	V	y^ω	None
2	$\{x\}$	Reach y	otherwise
3	$\{v, u_1\}$	Stay in U_v or reach x and y	otherwise
4	$\{v, u_1, v_1, u_2\}$	Reach both x and y	otherwise
5	$\{y\}$	None	y^ω

Table 3.1: *Equivalence classes of* $\approx_S$

□

Theorem 3.32 shows that there exist games with weak winning condition where our algorithm yields a substantial reduction of the needed memory.

Next, we extend this result to Request-Response specifications. For the proof, we utilize an argument similar to one from the previous proof. More precisely, we exploit the fact that attractor strategies sometimes are too hasty to reach a final vertex.

Example 3.33. Consider the Request-Response game $\Gamma_k = (G_k, \Omega_k)$ with the game graph G_k depicted in Figure 3.7 (for $k = 3$) and the winning condition Ω_k containing the following $2k+1$ pairs of sets:

$$\Omega_k = \{(P_0, R_0)\} \cup \{(P_1, R_1), (P_1', R_1'), \dots, (P_k, R_k), (P_k', R_k')\}$$

In Figure 3.7, all sets are written next to the vertices they contain; in particular, vertex y is contained in all response-sets. The play starts at vertex v. Player 1 makes k independent decisions, moving either to a vertex in P_i or P_i' ($i = 1, \dots, k$). At vertex w Player 0 takes over, making k decisions himself.

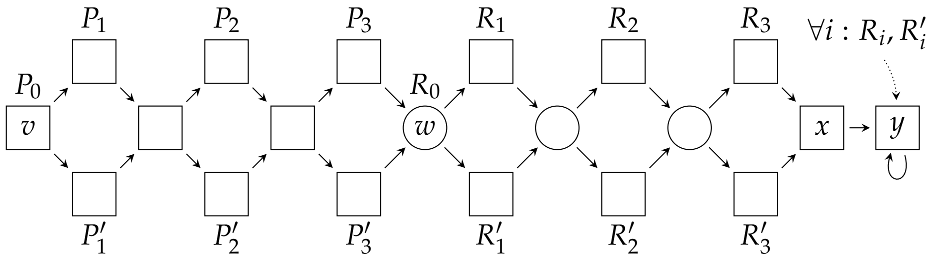

Figure 3.7: *Request-Response game graph G_3*

Note that each pair in Ω_k is eventually responded to, namely at vertex y; the pair (P_0, R_0) is already responded to at vertex w. Hence, every (positional) strategy for Player 0 is a winning strategy. However, we show that a standard algorithm computes a winning strategy of size exponential in k, while both G_k and Ω_k are of size linear in k.

Theorem 3.34. *Let $\Gamma_k = (G_k, \Omega_k)$ be the Request-Response game from Example 3.33 and let $\Gamma_k' = (G_k', F_k)$ be the expanded Büchi game obtained from Γ_k (according to page 30). Then, Player 0 wins Γ_k from vertex v such that the following hold:*

1. *The positional winning strategy for Player 0 in Γ_k' from $(\varnothing, 1, 0, v)$ computed by the algorithm described on page 23 yields a winning strategy for Player 0 in Γ_k from v of size at least 2^k.*

2. *The equivalence relation $\approx_S$ computed from Γ_k by Algorithm 3.4 (see page 66) has one equivalence class.*

Proof. If Player 0 precisely mimics the decisions of Player 1, i.e., she moves to the R_i-vertex if and only if Player 1 has moved to the P_i-vertex (for all $i = 1, \ldots, k$), then in the Büchi game the final vertex $(\varnothing, 1, 1, y)$ is visited after the first $4k+1$ moves. On the way from vertex w to vertex y, the cyclic counter in the memory is increased by one for $2k+1$ times: it starts with value 1 at vertex w and has value $2k+1$ when reaching vertex x; it is reset to value 1 when the play proceeds from vertex x to vertex y (in Γ_k) and the final vertex $(\varnothing, 1, 1, y)$ is visited (in Γ'_k).

If Player 0 makes a mistake, i.e., there exists $1 \leq j \leq k$ such that she moves to the R_j-vertex if and only if Player 1 has moved to the P'_j-vertex, then the final vertex $(\varnothing, 1, 1, y)$ is visited as well, but it takes longer to reach it than in the case where she plays "correctly". This is due to the fact that her false answer to the j-th request prevents the cyclic counter from being increased.

By our remarks above, there is a unique shortest path from vertex w to vertex y which visits a final vertex in the Büchi game Γ'_k. It is the path which precisely mimics Player 1's behavior between vertex v to vertex w. Solving the Büchi game we obtain an attractor strategy, which means that a final vertex is assumed as soon as possible. Hence, the strategy chooses the "correct" path from vertex w to vertex y (in Γ_k). This requires a memory of size at least 2^k because the strategy needs to memorize each of the k decisions of Player 1.

Since every request is eventually responded to (and never requested again), the set of active pairs becomes finally empty, and it remains empty for the rest of the play. This means that the cyclic counter is eventually reset to 1 and the final vertex $(\varnothing, 1, 1, y)$ is visited (on G'_k). Moreover, this vertex is repeatedly visited after the next $2k+1, 2 \cdot (2k+1), 3 \cdot (2k+1), \ldots$ moves. Summing up, every infinite path on G'_k visits a final vertex (infinitely often).

Let us now analyze Algorithm 3.4. Consider the Büchi game automaton $\mathcal{A}$ of Γ'_k (for arbitrary k). By the remarks above, every infinite path in $\mathcal{A}$ (without q_{sink}) leads through the final state $(\varnothing, 1, 1, y)$, even when starting at non-reachable states. Accordingly, all states in $S \times V$ are declared final in the closure $\mathrm{cl}(\mathcal{A})$ of $\mathcal{A}$ (cf. Definition 3.22). Thus, we obtain $\approx^{bi}_{di}$ in $\mathrm{cl}(\mathcal{A})$ to be the set

$$\{((s_1, v), (s_2, v)) \mid s_1, s_2 \in 2^{\{1,\ldots,k\}} \times \{1, \ldots, k\} \times \mathbb{B}, v \in V\}.$$

This implies that all memory contents are equivalent, i.e., $S/_{\approx_S}$ is a singleton. □

Let us show a similar result for the class of strong winning conditions. We consider the Streett game in the upcoming Example 3.35 and make natural assumptions of the winning strategy for Player 0 in the simulating parity game.

More precisely, we demand that she behaves "optimally". This is meant in the sense that she continuously chooses those edges which globally guarantee the best colors she can enforce. A color m is better than a color n if $n \prec m$, where $\prec$ is the reward order from page 68.

Example 3.35. Let G_k be the graph shown in Figure 3.8 (for $k = 3$) and Ω_k the following Streett winning condition:

$$\Omega_k = \{(E_1, F_1), (E_{-1}, F_{-1}), \dots, (E_k, F_k), (E_{-k}, F_{-k}), (V, V)\}$$

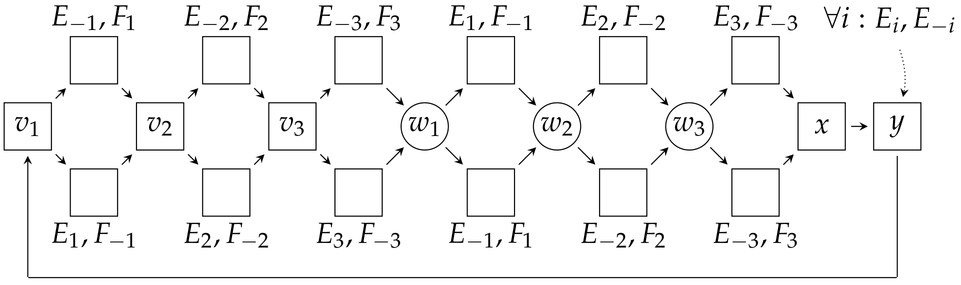

Figure 3.8: *Streett game graph G_3*

As in Figure 3.7, all sets are written next to the vertices they contain. The game starts at vertex v_1 and proceeds similarly to the one from Example 3.33. The major difference is that vertex v_1 is visited infinitely often, naturally dividing each play into rounds. At the end of each round, i.e., when the play proceeds from vertex y to vertex v_1, the highest possible color $4k{+}2$ is seen in the simulating parity game Γ'_k. This is due to the fact that some index must be at the last position of the current IAR and, accordingly, the pointer e' has the value $2k{+}1$ (cf. page 36). Thus, each play satisfies the parity winning condition, and each (positional) strategy for Player 0 is winning.

For a better understanding of item 1 of the following theorem, the reader should recall (from page 17) that a positional strategy f can be represented by the following set of edges: $E_f := \{(v, f(v)) \mid v \text{ is a vertex owned by Player } 0\}$.

Theorem 3.36. *Let $\Gamma_k = (G_k, \Omega_k)$ be the Streett game from Example 3.35 and let $\Gamma'_k = (G'_k, c_k)$ be the expanded parity game obtained from Γ_k (according to page 36) where $s_0 := ((1 \cdots 2k+1), 1, 1)$ is the initial memory content. Then, Player 0 wins Γ_k from vertex v_1 such that the following hold:*

1. *Each positional winning strategy f'_k for Player 0 in Γ'_k from (s_0, v_1) with*

$$\{c_k(s', v') \mid ((s, v), (s', v')) \in E_{f'_k}\} \cap \{2n + 1 \mid n \in \mathbb{N}\} = \varnothing$$

yields a winning strategy f_k for Player 0 in Γ_k from v_1 of size at least 2^k.

2. *The equivalence relation $\approx_S$ computed from Γ_k by Algorithm 3.5 (see page 73) has one equivalence class.*

Proof. For simplicity, we assume $k = 3$; the proof is analogous for other values of k. First of all, we fix a convention on the entries in an IAR: let the pair (V, V) be represented by V and every other pair by its unique index: for example, (E_{-2}, F_{-2}) is represented by -2; if index i has value $-j$, then we set $-i{:=}j$, for $1 \le j \le 3$.

The intersection in item 1 means that Player 0 chooses only edges leading into vertices of even color. To prove that item, note that in any round the permutation reached (in Γ'_3) when vertex w_1 is reached (in Γ_3) is of the form

$$(V\ i_3\ i_2\ i_1\ p),$$

where $i_j \in \{j, -j\}$ for $1 \le j \le 3$, and p is some permutation of $\{-i_1, -i_2, -i_3\}$. For example, let Player 1 at v_1 decide to move up, then again up, and then down; then the permutation is $(V\ 3\ -2\ -1\ p)$, because the indices -1, -2 and 3 are shifted to the second position one after another, and V stays at the front (cf. page 36). Moreover, p is a permutation of $\{1, 2, -3\}$, i.e., the set of all indices i for which Player 1 has moved to E_{-i} and F_i, recently.

If Player 0 moves upwards at w_1, i.e., she mimics Player 1's behavior at vertex v_1, then the permutation becomes $(V\ 1\ 3\ -2\ -1\ p')$, and p' is either $(2\ -3)$ or $(-3\ 2)$. That means, e' is assigned the value 5, 6 or 7.[10] Simultaneously, f' gets the value 5 because F_{-1} is visited and -1 is at the fifth position in the new permutation. Accordingly, it holds $e' \ge f'$, which means that we see color 10, 12 or 14.

Conversely, if Player 0 moves downwards at w_1, then the permutation becomes $(V\ -1\ 3\ -2\ p'')$, for some appropriate p''. Hence, e' is assigned 4 because index -1 comes from the fourth position. Moreover, index 1 is located somewhere in p'', either at position 5, 6 or 7. Thus, we have $f' \ge 5 > 4 = e'$; accordingly, we see an odd color: either 9, 11 or 13.

Making an analogous observation at vertices w_2, w_3, we can deduce the following: if Player 0 mimics Player 1's behavior, then she visits an even color, say l; if she makes the "wrong" move, then a vertex of odd color less than l is seen. Calling the latter situation an *error*, note that Player 0 can play errorless by memorizing Player 1's decisions. By an argument analogous to that in

[10]Note that the definition of e' refers to the old permutation, and index 1 comes from position 5, 6 or 7 (cf. page 36).

the proof of Theorem 3.34, implementation of an errorless winning strategy requires a memory of size at least 2^k.

To see item **??**, let us consider Algorithm 3.5. Every play on G'_k must traverse an edge $((s_1, y), (s_2, v_1))$ infinitely often, for some IARs s_1, s_2. Thereby, a vertex with the highest possible color $4k + 2$ is visited, because there must be $1 \leq j \leq k$ such that the index j or $-j$ is at the last position of the IAR s_1. Thus, in the simulation game the priority memory is reset to $\checkmark$ and a final vertex is visited, infinitely often. Accordingly, Duplicator has a winning strategy from each vertex in the simulation game graph. Summing up, all states (having the same V-component) in the parity game automaton $\mathcal{A}$ of Γ'_k are $\approx^{rh}_{de}$-equivalent, which means that all memory contents are declared $\approx_S$-equivalent. Thus, we obtain a reduced memory of size one. □

3.6.2 Where Game Automata are too Weak

In the previous section, we have shown that there exist games where our approach yields a substantial reduction of the required memory. For the sake of completeness, in the present section we give a negative example: our algorithm computes a memory of exponential size. However, a positional winning strategy is computed when the reduced game is solved.

Example 3.37. Consider the game graph G_n depicted in Figure 3.9 and the following Staiger-Wagner winning condition:

$$\mathcal{F}_n = \{R \mid u_i \in R \iff x_i \in R \text{ and } w_i \in R \iff z_i \in R, i = 1, \ldots, n\} \cup$$

$$\{R \mid y'_i \in R, i = 1, \ldots, n\}$$

Let us first explain possible winning strategies, by analyzing the two types of sets contained in the winning condition. The first kind covers those plays where Player 0 mimics Player 1's behavior. At each vertex y_i she has to choose x_i or z_i, and she has to move to x_i if and only if Player 1 has moved to u_i. We call this the "mimic" strategy; clearly, it is winning, but it requires a memory of size at least 2^n.

The second kind of sets capture the case where Player 0 moves to y'_i (for all i), independent of Player 1's behavior. This is the only positional winning strategy from v_1; we call it "autonomous". Note that Player 0's decision where to move from vertex y_1 cannot be revised later: switching from the mimic strategy to the autonomous one (or vice versa) means that Player 0 loses.

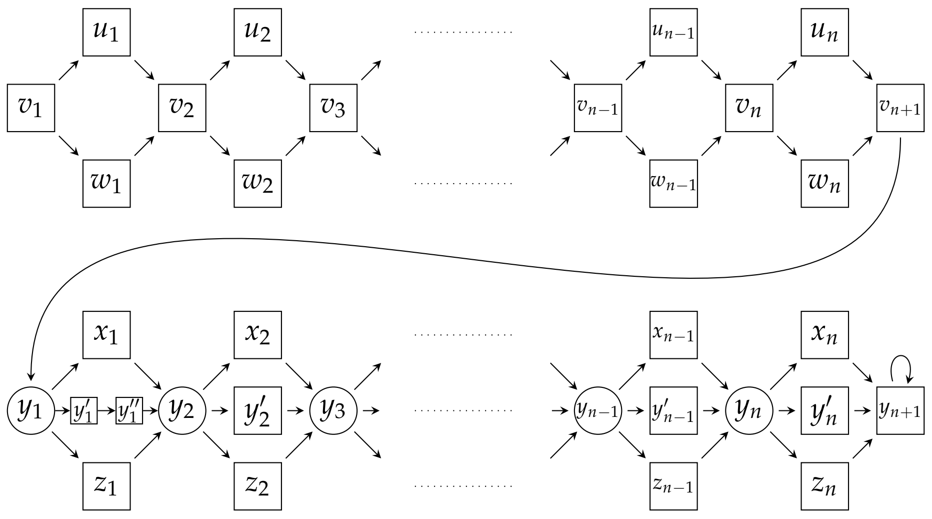

Figure 3.9: *Staiger-Wagner game graph G_n*

Theorem 3.38. *Let $\Gamma_n = (G_n, \mathcal{F}_n)$ be the Staiger-Wagner game from Example 3.37 and let $\Gamma_n' = (G_n', c_n)$ be the expanded weak parity game obtained from Γ_n (according to page 28). Then, Player 0 wins Γ_n from vertex v_1 such that the following hold:*

1. *The positional winning strategy for Player 0 in Γ_n' from $(\varnothing, v_1)$ computed by the algorithm from [Cha06] yields the only positional winning strategy for Player 0 in Γ_n from v_1.*

2. *The reduced game graph G_n'' computed by Algorithm 3.3 has at least 2^n memory contents.*

Proof. To prove item 1 of the theorem it is important to understand that the winning strategy returned by the algorithm from [Cha06] is the autonomous one. The highest possible number of vertices which can be visited in a play (starting at v_1) is $4n+3$. For this, Player 0 has to play the autonomous strategy: it visits two vertices between y_1 and y_2, where the mimic strategy visits only one vertex between y_1 and y_2.

The algorithm from [Cha06] computes the attractor sets A_k, starting with $k = 2 \cdot (7n+3)$[11] and decreasing until $k = 0$ (see proof of Theorem 2.3), where $A_{k'}$ for $k' = 2 \cdot (4n+3)$ is the first non-empty set that is computed. It is not hard to see that each vertex (M, y_1) reachable in G_n' is contained in $A_{k'}$, by considering the autonomous strategy. Since each play (starting at v_1) reaches such a vertex (M, y_1), we also get $(\varnothing, v_1) \in A_{k'}$. No matter which way Player 1

[11]In Γ_n we have $7n+3$ vertices and accordingly the highest possible color in Γ_n' is $2 \cdot (7n+3)$.

chooses to reach vertex y_1 (in G_n), the algorithm always directs Player 0 to play autonomously.

For item 2, consider vertices $(M_1, y_1), (M_2, y_1)$ reachable in G'_n, with $M_1 \neq M_2$. Let $\mathcal{A}$ be the weak parity game automaton of Γ'_n, for arbitrary n. Then there exists $1 \leq j \leq n$ such that w.l.o.g. it holds $u_j \in M_1 \setminus M_2$. Moreover, there exists an input α of $\mathcal{A}$ with x_j appearing in it such that α is accepted from (M_1, y_1) and rejected from (M_2, y_1). This implies that $(M_1, y_1), (M_2, y_1)$ are non-equivalent in $\mathcal{A}$ and, accordingly, it holds $M_1 \not\approx_S M_2$. Since there are at least 2^n candidates for M_1 and M_2, we get that the reduced game graph G''_n computed by Algorithm 3.3 has at least 2^n memory contents. □

3.6.3 Implementation

In this section we describe some aspects relevant for the implementation of our memory reduction algorithm.

We work with the tool *GASt* ("Games, Automata and Strategies").[12] It has been developed since the diploma thesis of Nico Wallmeier in 2002 and is predominantly implemented in *JAVA* [Wal03]. *GASt* provides numerous algorithms for synthesis problems in the field of infinite games (and automata over infinite words), for example solutions to all types of games presented in Chapter 2. Moreover, the tool comprises a technique for the optimization of "waiting times" in winning strategies for Request-Response games (cf. [Wal08]).

GASt contains all the game simulation algorithms presented in Section 2.3. First, a given game graph G is traversed by a depth-first search to construct the reachable part of the expanded game graph G'. Afterwards, the associated game is solved and the positional winning strategies obtained are used to construct winning strategies for the game underlying graph G. We have integrated our memory reduction algorithm as follows: the reduction of the expanded game graph G', i.e., the computation of G'', is invoked directly after the construction of G', *before* the computation of positional winning strategies.

We have implemented our algorithm for four types of winning conditions: Staiger-Wagner, Request-Response, Muller and Streett. Each one of them is encapsulated in its own class, inheriting functionalities from a common superclass. An important difference to the formal presentation of our technique is that the computations are carried out directly on the game graph G'; the notion of a game automaton is just a formality used to prove correctness of our approach.

[12]Until late 2006 the tool was called *SymProg*.

In Sections 3.3 and 3.4 we have shown that for both weak parity game automata and Büchi game automata the problem of state space reduction can be reduced to the minimization problem for standard DFA. Hence, we apply the block partitioning algorithm from [Hop71] in both cases[13], after some individual preprocessing. In the case of a weak parity game (as obtained by game simulation from a Staiger-Wagner game), we compute a maximal coloring on the SCC-graph of G'. The actual translation to a DWA is only implicit, meaning that a set of final states is not explicitly constructed; instead, it is identified with the set of states which have an even color. By the results of Section 3.3.2 only vertices with the same color can be equivalent. Hence, in the quotient game graph that is to be computed we can adopt the color assigned by the maximal coloring. The preprocessing for a Büchi game (as obtained by game simulation from a Request-Response game) is the computation of the closure (cf. Definition 3.22) of the expanded Büchi game graph, manipulating the set of final states.

To implement Algorithm 3.5 for Muller and Streett games, we are faced with the reduction of a parity game graph. Therefor, we have implemented the delayed simulation game for the parity condition. It is constructed as a usual Büchi game, making use of functionalities which have earlier been implemented in *GASt*. The solution to this game is processed to obtain the relation $\approx^{rh}_{de}$.

For each of the considered winning conditions, the computed equivalence relation on $S \times V$ has been proven to be compatible. Thus, in our implementation, we can refine each one to obtain the relation $\approx_S$ on the memory component S. Finally, we use $\approx_S$ to compute the corresponding quotient game graph. To do so, we fix a total order $\prec_S$ on S and retain only the $\prec_S$-minimal element of each $\approx_S$-equivalence class; to simplify matters, the initial memory content s_0 is assumed $\prec_S$-minimal. All other memory contents are erased, and the quotient game graph is then computed in the obvious way.

3.6.3.1 Computation Results

Whereas *GASt* provides synthesis algorithms based on both the enumerative and the symbolic representation of the state space, we have implemented our approach only for the enumerative case. Due to the exponential size of a game graph obtained by game simulation, the running time of our algorithm grows very rapidly. Sometimes considering only the reachable vertices of G'

[13]The restriction that our technique requires a deterministic transition structure is neutralized by the fact that only vertices with the same V-component are tested for equivalence.

already yields a substantial reduction of the needed memory. In that case, we have to restrict the definition of $\approx_S$ further. More on this can be found in Section 6.1 in [Hol07]. However, to obtain the results presented in Section 3.6.1, it is required to consider the non-reachable part of G' as well.

The following tables summarize some computation results for the games from Examples 3.31 and 3.33, with and without our memory reduction algorithm. One can observe the exponential growth rate of both the size of the expanded game graph (see $|S_n|, |V'_n|$ in Table 3.2 and $|S_k|, |V'_k|$ in Table 3.3) and the time needed to solve the respective game. However, the computation of $\approx_S$ and the construction of the quotient game graph is done efficiently, because we measure the running time of our algorithm in the size of the expanded game graph. Moreover, note that in the first case the memory is reduced to size five whereas in the second case we obtain a memory of size one, no matter the value of the parameter (n or k).

n	$\|V_n\|$	Γ'_n			Memory Reduction		Γ''_n		
		$\|S_n\|$	$\|V'_n\|$	Solve	$\approx_S$	Quotient	$\|S'_n\|$	$\|V''_n\|$	Solve
2	7	69	699	68 ms	1.8 s	0.3 s	5	23	$\leq$ 5 ms each
3	9	203	2906	0.31 s	31.5 s	3.7 s	5	29	
4	11	609	11291	3.15 s	684 s	56 s	5	35	

Table 3.2: *Computation results for Example 3.31*

k	$\|V_k\|$	Γ'_k			Memory Reduction		Γ''_k		
		$\|S_k\|$	$\|V'_k\|$	Solve	$\approx_S$	Quotient	$\|S'_k\|$	$\|V''_k\|$	Solve
2	14	30	716	120 ms	2 s	20 ms	1	14	$\leq$ 6 ms each
3	20	81	3743	860 ms	113 s	70 ms	1	20	
4	26	211	16947	36 s	80 mins	0.5 s	1	26	
5	32	560	71596	out of mem	44.5 h	2.6 s	1	32	

Table 3.3: *Computation results for Example 3.33*

Conclusion of Part I

Summary

In the first part of the thesis, we have dealt with the problem of reducing the memory necessary for implementing winning strategies in regular infinite two-player games. Our motivation has been initiated by the fact that, in many cases, winning strategies constructed by standard algorithms require considerably more space to be realized than actually needed. As a remedy, we have introduced a technique which is independent of particular strategies, approaching the problem of memory reduction before the computation of winning strategies.

Our Approach. The key idea of our algorithm is to apply a game simulation extending the given game arena by a finite memory component, and to reduce the extended game graph by identifying equivalent memory contents, afterwards. This has been realized by a transformation of the simulating game into an equivalent deterministic word automaton, called game automaton, which recognizes the language of all plays winning for Player 0 in the given game. We compute an equivalence relation $\approx_S$ on the set S of memory contents, declaring two memory contents equivalent if, from them, Player 0 wins precisely the same plays. The correctness of our technique follows from the fact that quotienting with respect to $\approx_S$ preserves the structural properties of the extended game graph (see Theorem 3.10). Our algorithm has as parameters a game simulation and a language-preserving equivalence relation to reduce the state space of an ω-automaton.

We have applied our approach to distinct classes of winning conditions, among them Muller and Streett, i.e., two standard forms for ω-regular objectives, the practically relevant class of Request-Response conditions, and Staiger-Wagner objectives, capturing the class of weak specifications. Both Muller and Streett games can be simulated by parity games, and Request-Response and Staiger-Wagner games can be simulated by Büchi and weak parity games, respectively. We were thus faced with the reduction of game automata with either a parity or a Büchi or a weak parity acceptance condition. Since the minimization problem for ω-automata is known to be PSPACE-hard already for deterministic Büchi automata, one of the main challenges of our approach was to find efficient techniques for state space reduction of the aforementioned types of game automata.

State Space Reduction of Game Automata. We have shown that state space reduction for weak parity game automata can be reduced to the minimization problem for standard DFA. To do so, we proceeded via a transformation to weak Büchi game automata and used an approach proposed by Löding [Löd01]. Thereby, we obtained a running time of our memory reduction algorithm for Staiger-Wagner games which is polynomial in the size of the simulating weak parity game: if n is the number of states of the given game automaton, then we require time $\mathcal{O}(n \cdot (\log n)^2)$.

For state space reduction of both Büchi game automata and parity game automata, we have relied on heuristics based on the notion of delayed simulation. It has been introduced in [EWS05] by Etessami et al. for Büchi conditions and has been extended by Fritz and Wilke for (alternating) parity acceptance [FW06]. Delayed simulation is characterized by means of an infinite game between two players, called Spoiler and Duplicator. In our setting, a state q delayed simulates a state q' if Duplicator (starting at q) can fulfill all the obligations imposed by Spoiler (starting at q'), in a delayed fashion. In the case of Büchi acceptance an obligation is a visit to a final state, and for a parity condition it means that Spoiler sees a better color than Duplicator.

We have shown that, for Büchi game automata, the computation of delayed simulation can be reduced to minimization of standard DFA. This yields the same asymptotic running time of our technique for Request-Response games as for Staiger-Wagner games.

For the parity condition, computing delayed simulation amounts to solving the corresponding simulation game explicitly. It is a Büchi game on a graph which is equipped with a priority memory; this memory keeps track of pending obligations, and every time Duplicator fulfills one a final vertex is assumed. Solving the delayed simulation game for a parity game automaton requires time $\mathcal{O}(n^2 \cdot k)$, where n is the number of states and k the number of colors. The costs of our algorithm for memory reduction in Muller and Streett games are thus polynomially bounded in the size of the extended parity game.

Results. Our main result is that the approach of reducing the extended game graph, which is obtained by a game simulation, can effect an exponential gain in the size of the memory.

For the class of Staiger-Wagner conditions, we gave an example where a standard algorithm computes a winning strategy of exponential size, but our technique reduces the memory to constant size. For the class of Request-Response specifications, we presented a family of games where our approach optimizes the simulating game graph, i.e., the memory is reduced to size one. Thus, for each player a positional winning strategy can be computed.

In both the proofs of the aforementioned results, we made use of the fact that attractor strategies choose the shortest way to reach a final vertex. Moreover, the algorithm from [Cha06] for solving weak parity games prefers high colors to low ones. Both properties turned out to be disadvantageous for the purpose of finding simple winning strategies, and they are ruled out by our algorithm.

For obtaining a similar result for Streett games, we had to make an assumption of the winning strategy computed for Player 0 in the simulating parity game. More precisely, we postulated that Player 0 behaves optimally, in the sense that she plays a strategy which guarantees to visit the best colors possible. We showed that, under this natural assumption, one obtains a winning strategy for Player 0 of optimal size.

Further Prospects

Delayed Simulation is too Fine. Unfortunately, a positive result for Muller games (with a substantial reduction of the memory) is still missing. In fact, one can show that the only game graph where our algorithm is capable of reducing the memory to size one is the simple cycle (of any length). The major problem is that the definition of delayed simulation for the parity condition has to be very restrictive in order to preserve the recognized language upon quotienting. It is not hard to see that, although Muller conditions are prefix-independent, finite prefixes of a run in a parity (game) automaton must not be neglected when computing the delayed simulation relation. To that effect, it would be interesting to search for alternative definitions.

Alternating Acceptance. Our algorithm lacks in a mechanism for taking into account the behavior of the two players. One improvement in this direction could be to retain the partition of the game graph into Player 0's and Player 1's vertices and to apply an approach for state space reduction of alternating automata (see for example [FW02, FW06]).

Complexity of Memory Optimization. We have not considered finding a lower bound for the complexity of computing winning strategies of optimal size. On the one hand, Hunter and Dawar have shown in [HD05] that the problem of deciding the winner in a Muller game with a winning condition represented by a *win-set* is PSPACE-complete. This immediately implies PSPACE-completeness of memory optimization for win-set Muller games. On the other hand, Horn has proven explicit Muller games, i.e., the kind of representation chosen in this thesis, to be solvable in P [Hor08]. It would be interesting to see whether this result carries over to the problem of finding winning strategies of optimal size. Similar questions can also be asked for Staiger-Wagner games.

Part II

Infinite Games with Finite Delay

Chapter 4

Games with Delay

In this chapter we introduce infinite games with delay. The basic idea underlying those games is to refine the classical way of playing Gale-Stewart games. One of the two players is allowed to defer the delivery of each of his moves for an arbitrary finite number of steps. Note that this induces a generalized notion of a strategy such that one player has the additional move *skip*. The delay at a given point of a play is the difference in the total number of moves made by the players, disregarding each *skip*. We perform an analysis of the decidability of these games, exploring the set of possible winning strategies in the delayed setting.

The concept of strategies which are generalized in the above sense corresponds to the class of continuous operators. Early research on the interrelation between games and operators can be found for example in the work of Büchi and Landweber [BL69], where the authors indicate the connection between a game with delay[1] and the continuity of an operator mapping from one space into another. In fact, Büchi and Landweber raise the fundamental question on the decidability of infinite games in terms of *delay operators*. Their famous result that the winner of a regular game is decidable captures the case of delay zero.

The idea of strategies with delay is also pursued in [HL72] where the authors show that the existence of a bounded-delay operator satisfying a given regular specification is decidable. In Chapter 5, we deal with the problem whether a regular specification can be solved by a continuous operator. We show that this problem is decidable, thereby extending the results of [HL72].

This chapter serves as a preparation for a resumption of the work of Büchi, Hosch and Landweber. In Section 4.1 we present the basic topological notions and discuss several restrictions of continuity; we exhibit a hierarchy which classifies operators according to the delay they require. In Section 4.2 we state the decision problem which we are going to approach for the rest of this work.

[1]Büchi and Landweber use the dual notion, called *shift*.

We reformulate the existence of a continuous operator satisfying a given condition as the existence of a winning strategy in a Gale-Stewart game with finite delay.

4.1 Delay Operators

Let us first give the most basic topological notions needed to introduce delay operators. The definitions presented here are along the lines of [TL93].

For an alphabet Σ, the function $\mathrm{dist} : \Sigma^\omega \times \Sigma^\omega \to [0,1]$ with

$$\mathrm{dist}(\alpha, \beta) := \begin{cases} 0 & \text{, if } \alpha = \beta \text{, and} \\ \frac{1}{2^n} & \text{, if } n \text{ is the minimal } n \text{ with } \alpha[n] \neq \beta[n] \end{cases}$$

defines a distance between ω-sequences. The longer the common prefix of α and β, the closer is the distance between them. The function dist is a *metric*, called *Cantor metric*, inducing a topology on Σ^ω; the pair (Σ^ω, d) is called *Cantor space*. The $\frac{1}{2^n}$*-neighborhood* of α is the set of all β such that $\mathrm{dist}(\alpha, \beta) < \frac{1}{2^n}$, i.e., it holds $\alpha[0,n] = \beta[0,n]$. In other words, β is contained in $\alpha[0,n]\Sigma^\omega$. According to the standard definition in topology, a set $L \subseteq \Sigma^\omega$ is *open* in the Cantor space if it is a union of neighborhoods of ω-words, i.e., it can be written as

$$L = W\Sigma^\omega,$$

for some $W \subseteq \Sigma^*$. The set W contains all the aforementioned prefixes $\alpha[0,n]$, each of which determines one neighborhood.

An *operator* is a function $\lambda : \Sigma^\omega \to \Sigma^\omega$. It is called *continuous* if the preimage of every open set $U \subseteq \Sigma^\omega$ is open, where the preimage of U is the set

$$\lambda^{-1}(U) := \{\alpha \in \Sigma^\omega \mid \lambda(\alpha) \in U\}.$$

For an operator which is continuous in the above sense, consider $\alpha \in \Sigma^\omega$ and an arbitrary finite prefix $\lambda(\alpha)[0,k]$ of $\lambda(\alpha)$. Since the set $\lambda(\alpha)[0,k]\Sigma^\omega$ is open, its preimage $\lambda^{-1}(\lambda(\alpha)[0,k]\Sigma^\omega)$ is open as well, i.e., it has the form $W\Sigma^\omega$ for some $W \subseteq \Sigma^*$. Since $\alpha \in W\Sigma^\omega$, there must be a finite prefix w of α with $w \in W$ and $\alpha \in \{w\}\Sigma^\omega \subseteq W\Sigma^\omega$. Each $\alpha' \in \Sigma^\omega$ with prefix w is also contained in $W\Sigma^\omega$, and therefore has a λ-image contained in $\lambda(\alpha)[0,k]\Sigma^\omega$, therefore it holds $\lambda(\alpha)[0,k] = \lambda(\alpha')[0,k]$. In other words, the input prefix w determines the output prefix $\lambda(\alpha)[0,k]$. Altogether, this means that an operator λ is continuous if and only if for each $\alpha \in \Sigma^\omega$ each finite prefix of $\lambda(\alpha)$ is determined by a finite prefix of α.

For our purposes, the latter characterization of continuity is more convenient. To capture the constraint that each finite prefix of the output is determined by a finite prefix of the input, we use a labeling l of the full $|\Sigma|$-branching tree, mostly referred to as *labeling tree*. For $\Sigma = \{a_1, \dots, a_k\}$, each vertex is assigned a letter in $\{a_1, \dots, a_k\} \uplus \{\triangleright\}$, where $\triangleright$ means that the next output letter is not yet determined. The output $\lambda(\alpha)$ is the sequence of l-values on the path corresponding to α, where all the letters $\triangleright$ are neglected.

Definition 4.1. An operator $\lambda : \Sigma^\omega \to \Sigma^\omega$ is *continuous* if there exists a mapping $l : \Sigma^* \to \Sigma \uplus \{\triangleright\}$ such that for all $\alpha \in \Sigma^\omega$ the ω-word $l(\alpha) := l(\alpha[0])l(\alpha[0,1])l(\alpha[0,2]) \cdots$ satisfies the following conditions:

1. The word $l(\alpha)$ does not end with $\triangleright^\omega$.
2. $\lambda(\alpha) = \mathrm{strip}(l(\alpha))$, where $\mathrm{strip}(l(\alpha))$ is the ω-word $l(\alpha)$ with all letters $\triangleright$ removed.

Note that in the above definition the word $l(\alpha)$ can have $\triangleright$-infixes of arbitrary finite length. However, this does not contradict continuity. As an example, let $\mathbb{B} := \{0,1\}$ and consider the following continuous operator.

Example 4.2. Let $\lambda_1 : \mathbb{B}^\omega \to \mathbb{B}^\omega$ be defined such that $\lambda_1(b_0 b_1 b_2 \cdots) := b'_0 b'_1 b'_2 \cdots$ where for all $i \in \mathbb{N}$

$$b'_i := \Big(\sum_{j=0}^{2^i} b_j \Big) \bmod 2$$

Note that for determining the bit b'_i of the output one has to know the (finite) prefix $b_0 \cdots b_{2^i}$ of the input. The continuity of λ_1 is verified for example by the following labeling $l : \mathbb{B}^* \to \mathbb{B} \uplus \{\triangleright\}$:

$$l(b_0 \cdots b_j) := \begin{cases} \Big(\sum_{k=0}^{j} b_k \Big) \bmod 2 & \text{, if } j = 2^i \text{ for some } i \in \mathbb{N} \\ \triangleright & \text{, otherwise} \end{cases}$$

Given an operator λ and an input α, we say that a prefix u of $\lambda(\alpha)$ *depends only* on a prefix v of α if each α' with the prefix v has a λ-image with prefix u. We use a function for describing an upper bound for the length of input prefixes on which output prefixes of a given length depend. If such a bound exists for each particular length, then we say that λ is uniformly continuous.

Definition 4.3. Let $h : \mathbb{N} \to \mathbb{N}$ be a strictly increasing function. We say that λ is a *h-delay* operator if, for each $\alpha \in \Sigma^\omega$, the prefix $\lambda(\alpha)[0,i]$ depends only on $\alpha[0, h(i)]$.

An operator λ is said to be *uniformly continuous* if there exists a function h of the above form such that λ is a h-delay operator.

A property of the space Σ^ω we intend to make use of is the fact that the set of continuous operators and the set of uniformly continuous operators coincide.

First of all, note that each uniformly continuous operator is indeed continuous. Let $h : \mathbb{N} \to \mathbb{N}$ be strictly increasing and let λ be a h-delay operator. Then, given any $i \in \mathbb{N}$, the value $h(i)$ indicates the maximal level one has descend to in some $|\Sigma|$-branching tree such that on each path at least $i{+}1$ non-$\triangleright$ labels are seen. More precisely, we label all vertices on the levels 0 to $h(0){-}1$ with $\triangleright$. Afterwards, we label each node $\alpha[0,h(0){-}1]$ on level $h(0)$ with the first letter of $\lambda(\alpha[0,h(0){-}1]\beta)$, for arbitrary $\alpha,\beta \in \Sigma^\omega$. Note that this letter is independent of β, as it depends only on $\alpha[0,h(0){-}1]$, by assumption. Then we label all vertices on levels $h(0){+}1$ to $h(1){-}1$ with $\triangleright$ and each node $\alpha[0,h(1){-}1]$ on level $h(1)$ with the second letter of $\lambda(\alpha[0,h(1){-}1]\beta')$, for any $\beta' \in \Sigma^\omega$, and so forth.

Each continuous operator from Σ^ω to Σ^ω is also uniformly continuous. This follows from the fact that the space Σ^ω is compact, i.e., it is closed and bounded. Equivalently, one can show the property by using König's Lemma.

Lemma 4.4. *Let $\lambda : \Sigma^\omega \to \Sigma^\omega$ be continuous. Then, there exists a strictly increasing function $h : \mathbb{N} \to \mathbb{N}$ such that λ is a h-delay operator.*

Proof. Let $l : \Sigma^* \to \Sigma \cup \{\triangleright\}$ be a labeling of the full $|\Sigma|$-branching tree witnessing the continuity of λ. We claim that for each node $v \in \Sigma^*$ there exists a number $k(v) \in \mathbb{N}$ such that for each word $a_1 \cdots a_{k(v)} \in \Sigma^{k(v)}$ at least one node w in the set $\{v, va_1, \ldots, va_1 \cdots v_{k(v)}\}$ has a non-$\triangleright$ label, i.e., it holds $l(w) \neq \triangleright$. Towards a contradiction, assume $u \in \Sigma^*$ violates this condition and consider the tree of all $\triangleright$-labeled nodes rooted at u. Since $k(u)$ does not exist, the tree has nodes on levels of arbitrary high depth and, hence, is infinite. Moreover, every node has at most $|\Sigma|$ successors, i.e., the tree is finitely branching. Thus, by König's Lemma it contains an infinite path. As a consequence, there exists $\beta \in \Sigma^\omega$ such that $l(u\beta)$ ends with $\triangleright^\omega$; a contradiction to Definition 4.1.

Knowing that for each $v \in \Sigma^*$ the number $k(v)$ exists, we define the function $h : \mathbb{N} \to \mathbb{N}$ inductively by $h(0) := k(\varepsilon)$ and

$$h(i+1) := h(i) + 1 + \max_{\{v \mid |v| = h(i)+1\}} k(v).$$

The definition of h guarantees that, for each $i \in \mathbb{N}$, there is a non-$\triangleright$ label on each path between (and including) levels $h(i){+}1$ and $h(i{+}1)$. Thus, on each path of the tree there are at least $i{+}1$ non-$\triangleright$ labels up to (and including) level $h(i)$. This means, if the input prefix $\alpha[0,h(i){-}1]$ is known, then the output prefix $\lambda(\alpha)[0,i]$ is determined. By Definition 4.3, λ is a h-delay operator and hence uniformly continuous. □

A function h witnessing uniform continuity of a given operator exhibits a bound for the number of additional input letters needed to determine the next output letter. As we want to consider strategies inducing only finite delay, the results we are going to show in Chapter 5 rely on the existence of such a bound. Indeed, an operator which is continuous but not uniformly continuous may require more than finite delay. To see this, we consider an example which has the crucial property that the space considered is no longer closed.

Example 4.5. Let the operator $\lambda_2 : \mathbb{B}^\omega \setminus \{0^\omega\} \to \mathbb{B}^\omega$ be defined such that for all $b \in \mathbb{B}, \beta \in \mathbb{B}^\omega$ we have

$$\lambda_2(0^*1b\beta) := b1^\omega.$$

The operator λ_2 checks if there is 0 or 1 directly after the first 1 in the input. Note that λ_2 is continuous, since the first bit of the output is determined by a finite prefix of the input. However, there exists no strictly increasing function $h : \mathbb{N} \to \mathbb{N}$, such that λ_2 is a h-delay operator. This is due to the fact that we cannot bound the length of the input prefix to clarify whether $b = 0$ or $b = 1$. Hence, the operator λ_2 is not uniformly continuous. In other words, if one is asked to give the first bit of the output, then no particular length of prefixes does suffice to give the answer. Rather, the whole input α is needed, i.e., we require infinite delay.

As we have already seen, the classes of continuous operators $\Sigma^\omega \to \Sigma^\omega$ and uniformly continuous operators $\Sigma^\omega \to \Sigma^\omega$ are exactly the same. This holds analogously for any finite alphabet of size at least two. Therefore, we can restrict ourselves to the Boolean alphabet $\mathbb{B} := \{0, 1\}$, from now on.

Among the uniformly continuous operators we distinguish an even more restricted class. Intuitively, encountering a sequence of $\triangleright$-letters in the tree described by the labeling function l (cf. Definition 4.1) means that the next output letter is not yet determined. If from some level onwards letter $\triangleright$ does not appear anymore, then $l(\alpha) \in (\triangleright^*\mathbb{B})^*\mathbb{B}^\omega$, for each $\alpha \in \mathbb{B}^\omega$. In this case, the function h can be defined such that $h(i{+}1) = h(i){+}1$, for all $i \geq i_0$ with i_0 chosen appropriately.

Definition 4.6. Let $h : \mathbb{N} \to \mathbb{N}$ be a strictly increasing function. We say that h is of *bounded delay* if there exists $i_0 \in \mathbb{N}$ such that $h(i{+}1) = h(i){+}1$, for all $i \geq i_0$. We say that $\lambda : \mathbb{B}^\omega \to \mathbb{B}^\omega$ is a *bounded-delay* operator (or an operator of *bounded delay*) if and only if it is a h-delay operator for some function h of bounded delay.

If λ is a bounded-delay operator, then there exists $d \in \mathbb{N}$ with $h(i) \leq i{+}d$ for all $i \in \mathbb{N}$, and $h(i) = i{+}d$ for all $i \geq i_0$. This means that, for each $\alpha \in \mathbb{B}^\omega$, the output prefix $\lambda(\alpha)[0,i]$ depends only on $\alpha[0,i+d]$. The smallest possible value for d is the maximal number of $\triangleright$-letters required on any path in an adequate labeling tree.

Example 4.7. Let $\lambda_3 : \mathbb{B}^\omega \to \mathbb{B}^\omega$ be the following bounded-delay operator:

$$\lambda_3(\alpha) := \begin{cases} (1-\alpha[0])0^\omega & \text{, if } (\alpha[0]+\ldots+\alpha[3]) \bmod 2 = 0 \\ (1-\alpha[0])1^\omega & \text{, otherwise} \end{cases}$$

Note that, for each $\alpha \in \mathbb{B}^\omega$, the first output bit $\lambda_3(\alpha)[0]$ depends only on $\alpha[0]$. However, the second output bit $\lambda_3(\alpha)[1]$ is determined only after the fourth input bit $\alpha[3]$ is known. Consider the possible labeling function l which is given by the tree in Figure 4.1. Each node has two entries: the upper one is the (finite) input prefix w corresponding to that node, and the lower one is the value $l(w)$. All nodes below level four have a non-$\triangleright$ label (and are left out in the figure). Node 0 has label 1, because every α starting with 0 has a λ_3-image starting with 1 (analogously for the node $w = 1$ with label 0). According to the definition of λ_3, a node $w \in \mathbb{B}^4$ has label 0 if w contains an even number of 1s.

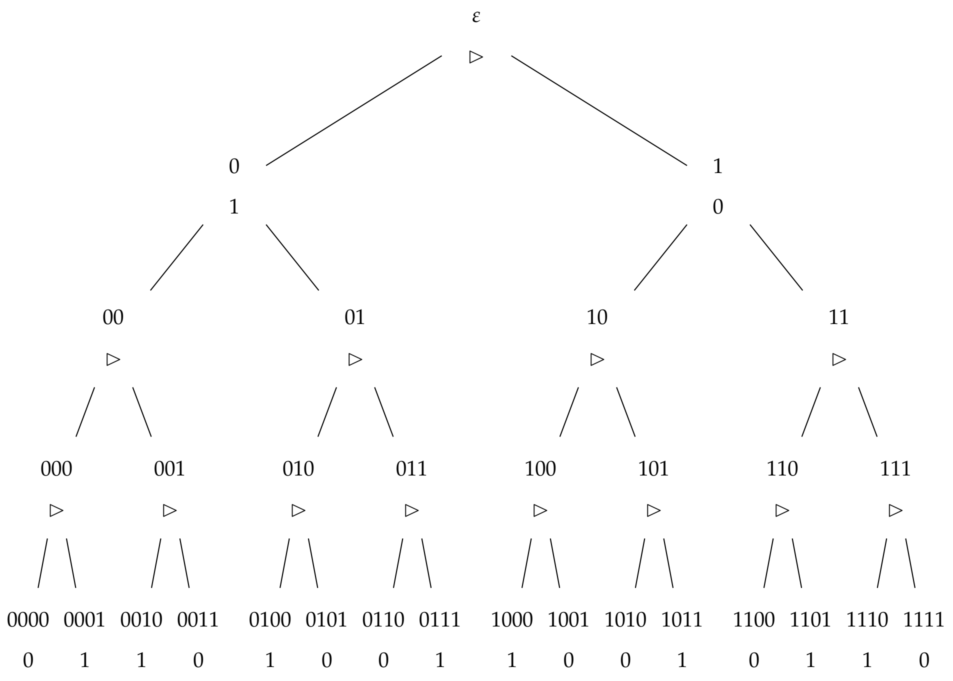

Figure 4.1: *A labeling tree inducing bounded delay*

In topological terms, an operator $\lambda : \mathbb{B}^\omega \to \mathbb{B}^\omega$ is of bounded delay if it is *Lipschitz continuous*, the latter meaning that there exists $c \in \mathbb{R}$ such that for all $\alpha, \beta \in \mathbb{B}^\omega$ it holds $\mathrm{dist}(\lambda(\alpha), \lambda(\beta)) \leq c \cdot \mathrm{dist}(\alpha, \beta)$. If c satisfies the latter condition, then we say that λ is *c-Lipschitz continuous*, and c is the corresponding *Lipschitz constant*. Thus, if λ is a h-delay operator with $h(i) \leq i{+}d$ (for all $i \in \mathbb{N}$), then the Lipschitz constant can be chosen as $c := 2^d$. Conversely, if λ is not of bounded delay, then for each d there exist $\alpha \in \mathbb{B}^\omega, i \in \mathbb{N}$ such that $\lambda(\alpha)[0,i]$ is not determined by $\alpha[0, i{+}d]$. This means that there exists $\beta \in \mathbb{B}^\omega$ with $\beta[0, i{+}d] = \alpha[0, i{+}d]$, i.e., satisfying $\mathrm{dist}(\alpha, \beta) < 2^{-(i+d)}$, such that $\lambda(\beta)[0,i] \neq \lambda(\alpha)[0,i]$, therefore $\mathrm{dist}(\lambda(\alpha), \lambda(\beta)) \geq 2^{-i}$. This contradicts Lipschitz continuity as it holds

$$\frac{\mathrm{dist}(\lambda(\alpha), \lambda(\beta))}{\mathrm{dist}(\alpha, \beta)} \geq \frac{2^{-i}}{2^{-(i+d)}} = 2^d,$$

which cannot be bounded by any $c \in \mathbb{R}$.

A further restriction of bounded-delay operators are constant-delay operators. For such operators it holds that, for each $\alpha \in \mathbb{B}^\omega$, the labeling $l(\alpha)$ is contained in $\triangleright^* \mathbb{B}^\omega$.

Definition 4.8. Let h be of bounded delay, according to Definition 4.6. We say that h is of *constant delay d* if $h(0) = d$ for some $d \in \mathbb{N}$ and $h(i{+}1) = h(i){+}1$ for all $i \in \mathbb{N}$. An operator $\lambda : \mathbb{B}^\omega \to \mathbb{B}^\omega$ is a *d-delay* operator (or an operator of *constant delay d*) if and only if it is a h-delay operator for some function h of constant delay d.

By definition, each d-delay operator is a bounded-delay operator. However, the converse also holds, if the constant d is chosen appropriately. Let λ be a h-delay operator for some function h of bounded delay, with i_0 given as in Definition 4.6. Then, one can define a new function h' such that $h'(0) := h(i_0)$ and $h'(i{+}1) := h'(i){+}1$ for $i \in \mathbb{N}$. Note that λ is an operator of constant delay $h'(0)$. In Example 4.7 we get $h'(0) = 4$, because below (and including) that level no further $\triangleright$-letters occur in the labeling tree.

In this section we have exhibited a hierarchy of delay operators, i.e., continuous functions from one space into another. For a finite alphabet Σ, the classes of continuous operators from Σ^ω to Σ^ω and uniformly continuous operators from Σ^ω to Σ^ω coincide (indicated by the dashed line in Figure 4.2). Moreover, we have seen that each operator of bounded delay is also one of constant delay.

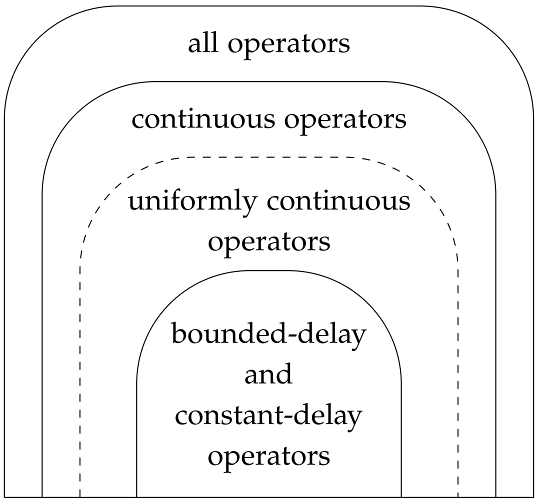
all operators
continuous operators
uniformly continuous operators
bounded-delay and constant-delay operators

Figure 4.2: *A hierarchy of operators*

4.2 Decision Problem

In this section we bridge between the existence of an operator λ satisfying a given condition between input and output and the existence of a winning strategy for a particular player in an infinite game. The first player builds up the input sequence $\alpha \in \mathbb{B}^\omega$ and is hence called Player I; I stands for Input. The second player builds up the output sequence $\beta := \lambda(\alpha) \in \mathbb{B}^\omega$ and, accordingly, is called Player O. In the framework of infinite games, continuity means that Player O may postpone a choice for a finite number of moves of the opponent, namely until she has enough information to give the next output bit.

The core matter of our analysis concerns two things: first of all, we ask whether the existence of a continuous operator guaranteeing particular properties is decidable and, second, if such an operator exists then we want to synthesize one which is as simple as possible; this is meant in the sense that it is located on a low level in the hierarchy of Figure 4.2. More precisely, we approach the following two problems.

Problem 4.9. Let L be an ω-language over $\mathbb{B}^2$. Does there exist a continuous operator $\lambda : \mathbb{B}^\omega \to \mathbb{B}^\omega$ such that for all $\alpha \in \mathbb{B}^\omega$ it holds $\binom{\alpha}{\lambda(\alpha)} \in L$?

Problem 4.10. Let L be an ω-language over $\mathbb{B}^2$. Does there exist a bounded-delay operator $\lambda : \mathbb{B}^\omega \to \mathbb{B}^\omega$ such that for all $\alpha \in \mathbb{B}^\omega$ it holds $\binom{\alpha}{\lambda(\alpha)} \in L$?

Our main goal is to prove that, for regular L, Problem 4.9 is decidable. To do so, we first reformulate it in the framework of infinite games and explain why there exists a continuous operator solving L if and only if Player O has a winning strategy in a generalized Gale-Stewart game.

Decidability of Problem 4.10 has already been shown in [HL72]. However, we improve the results obtained there by establishing a connection to operators of unbounded, i.e., finite but not bounded, delay. Our techniques show that if Player O has a winning strategy inducing a continuous operator, then she also has one requiring only constant delay. This means that, for regular L, the answer to the question in Problem 4.10 is the same as the answer to the question in Problem 4.9. Moreover, our proofs reveal a better upper bound for the required constant than given in [HL72].

In Section 4.1, we have assumed that the function $h : \mathbb{N} \to \mathbb{N}$ is strictly increasing. For our purposes, it is more convenient to consider the function h^Δ, denoting the number of additional input bits until the next output bit:

$$h^\Delta(i) := \begin{cases} h(i) + 1 & \text{, if } i = 0 \\ h(i) - h(i-1) & \text{, if } i > 0 \end{cases}$$

For strictly increasing $h : \mathbb{N} \to \mathbb{N}$ we obtain a unique $h^\Delta : \mathbb{N} \to \mathbb{N}_+$, and vice versa. Accordingly, we say that h^Δ is of bounded delay if h is of bounded delay, and analogously for constant delay. If h is of bounded delay, then there exists $i_0 \in \mathbb{N}$ such that $h^\Delta(i) = 1$, for all $i > i_0$. Moreover, if h is of constant delay then it holds $i_0 = 0$ (cf. Definitions 4.6 and 4.8). For technical convenience, from now on we work only with the functions h^Δ, calling them *delay functions*. However, to simplify matters, we usually write f (or g) instead of h^Δ. Moreover, we use a special notation for functions inducing constant delay. We write $\langle d \rangle$ for the delay function $f : \mathbb{N} \to \mathbb{N}_+$ of constant delay d: $\langle d \rangle(0) = d+1$ and $\langle d \rangle(i) = 1$ for $i \geq 1$.

Let us now introduce the *delay game* $\Gamma_f(L)$; it is a generalization of a Gale-Stewart game and has as parameters an ω-language L over $\mathbb{B}^2$ and a delay function $f : \mathbb{N} \to \mathbb{N}_+$. In each round, each of the players makes one move, where Player I begins. The function f imposes a delay on the moves of Player O. This means that in round i Player I has to choose $f(i)$ bits, and Player O chooses one bit, afterwards. This way the players build up two infinite sequences; Player I builds up $\alpha := a_0 a_1 a_2 \cdots$ and Player O builds up $\beta := b_0 b_1 b_2 \cdots$ ($a_i, b_i \in \mathbb{B}$). The corresponding play is winning for Player O if the word

$$\alpha^\wedge\beta := \binom{a_0}{b_0}\binom{a_1}{b_1}\binom{a_2}{b_2}\cdots$$

is contained in L; otherwise, it is winning for Player I.

Observe that the possible strategies for Player O in $\Gamma_f(L)$ correspond precisely to h-delay operators, for $f = h^\Delta$. This is due to the fact that Player O must output her i-th bit after receiving the next $f(i)$ bits chosen by Player I. Thus, the question whether there exists a h-delay operator λ such that the inclusion $\{\binom{\alpha}{\lambda(\alpha)} \mid \alpha \in \mathbb{B}^\omega\} \subseteq L$ holds is equivalent to the question whether there exists a winning strategy for Player O in $\Gamma_f(L)$. We say that L is *solvable with finite delay* (or has a *finite-delay solution*) if there exists $f : \mathbb{N} \to \mathbb{N}_+$ such that Player O wins $\Gamma_f(L)$, and accordingly for functions of bounded and constant delay.

By the explanations above, we can reformulate Problem 4.9 as follows.

Problem 4.11 (FiniteDelay). Let L be an ω-language over $\mathbb{B}^2$. Is L solvable with finite delay?

In Chapter 5, we approach Problem 4.11 and show that it is decidable for the class of regular ω-languages. Moreover, we prove that every finite-delay solution can be reduced to one of constant delay, where the constant is at most doubly exponential in the size of a parity automaton recognizing the specification L.

Theorem 4.12. *Let $\mathcal{A}$ be a DPA over $\mathbb{B}^2$ recognizing the regular ω-language L. Then the following hold:*

1. *The problem whether L is solvable with finite delay is in* 2ExpTime.
2. *If L is solvable with finite delay, then it is solvable with constant delay d, for some d that is doubly exponential in $|\mathcal{A}|$.*

The proofs we are going to present make frequent use of convenient properties of the delay game. First of all, for each regular language $L \subseteq (\mathbb{B}^2)^\omega$ and each delay function $f : \mathbb{N} \to \mathbb{N}_+$, the delay game $\Gamma_f(L)$ is determined. This is due to the fact that it can be modeled by a parity game on a finite or countably infinite graph with finitely many colors (cf. Section 2.1.4). The only difference to the Gale-Stewart game without delay (see Section 1.4.3) is that we have to keep track of a finite word Player I moves ahead. Moreover, if f is of bounded delay, then the parity game graph can be assumed to be finite, because the number of different words of bounded length is finite. That means, we can decide the winner of the corresponding delay game by standard techniques (see for example [GTW02]).

Another property we need is that winning is monotone with respect to an increase or decrease of the delay. For two functions $f, g : \mathbb{N} \to \mathbb{N}_+$, we write $f \sqsubseteq g$ if $f(i) \leq g(i)$ for all $i \in \mathbb{N}$. The relation $\sqsubseteq$ is a partial order on the set of all delay functions. If $f \sqsubseteq g$ and Player O wins Γ_f, then she also wins Γ_g, because in Γ_g she has at least as much information about Player I's moves as in Γ_f, at any given point in a play. (Note that monotonicity also holds if L is non-regular.) Our observations can be subsumed as follows.

Remark 4.13. Let $L \subseteq (\mathbb{B}^2)^\omega$ be an ω-language and $f_0 : \mathbb{N} \to \mathbb{N}_+$. Then the following hold:

1. If L is regular, then the delay game $\Gamma_{f_0}(L)$ is determined.
2. If L is regular and f_0 is of bounded delay, then the winner of $\Gamma_{f_0}(L)$ is decidable.
3. If Player O wins $\Gamma_{f_0}(L)$ then she also wins $\Gamma_f(L)$, for all $f \sqsupseteq f_0$. Analogously, if Player I wins $\Gamma_{f_0}(L)$ then he also wins $\Gamma_f(L)$, for all $f \sqsubseteq f_0$.

In Chapter 6 we extend our game model to a concurrent setting, such that both players may postpone their moves for a finite number of steps. This yields a more general class of games where determinacy is no longer guaranteed. However, we prove that for the case of regular specifications we can decide whether a finite-delay solution exists, by a reduction to the techniques from Chapter 5.

Chapter 5

Finite Delay in Regular Games

The motivation of this chapter mainly comes from the work [HL72] of Hosch and Landweber. There, the authors show that the problem whether a regular specification admits a bounded-delay solution is decidable. We extend this result to arbitrary finite delay, by approaching Problem 4.11 for the class of regular ω-languages. Moreover, we prove that every regular specification allowing for a finite-delay solution also has a constant-delay solution.

Our proof proceeds via a two-stage reduction. The first step is to introduce the *block game* (see Section 5.1), in which Player I has more freedom in the choices he can make. More precisely, the length of the word chosen by him in round i needs not exactly be $f(i)$, as was required in Γ_f. Rather, it has to be of a length contained in a certain interval. We show that this change in the rules of the game does not make a difference for the existence of finite-delay solutions.

The second step of the reduction is done in Section 5.2 where we introduce the *semigroup game*. It is uniquely determined by the semigroup underlying the parity automaton $\mathcal{A}$ recognizing the winning condition L; in particular, this game is independent of any delay function. We show that a winning strategy for Player I in the semigroup game can be used to simulate a winning strategy for him in every block game, and vice versa. Thereby, we reduce the question whether L is solvable with finite delay to the question whether Player O has a winning strategy in the semigroup game. The latter can be answered by applying standard techniques, because the semigroup game is a parity game on a finite graph; more precisely, the number of vertices is doubly exponential in the size of $\mathcal{A}$, i.e., the number of states and the number of colors.

In Sections 5.1 and 5.2, we have the convention that the function f imposes a delay on the moves of Player O. In Section 5.3, we consider the symmetric setting where the player moving ahead is either Player I or Player O. We show that our results are independent of the fact which player defers his moves, by reducing the symmetric setting to the one-sided case. From this we deduce that, for each regular language L (over $\mathbb{B}^2$), either one of the players wins independent of the fact who moves ahead, or there exists a constant d_L (uniquely determined by L) such that one of the players wins with each delay at most d_L-1 and the other player wins with each delay at least d_L. The constant d_L is called the *delay value* of L. From our results, we derive a generalized determinacy of regular Gale-Stewart games (see Theorem 5.23).

Before we start with the technical part, we agree on the fact that $L \subseteq (\mathbb{B}^2)^\omega$ is a regular ω-language, given by a DPA $\mathcal{A}$ over $\mathbb{B}^2$. The size of $\mathcal{A}$, denoted $|\mathcal{A}|$, is given by the number n of states and the number m of colors. Henceforth, we assume L implicitly and mostly leave it out in our notation.

One of the core parts of the overall proof will be to define an equivalence relation on the set of all finite words over $\mathbb{B}^2$, such that two words are equivalent if they induce the same behavior on the given DPA $\mathcal{A}$. (The precise definition of "behavior" is given in Section 5.2.1.) The intuition behind the *block game* is to guarantee that any given equivalence class of the aforementioned relation contains a word which can be chosen as a whole, i.e., in only one move. To ensure this, we need to allow Player I to select words of variable length. This is realized in the next section.

5.1 The Block Game

In this section we accomplish the first step of our main reduction. We relax the number of bits Player I can choose in each move. For this we introduce the *block game* Γ'_f, which differs from Γ_f in two ways: first, the lengths of the words to be chosen by the players are decided by Player I, within certain intervals determined by f; second, Player I is one move ahead compared to Γ_f.

A play in Γ'_f is built up as follows: in his first move, Player I chooses $u_0 \in \mathbb{B}^{[f(0),2f(0)]}$ and $u_1 \in \mathbb{B}^{[f(1),2f(1)]}$, and then Player O chooses $v_0 \in \mathbb{B}^{|u_0|}$. In each round thereafter, i.e., for $i \geq 2$, Player I chooses $u_i \in \mathbb{B}^{[f(i),2f(i)]}$ and Player O responds by a word $v_{i-1} \in \mathbb{B}^{|u_{i-1}|}$. The winning condition is defined as for the "normal" delay game Γ_f, i.e., it is given by the language L recognized by $\mathcal{A}$.

Note that the block game Γ'_f can be modeled by a parity game on a finite or countably infinite graph with finitely many colors. In the vertices we keep track of at most two finite words Player I moves ahead. Thus, by an argument analogous to that for the normal delay game, the block game is determined.

Remark 5.1. Let $f : \mathbb{N} \to \mathbb{N}_+$. The block game Γ'_f is determined.

The first step of our reduction is to show that the existence of a finite-delay solution is not influenced when switching over from the delay game to the block game. More precisely, we prove the following equivalence $(*)$:

$$\begin{gathered}\text{For all functions } f\text{, Player I wins the delay game } \Gamma_f.\\ \Longleftrightarrow \\ \text{For all functions } f\text{, Player I wins the block game } \Gamma'_f.\end{gathered}$$

Note that, by definition, the left side of the above equivalence holds if and only if L is *not* solvable with finite delay. To accomplish the implication from left to right, let us show that Player I wins the block game induced by f, if he wins the delay game induced by f', where for given $f : \mathbb{N} \to \mathbb{N}_+$ the function f' is defined by

$$f'(0) := f(0) + f(1) \text{ and } f'(i) := f(i+1) \text{ for } i > 0.$$

The intuition behind the definition of $f'(0)$ is that, in the block game, Player I has to choose two words of total length at least $f(0) + f(1)$ in the first round.

Proposition 5.2. *Let $f : \mathbb{N} \to \mathbb{N}_+$. If Player I wins $\Gamma_{f'}$ then he also wins Γ'_f.*

Proof. Assume Player I has a winning strategy in $\Gamma_{f'}$. For $i \in \mathbb{N}$, let u_i be the words chosen by Player I in $\Gamma_{f'}$ and u'_i the words chosen by Player I in Γ'_f (and analogously v_i, v'_i for Player O). The winning strategy yields $u_0 \in \mathbb{B}^{f'(0)}$ as Player I's first move. Since $f(0) + f(1) = f'(0)$ we can choose $u'_0 u'_1 = u_0$ as Player I's first move in Γ'_f. Player O answers by $v'_0 \in \mathbb{B}^{|u'_0|}$. We can use v'_0 in $\Gamma_{f'}$ to simulate the moves $v_0, \dots, v_{|v'_0|-1}$ of Player O, each of which consists of one bit. Player I answers by $u_1, \dots, u_{|v'_0|}$ of lengths $f'(1), \dots, f'(|v'_0|)$. Since $|v'_0| \geq 1$, the sum $f'(1) + \cdots + f'(|v'_0|)$ is non-empty and at least $f'(1) = f(2)$. Accordingly, the word $u_1 \cdots u_{|v'_0|}$ is long enough to give u'_2 with $f(2) \leq |u'_2| \leq 2f(2)$. We choose u'_2 as the prefix of $u_1 \cdots u_{|v'_0|}$ of length $f(2)$. Player O answers in Γ'_f by v'_1 of length $|u'_1|$, and we can use it to simulate another $|v'_1|$ rounds in $\Gamma_{f'}$. Thereby, we obtain enough bits to give u'_3, and so on. This way, we build up the same ω-words in $\Gamma_{f'}$ and Γ'_f. Since Player I wins $\Gamma_{f'}$ he wins Γ'_f as well. ☐

For the implication from right to left of the equivalence $(*)$ we show that Player I wins the delay game induced by f, if he wins the block game induced by f'' (see below). To accomplish the proof, we need to define f'' such that it is growing much faster than f. The reason for this is as follows: if we want to simulate Player O's i-th move v'_i in the block game, then we need to guarantee that Player I's $(i+1)$-th move is long enough to be able to play at least $|v'_i|$ rounds in the normal delay game, because in each of these rounds Player O plays only one bit, which can be used for v'_i.

For $f : \mathbb{N} \to \mathbb{N}_+$, let f'' be inductively defined by $f''(0) := f(0)$ and

$$f''(i+1) := \sum_{j=0}^{2(f''(0)+\ldots+f''(i))} f(j).$$

The greatest value for the index j in the above sum is $2(f''(0) + \ldots + f''(i))$, which has a value at least $2(i+1)$. Hence, $f''(i+1)$ has $f(i+1)$ as summand. This yields $f(i) \leq f''(i)$, for all $i \in \mathbb{N}$, therefore $f \sqsubseteq f''$.

Proposition 5.3. *Let $f : \mathbb{N} \to \mathbb{N}_+$. If Player I wins $\Gamma'_{f''}$ then he also wins Γ_f.*

Proof. Assume Player I has a winning strategy in $\Gamma'_{f''}$. For $i \in \mathbb{N}$, let u'_i be the words chosen by Player I in $\Gamma'_{f''}$ and u_i the words chosen by Player I in Γ_f (and analogously v'_i, v_i for Player O). Player I's winning strategy yields $u'_0 \in \mathbb{B}^{[f''(0),2f''(0)]}$ and $u'_1 \in \mathbb{B}^{[f''(1),2f''(1)]}$ as his first move in $\Gamma'_{f''}$. For $i \in \mathbb{N}$, let d'_i be the length of u'_i. Since

$$d'_0 + d'_1 \geq f''(0) + f''(1) = f(0) + \sum_{j=0}^{2f''(0)} f(j),$$

we can give the moves $u_0, \ldots, u_{d'_0}$ of Player I in Γ_f. This yields Player O's answers $v_0, \ldots, v_{d'_0-1}$, i.e., d'_0 bits. We can use them to simulate v'_0, i.e., Player O's first move in $\Gamma'_{f''}$. Player I's winning strategy yields u'_2 of length $f''(2) \leq d'_2 \leq 2f''(2)$. We need to give another d'_1 moves of Player I in Γ_f to obtain Player O's answers $v_{d'_0}, \ldots, v_{d'_0+d'_1-1}$. For that we need $f(d'_0+1) + \ldots + f(d'_0+d'_1)$ bits. With u'_2 in our hands we can give these moves, because

$$\begin{array}{rcrl} d'_2 & \geq & f''(2) & = \; f(0) + \ldots + f(2f''(0) + 2f''(1)) \\ & & & \geq \; f(0) + \ldots + f(d'_0 + d'_1) \\ & & & \geq \; f(d'_0+1) + \ldots + f(d'_0 + d'_1). \end{array}$$

Iterating this, we obtain the same ω-words built up in $\Gamma'_{f''}$ and Γ_f. Since Player I wins $\Gamma'_{f''}$ he also wins Γ_f. ☐

The following corollary of Propositions 5.2 and 5.3 completes the first step in our main reduction: for the implication from item 1 to item 2, assume that Player I wins the delay game Γ_f, for all $f : \mathbb{N} \to \mathbb{N}_+$, and let $f_0 : \mathbb{N} \to \mathbb{N}_+$ be any delay function. Since Player I wins Γ_f for every f, he particularly wins $\Gamma_{f'_0}$. Hence, by Proposition 5.2, he also wins the block game Γ'_{f_0}. The reverse implication is shown analogously, using Proposition 5.3.

Corollary 5.4. *Let L be a regular language over $\mathbb{B}^2$. Then the following are equivalent:*

1. *For all $f : \mathbb{N} \to \mathbb{N}_+$, Player I wins the delay game $\Gamma_f(L)$.*
2. *For all $f : \mathbb{N} \to \mathbb{N}_+$, Player I wins the block game $\Gamma'_f(L)$.*

5.2 The Semigroup Game

In this section we accomplish the second step of our reduction, i.e., the main step for the proof of Theorem 4.12. We introduce the *semigroup game*, which only depends on the given parity automaton $\mathcal{A}$, but which is independent of any particular delay function. It is called semigroup game because a move of a player is an element of the semigroup describing the underlying behavior of $\mathcal{A}$ (see for example [PP95]).

We extract from $\mathcal{A}$ two equivalence relations, one for each player, such that a move of a player is an equivalence class of his or her respective relation. Roughly speaking, an equivalence class contains all words which effect the same behavior on $\mathcal{A}$. The first relation (for Player O) is denoted $\sim_{\mathrm{O}}$ and induces a finite semigroup on $(\mathbb{B}^2)^*$. The second relation (for Player I) ranges over $\mathbb{B}^*$; it is denoted $\sim_{\mathrm{I}}$ and is a refinement of $\sim_{\mathrm{O}}$. We show that each of the two relations has finite index. This is due to the fact that the semigroup underlying $\mathcal{A}$ has only finitely many elements – a major ingredient for obtaining our main decidability result. Moreover, we show that each equivalence class of both the two relations is a regular $*$-language computable from $\mathcal{A}$. As a consequence, we can compute both $\sim_{\mathrm{O}}$ and $\sim_{\mathrm{I}}$.

The semigroup game is a parity game on a graph of size at most doubly exponential in the number n of states and the number m of colors of $\mathcal{A}$; hence, its winner is computable. The main result of this section is that Player I wins the semigroup game if and only if he wins the block game Γ'_f for all delay functions f. From the proof of the latter equivalence we obtain the reduction of finite delay to constant delay as follows: having computed both equivalence relations, we can bound the length of a shortest representative in each class by a doubly exponential constant n' depending only on n and m. Finally, we obtain the result that, if Player O wins the semigroup game, then L is solvable with constant delay $2n'-1$.

5.2.1 From Parity Automata to Semigroups

Our approach to transform parity automata into finite semigroups is similar to the constructions presented in [PP95, Pin95]. Let $\mathcal{A} = (Q, q_0, \delta, c)$ be a DPA over $\mathbb{B}^2$. We use the semiring $\mathcal{S} := (\{\perp\} \cup c(Q), +, \cdot)$ in which addition is defined as maximum, i.e., $x + y := \max\{x, y\}$ with $\perp$ being the least element, and multiplication is defined as follows:

$$x \cdot y := \begin{cases} \max\{x, y\} & \text{, if } x \neq \perp \text{ and } y \neq \perp \\ \perp & \text{, otherwise} \end{cases}$$

Note that the set $L_{\text{eq}} := (\mathbb{B}^2)^*$, i.e., the set of pairs of words of equal length, is a regular language. With each pair $\binom{u}{v} \in L_{\text{eq}}$ we associate a matrix $\mu\binom{u}{v}$ of size $|Q|^2$ with entries in $\mathcal{S}$.

Definition 5.5. Let $\mathcal{A} = (Q, q_0, \delta, c)$ be a DPA over $\mathbb{B}^2$ and $\mu : L_{\text{eq}} \to \mathcal{S}^{Q\times Q}$ the matrix defined as follows:

$$\mu\binom{u}{v}_{p,q} := \begin{cases} \bot & \text{, if } \delta^*\big(p, \binom{u}{v}\big) \neq q \\ \max\{c(\rho)\} & \text{, if } \delta^*\big(p, \binom{u}{v}\big) = q \text{ and } \rho \text{ is the associated } \mathcal{A}\text{-path} \end{cases}$$

We say that $\mu\binom{u}{v}$ is the *behavior* of $\mathcal{A}$ on $\binom{u}{v}$.

Example 5.6. Consider the parity automaton $\mathcal{A}$ depicted in Figure 5.1. The coloring is given in the lower part of each state, and q_0 is the initial state. The symbols $*$ in the transition labels stand for any bit.

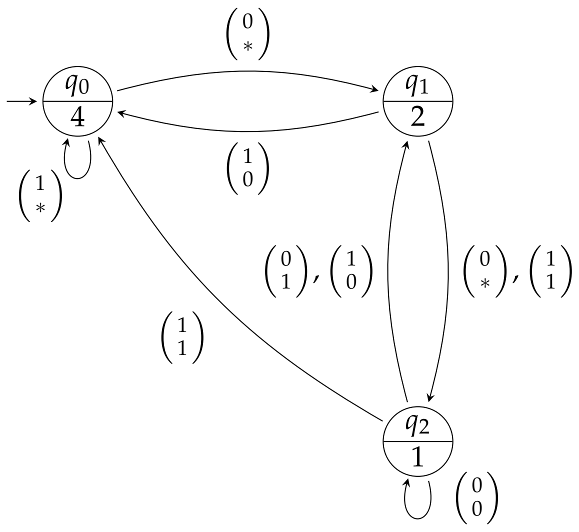

Figure 5.1: *Parity automaton $\mathcal{A}$*

Let the word $\binom{u}{v} = \binom{01}{00} \in L_{\text{eq}}$ be given. (We associate both the first row and the first column of $\mu\binom{01}{00}$ to q_0, and analogously with the other rows and columns for states q_1, q_2.) Analyzing the run of $\mathcal{A}$ on $\binom{01}{00}$ from each particular state, we obtain the behavior

$$\mu\binom{01}{00} = \begin{pmatrix} 4 & \bot & \bot \\ \bot & 2 & \bot \\ \bot & 2 & \bot \end{pmatrix}.$$

For example, from state q_0 the automaton $\mathcal{A}$ changes state to q_1 when reading $\binom{0}{0}$ and returns to q_0 when reading $\binom{1}{0}$. The maximal color seen in this run is 4, which is the reason why we put it in the upper left corner of $\mu\binom{01}{00}$. The reader may verify the other entries of the matrix, analogously.

Note that the set $\mathcal{S}^{Q\times Q}$ of all possible matrices induces a finite semigroup with neutral element $\mu\binom{\varepsilon}{\varepsilon}$ and associative multiplication: for all finite words $u, u', u'', v, v', v'' \in \mathbb{B}^*$ with $|u| = |v|, |u'| = |v'|$ and $|u''| = |v''|$ it holds

$$\mu\binom{u}{v} \cdot \mu\binom{u'u''}{v'v''} = \mu\binom{uu'}{vv'} \cdot \mu\binom{u''}{v''}.$$

We declare two elements in L_{eq} to be $\sim_{\text{O}}$-equivalent if they induce the same behavior on $\mathcal{A}$.

Definition 5.7. Let $\binom{u}{v}, \binom{u'}{v'}$ be two words in $(\mathbb{B}^2)^*$. We define $\binom{u}{v} \sim_{\text{O}} \binom{u'}{v'}$ if it holds $\mu\binom{u}{v} = \mu\binom{u'}{v'}$.

Note that $\binom{0011}{0110}$ has the same behavior as $\binom{01}{00}$, and hence $\binom{0011}{0110} \sim_{\text{O}} \binom{01}{00}$. In the semigroup game we are going to define later on, each move of Player O is a behavior of $\mathcal{A}$, i.e., an equivalence class of $\sim_{\text{O}}$. Obviously, the relation $\sim_{\text{O}}$ is an equivalence relation. For each $\binom{u}{v}$, the equivalence class $\left[\binom{u}{v}\right]$ is identified by the matrix $\mu\binom{u}{v} \in \mathcal{S}^{Q\times Q}$. Since $\mathcal{S}$ and Q are finite, $\mathcal{S}^{Q\times Q}$ is finite as well, and so the relation $\sim_{\text{O}}$ has finite index, i.e., it has finitely many equivalence classes. We denote the index of $\sim_{\text{O}}$ by $\text{index}(\sim_{\text{O}})$. Note that $L_{\text{eq}}/_{\sim_{\text{O}}}$ induces a finite semigroup with neutral element $\left[\binom{\varepsilon}{\varepsilon}\right]$, and μ is a semigroup morphism from $(L_{\text{eq}}/_{\sim_{\text{O}}}, \cdot)$ to $(\mathcal{S}^{Q\times Q}, \cdot)$. We show that we can compute the equivalence relation $\sim_{\text{O}}$ from $\mathcal{A}$.

Lemma 5.8. *Let $\binom{u}{v} \in L_{\text{eq}}$. Then, the set $\left[\binom{u}{v}\right]$ is a regular $*$-language over $\mathbb{B}^2$.*

Proof. We construct an automaton $\mathcal{A}_{[\binom{u}{v}]}$ recognizing $\left[\binom{u}{v}\right]$ as follows: first, we construct for all $p, q \in Q, k \in c(Q)$ the automaton $\mathcal{A}_{p,q,k}$ recognizing the language of all (finite) words inducing a path from p to q in $\mathcal{A}$ where k is the highest color seen on that path. The idea for this construction is to simulate the behavior of $\mathcal{A}$ while memorizing the highest color seen. To this end, define $\mathcal{A}_{p,q,k} := (c(Q) \times Q, (c(p), p), \delta', \{(k, q)\})$ over $\mathbb{B}^2$ where

$$\delta'\left((k', p'), \binom{x}{y}\right) := \left(\max\left\{k', c\left(\delta\left(p', \binom{x}{y}\right)\right)\right\}, \delta\left(p', \binom{x}{y}\right)\right)$$

for all $k' \in c(Q), p' \in Q, x, y \in \mathbb{B}$. The automaton starts in state $(c(p), p)$ and simulates the behavior of $\mathcal{A}$ on its input. If it stops in state (k, q), then it accepts. The automaton $\mathcal{A}_{[\binom{u}{v}]}$ is then obtained as the intersection automaton of all $\mathcal{A}_{p,q,k}$ for p, q, k such that $\mu\binom{u}{v}_{p,q} = k$. $\square$

Since $\sim_{\mathrm{O}}$ has finite index, we can find automata for all its equivalence classes in the following way: for $r \in \mathbb{N}$, let $\mathcal{A}_1, \dots, \mathcal{A}_r$ be the automata already constructed. Then, it holds

$$\mathrm{index}(\sim_{\mathrm{O}}) = r \iff \bigcup_{i=1,\dots,r} L(\mathcal{A}_i) = L_{\mathrm{eq}}.$$

The equality on the right side of the above equivalence can be effectively checked, and if this test fails, then we repeat the construction from the above proof with a word contained in the language $L_{\mathrm{eq}} \setminus \bigcup_{i=1,\dots,r} L(\mathcal{A}_i)$.

Corollary 5.9. *Let $\mathcal{A}$ be a DPA with n states and m colors. Then, the relation $\sim_{\mathrm{O}}$ is computable in time $\mathcal{O}((mn)^{2n})$.*

Proof. Let $u, v \in \mathbb{B}^*$ with $|u| = |v|$. Since $\mathcal{A}$ is deterministic, there is exactly one entry distinct from $\perp$ in each of the n rows of $\mu\binom{u}{v}$, and each automaton $\mathcal{A}_{p,q,k}$ with $\mu\binom{u}{v}_{p,q} = k \neq \perp$ has at most mn states and $4(mn)$ transitions. Hence, the automaton $\mathcal{A}_{[\binom{u}{v}]}$ can be constructed in time $\mathcal{O}((mn)^n)$ and has at most $(mn)^n$ states. There are at most $(mn)^n$ possible matrices identifying all the $\sim_{\mathrm{O}}$-equivalence classes. Accordingly, the time needed to compute $\sim_{\mathrm{O}}$ is in $\mathcal{O}((mn)^n \cdot (mn)^n)$. □

The relation $\sim_{\mathrm{I}}$ is a refinement of $\sim_{\mathrm{O}}$, based on the following idea. Let $u, u' \in \mathbb{B}^*$ be two words which can be chosen by Player I in the block game. We declare them equivalent if for each word $v \in \mathbb{B}^*$ there exists a word $v' \in \mathbb{B}^*$ such that $\binom{u}{v}$ and $\binom{u'}{v'}$ induce the same behavior on $\mathcal{A}$. That means, if u, u' are equivalent then the different behaviors Player O can effect are precisely the same, no matter whether Player I chooses u or u'.

Definition 5.10. Let u, u' be two finite words over $\mathbb{B}$. We define $u \sim_{\mathrm{I}} u'$ if it holds

$$\forall \Big[\binom{u_0}{v_0}\Big] : \Big(\exists v : \binom{u}{v} \in \Big[\binom{u_0}{v_0}\Big] \iff \exists v' : \binom{u'}{v'} \in \Big[\binom{u_0}{v_0}\Big]\Big).$$

Clearly, $\sim_{\mathrm{I}}$ is an equivalence relation; in the semigroup game we are going to define, the moves of Player I are the $\sim_{\mathrm{I}}$-equivalence classes. For $u \in \mathbb{B}^*$, the $\sim_{\mathrm{I}}$-equivalence class of u (denoted $[u]$) can be identified with a subset of the set of all $\sim_{\mathrm{O}}$-classes; this subset contains all $\sim_{\mathrm{O}}$-classes which have a representative with u in the first component. That means, $\sim_{\mathrm{I}}$ is essentially the power set of $\sim_{\mathrm{O}}$. Since $\sim_{\mathrm{O}}$ has finite index, we get that $\sim_{\mathrm{I}}$ has finite index as well; more precisely it holds $\mathrm{index}(\sim_{\mathrm{I}}) \leq 2^{\mathrm{index}(\sim_{\mathrm{O}})}$. Analogously to Lemma 5.8, we show that $\sim_{\mathrm{I}}$ is computable.

Lemma 5.11. *Let $u \in \mathbb{B}^*$. Then, the set $[u]$ is a regular $*$-language over $\mathbb{B}$.*

Proof. We construct an automaton $\mathcal{A}_{[u]}$ recognizing the language $[u]$ as follows: first, we have to check for which $\sim_{\mathrm{O}}$-classes $\left[\binom{u_0}{v_0}\right]$ there exists $v \in \mathbb{B}^{|u|}$ such that $\binom{u}{v} \in \left[\binom{u_0}{v_0}\right]$. Let $\mathcal{B}$ be a DFA recognizing $\left[\binom{u_0}{v_0}\right]$. We take the projection on the first component (deleting the second component from the transitions of $\mathcal{B}$) and test whether the resulting automaton, say $\mathcal{B}'$, accepts u. If we do the same for all $\sim_{\mathrm{O}}$-classes, then we obtain r automata $\mathcal{B}'_1, \dots, \mathcal{B}'_r$ accepting u, and s automata $\mathcal{B}'_{r+1}, \dots, \mathcal{B}'_{r+s}$ not accepting u, where $r + s = \mathrm{index}(\sim_{\mathrm{O}})$. From these automata we can effectively construct an automaton for $[u]$, because

$$[u] = \bigcap_{i=1,\dots,r} L(\mathcal{B}'_i) \cap \bigcap_{j=r+1,\dots,r+s} \overline{L(\mathcal{B}'_j)}.$$

□

For each $u \in \mathbb{B}^*$, we get a unique partition into two sets of automata, one containing those $\mathcal{B}'_i$ which accept u and one containing those $\mathcal{B}'_j$ which do not accept u. Thus, considering each of the $2^{\mathrm{index}(\sim_{\mathrm{O}})}$ possible combinations of the above intersection, we get that $\sim_{\mathrm{I}}$ is computable.

Corollary 5.12. *Let $\mathcal{A}$ be a DPA with n states and m colors. Then, the relation $\sim_{\mathrm{I}}$ is computable in time $\mathcal{O}(2^{(mn)^n + (mn)^{2n}})$.*

Proof. Recall that there are at most $(mn)^n$ $\sim_{\mathrm{O}}$-equivalence classes, each of which is recognized by an automaton of size $\mathcal{O}((mn)^n)$. For $u \in \mathbb{B}^*$, the construction of $\mathcal{A}_{[u]}$ includes determinization of those automata $\mathcal{B}'_j$ not accepting u. Hence, each automaton $\mathcal{A}_{[u]}$ needs time $\mathcal{O}((2^{(mn)^n})^{(mn)^n}) = \mathcal{O}(2^{(mn)^{2n}})$ to be constructed and has at most $2^{(mn)^{2n}}$ states. Accordingly, the time required to compute the whole equivalence relation $\sim_{\mathrm{I}}$ is in $\mathcal{O}(2^{\mathrm{index}(\sim_{\mathrm{O}})} \cdot 2^{(mn)^{2n}}) = \mathcal{O}(2^{(mn)^n + (mn)^{2n}})$. □

5.2.2 Definition of the Semigroup Game

In this section we define a game induced by a DPA $\mathcal{A}$ over $\mathbb{B}^2$, where the moves of Player I and Player O are classes from $\mathbb{B}^*/_{\sim_{\mathrm{I}}}$ and $L_{\mathrm{eq}}/_{\sim_{\mathrm{O}}}$, respectively. Accordingly, we call it the *semigroup game* of $\mathcal{A}$, and we denote it $\Gamma_{\mathrm{SG}}(\mathcal{A})$. (Usually, we assume $\mathcal{A}$ implicitly and write simply Γ_{SG}.)

The game Γ_{SG} is defined similar to the block game Γ'. The only difference is that the players do not choose concrete words but the respective classes from the relations $\sim_{\mathrm{I}}$ and $\sim_{\mathrm{O}}$. A play is built up as follows: in the first round, Player I chooses two infinite classes $[u_0], [u_1] \in \mathbb{B}^*/_{\sim_{\mathrm{I}}}$; after that, Player O chooses a class $\left[\binom{u_0}{v_0}\right] \in L_{\text{eq}}/_{\sim_{\text{O}}}$. In each round thereafter, i.e., for $i \geq 2$, Player I chooses an infinite class $[u_i] \in \mathbb{B}^*/_{\sim_{\text{I}}}$ and Player O chooses a class $\left[\binom{u_{i-1}}{v_{i-1}}\right] \in L_{\text{eq}}/_{\sim_{\text{O}}}$. Note that, for each i, Player O's move $\left[\binom{u_i}{v_i}\right]$ is partially determined by Player I's choice $[u_i]$. Hence, the winning condition can be defined solely on Player O's moves: a play is winning for her if $\binom{u_0}{v_0}\binom{u_1}{v_1}\binom{u_2}{v_2}\cdots$ is accepted by $\mathcal{A}$.

Note that $\mathbb{B}^*/_{\sim_{\text{I}}}$ contains at least one infinite class because $\mathbb{B}^*$ is infinite and $\sim_{\text{I}}$ has finite index. Moreover, for each class $[u]$ there exists at least one class in $L_{\text{eq}}/_{\sim_{\text{O}}}$ associated with $[u]$ (by the definition of $\sim_{\text{I}}$), since u must appear in the first component of some word $\binom{u}{v}$. Hence, both players can always move. Furthermore, the winning condition of Γ_{SG} is well-defined because acceptance of $\mathcal{A}$ is independent of representatives. Given an infinite sequence over $\mathbb{B}^2$, one can iteratively replace any finite infix by a $\sim_{\text{O}}$-equivalent one, without touching membership in $L(\mathcal{A})$. More formally, for two ω-words $\binom{u_0}{v_0}\binom{u_1}{v_1}\cdots$ and $\binom{u'_0}{v'_0}\binom{u'_1}{v'_1}\cdots$ with $\binom{u_i}{v_i} \sim_{\text{O}} \binom{u'_i}{v'_i}$ for all $i \in \mathbb{N}$, it holds

$$\binom{u_0}{v_0}\binom{u_1}{v_1}\cdots \in L(\mathcal{A}) \iff \binom{u'_0}{v'_0}\binom{u'_1}{v'_1}\cdots \in L(\mathcal{A}).$$

Γ_{SG} can be modeled by a parity game on a graph with at most $2^{2(mn)^n+1}mn$ vertices. Thus, its winner is computable by standard techniques (see for example [GTW02]). In the vertices we keep track of the $\sim_{\text{I}}$-equivalence classes recently chosen by Player I, a color depending on the course of the play and the current state q of $\mathcal{A}$. The vertex reached by a move $\left[\binom{u}{v}\right]$ of Player O is colored by $\mu\binom{u}{v}_{q,q'}$, where q' is the state reached in $\mathcal{A}$ from q when reading $\binom{u}{v}$.

5.2.3 Simulation of the Block Game

In this section we prove correctness of the second step of our reduction. We show that Player I has a winning strategy for the block game Γ'_f for all functions $f : \mathbb{N} \to \mathbb{N}_+$ if and only if he wins the semigroup game Γ_{SG}. This completes the reduction and also yields the proof of item 1 of Theorem 4.12, i.e., decidability of Problem 4.11. The proof is divided into two parts, Lemma 5.13 and Lemma 5.14. Item 2 of Theorem 4.12, i.e., the reduction from finite delay to constant delay, is shown in Lemma 5.15, separately.

We first approach the connection between the block game and the semigroup game. The basic idea for the proof of Lemma 5.14 is, for arbitrary f, to simulate the moves of the players in Γ'_f by the corresponding equivalence classes of the relations $\sim_{\text{I}}$ and $\sim_{\text{O}}$, and vice versa. There, one has the problem whether a class $[u_i] \in \mathbb{B}^*/_{\sim_{\text{I}}}$ has an appropriate representative, i.e., one of length between $f(i)$ and $2f(i)$. We use Lemma 1.2 to show that there is a particular function f_0 such that for each f with $f \sqsupseteq f_0$ a representative of the required length indeed exists. Then, the following lemma completes the proof.

Lemma 5.13. *Player I wins Γ'_g for all functions $g : \mathbb{N} \to \mathbb{N}_+$ if and only if there exists a function $f_0 : \mathbb{N} \to \mathbb{N}_+$ such that Player I wins Γ'_f for all $f \sqsupseteq f_0$.*

Proof. The implication from left to right is immediate, setting $f_0 := \langle 0 \rangle$. We show the converse by contraposition. Assume there exists g_0 such that Player I does not win Γ'_{g_0}. Determinacy yields that Player O wins Γ'_{g_0}. By Proposition 5.2 Player O wins $\Gamma_{g'_0}$; that implies that she also wins Γ_g for all $g \sqsupseteq g'_0$ (see Remark 4.13). Proposition 5.3 yields that Player O wins $\Gamma'_{g''}$ for all $g \sqsupseteq g'_0$.

Towards a contradiction, let f_0 be a function such that Player I wins Γ'_f for all $f \sqsupseteq f_0$, and let f_* be the maximum of f_0 and g'_0, i.e., for all $i \in \mathbb{N}$ we have

$$f_*(i) := \max\{f_0(i), g'_0(i)\}.$$

Since $f_* \sqsupseteq g'_0$ it holds that Player O wins $\Gamma'_{f''_*}$. Recall that for each f it holds $f'' \sqsupseteq f$ (cf. page 108). Thus, Player I must win $\Gamma'_{f''_*}$, by assumption, because $f''_* \sqsupseteq f_* \sqsupseteq f_0$. This yields a contradiction which means that f_0 cannot exist. ☐

In the upcoming lemma we finally establish the simulation between the block game and the semigroup game. We show how to compute an appropriate candidate for the function f_0 mentioned in the previous lemma. We have to guarantee that every infinite $\sim_{\mathrm{I}}$-equivalence class contains a word of length between f_0 and $2f_0$; then, the same also holds for every f with $f \sqsupseteq f_0$. Since every $\sim_{\mathrm{I}}$-class is a regular $*$-language (cf. Lemma 5.11), we can apply Lemma 1.2 to prove that f_0 can be chosen to be the maximal number of states among all automata recognizing the $\sim_{\mathrm{I}}$-classes.

Lemma 5.14. *Player I wins Γ_{SG} if and only if there is a function $f_0 : \mathbb{N} \to \mathbb{N}_+$ such that Player I wins Γ'_f for all $f \sqsupseteq f_0$.*

Proof. We start with the implication from right to left. Let $f_0 : \mathbb{N} \to \mathbb{N}_+$ be a function such that Player I wins Γ'_f for all $f \sqsupseteq f_0$. We define a function f_* such that $f_* \sqsupseteq f_0$ and each word of length $f_*(i)$ is contained in an infinite $\sim_{\mathrm{I}}$-class, for all $i \in \mathbb{N}$. To this end, let d' be the length of a longest word in all finite $\sim_{\mathrm{I}}$-classes[1] and define, for all $i \in \mathbb{N}$, $f_*(i) := \max\{f_0(i), d' + 1\}$.

Since $f_* \sqsupseteq f_0$, Player I wins Γ'_{f_*} by assumption, and a winning strategy yields his first two moves u_0, u_1. By our above remarks, both $[u_0]$ and $[u_1]$

[1]If $\sim_{\mathrm{I}}$ has no finite equivalence class, then we define $d' := 0$.

must be infinite, and so he can choose them in Γ_{SG}. We simulate Player O's answer $\left[\binom{u_0}{v_0}\right]$ by choosing v_0 in Γ'_{f_*}, and Player I's winning strategy yields u_2 with $[u_2]$ being infinite. Choosing $[u_2]$ in Γ_{SG} we obtain the next move $\left[\binom{u_1}{v_1}\right]$ of Player O, and so on.

We argue that the plays built up induce the same maximal color occurring infinitely often. It suffices to show that in both plays a move of Player O leads $\mathcal{A}$ to the same state, via paths with equal maximal color. Then, the rest follows by induction. Let q_i be the current state of $\mathcal{A}$ and u_i, u_{i+1} be the latest words chosen by Player I. If Player O chooses $\left[\binom{u_i}{v_i}\right]$ in Γ_{SG}, then we reach the state $q_{i+1} := \delta^*\big(q_i, \binom{u_i}{v_i}\big)$ via the maximal color $\mu\binom{u_i}{v_i}_{q_i,q_{i+1}}$. The state q_{i+1} is well-defined because from q_i every $\binom{u'_i}{v'_i} \in \left[\binom{u_i}{v_i}\right]$ leads $\mathcal{A}$ to the same state, though via different paths, but with the same maximal color. In Γ'_{f_*} Player O chooses v_i. As in Γ_{SG}, we reach the state q_{i+1} via the maximal color $\mu\binom{u_i}{v_i}_{q_i,q_{i+1}}$.

Conversely, assume that Player I wins Γ_{SG}. Let $\mathcal{A}_1, \ldots, \mathcal{A}_r$ be automata recognizing all the $\sim_{\mathrm{I}}$-classes, and n' be the maximal number of states among these automata: $n' := \max\{n_1, \ldots, n_r\}$, where n_j is the number of states of $\mathcal{A}_j$ for $j = 1, \ldots, r$. For the rest of this proof, let f_0 be the constant function with $f_0(i) := n'$ for all $i \in \mathbb{N}$. (To simplify matters, we write f_0 instead of $f_0(i)$.) We first show that Player I wins Γ'_{f_0}: Player I's winning strategy in Γ_{SG} yields $[u_0], [u_1]$. Since $[u_0], [u_1]$ are infinite, we can apply Lemma 1.2. Accordingly, each $\mathcal{A}_j$ with $|L(\mathcal{A}_j)| = \infty$ accepts a word of length between f_0 and $f_0 + n_j$ and thus between f_0 and $2f_0$, because $n_j \leq f_0$. Hence, we can assume w.l.o.g. that $f_0 \leq |u_0|, |u_1| \leq 2f_0$. Player I chooses u_0, u_1 in Γ'_{f_0} and Player O answers by a word v_0 with $|v_0| = |u_0|$. We simulate this move by $\left[\binom{u_0}{v_0}\right]$ in Γ_{SG} and obtain Player I's answer $[u_2]$, so the next move of Player I in Γ'_{f_0} is u_2 (for appropriate u_2). Player O chooses v_1 with $|v_1| = |u_1|$, and so forth.

Using the same inductive argument as above, the plays built up have the same maximal color occurring infinitely often. Starting at q_i, Player O's move v_i in Γ'_{f_0} has the same effect as the corresponding move $\left[\binom{u_i}{v_i}\right]$ in Γ_{SG}, i.e., we reach the state $q_{i+1} := \delta^*\big(q_i, \binom{u_i}{v_i}\big)$ via the maximal color $\mu\binom{u_i}{v_i}_{q_i,q_{i+1}}$.

We complete the proof (of the implication from left to right) by showing that Player I wins Γ'_f for all $f \sqsupseteq f_0$. Let $||[a,b]|| := b-a$ be the size of the interval $[a,b]$. If $f \sqsupseteq f_0$, then (since $||[f_0, 2f_0]|| = n'$) it holds $||[f(i), 2f(i)]|| \geq n'$, for all $i \in \mathbb{N}$. Hence, to win Γ'_f Player I simply needs to choose longer representatives of the $\sim_{\mathrm{I}}$-classes than in Γ'_{f_0}. $\square$

In the proof of Corollary 5.12 we have argued that each $\sim_{\mathrm{I}}$-equivalence class is recognized by a DFA with at most $2^{(mn)^{2n}}$ states. Thus, we can bound the constant n' from the previous proof to be at most $2^{(mn)^{2n}}$; thereby, we have also found an appropriate candidate for the function f_0, namely $f_0(i) := 2^{(mn)^{2n}}$ for all $i \in \mathbb{N}$.

We have just accomplished the proof of item 1 of Theorem 4.12. The previous lemma completes our reduction of the problem whether a regular language L over $\mathbb{B}^2$ is solvable with finite delay to the question whether Player O wins the semigroup game induced by some DPA $\mathcal{A}$ recognizing L. As already remarked, the latter can be answered, because the semigroup game is a parity game on a finite graph.

Next, we approach item 2 of Theorem 4.12. We show that each regular specification which admits a finite-delay solution also allows for one of constant delay. The idea for the proof of the upcoming lemma is two consider the length of a shortest representative in each $\sim_{\mathrm{I}}$-class.

Lemma 5.15. *Let $f_0 := n'$ with n' as in the proof of Lemma 5.14. Then, Player O wins Γ_{SG} if and only if she wins $\Gamma_{\langle 2n'-1\rangle}$.*

Proof. Let u of length d' be a longest word in all finite $\sim_{\mathrm{I}}$-equivalence classes. Moreover, let $L(\mathcal{A}_1) = [u]$, where $\mathcal{A}_1$ has n_1 states. Then we have $d' < n_1$. Otherwise, the run of $\mathcal{A}_1$ on u had a loop, which is a contradiction to the finiteness of $L(\mathcal{A}_1)$. Since $n_1 \leq n'$ we get $d' < n'$ and so $d' + 1 \leq n'$. Thus, each $\sim_{\mathrm{I}}$-class containing a word of length at least f_0 is infinite.

Assume that Player O wins Γ_{SG}. We first show that Player O wins Γ'_{f_0}. Let u_0, u_1 with $n' \leq |u_0|, |u_1| \leq 2n'$ be the first move of Player I in Γ'_{f_0}. By the above remarks $[u_0], [u_1]$ are infinite, and we can simulate $[u_0], [u_1]$ in Γ_{SG}. Player O's winning strategy in Γ_{SG} yields $\left[\binom{u_0}{v_0}\right]$ for some suitable v_0. Let her choose v_0 in Γ'_{f_0}. Then Player I chooses u_2 and we simulate $[u_2]$ in Γ_{SG}, and so on.

As in the proof of Lemma 5.14, we obtain plays with the same maximal color occurring infinitely often, and so Player O wins Γ'_{f_0}.

We obtain a winning strategy for Player O in $\Gamma_{\langle 2n'-1\rangle}$, simulating one in Γ'_{f_0} as follows: the bits $a_0 \cdots a_{2n'-1}$ chosen by Player I in his first move in $\Gamma_{\langle 2n'-1\rangle}$ are simulated in Γ'_{f_0} by the two words u_0, u_1, each of which has length n', namely $u_0 := a_0 \cdots a_{n'-1}$ and $u_1 := a_{n'} \cdots a_{2n'-1}$. Player O's winning strategy yields her first move v_0 in Γ'_{f_0}; the word v_0 can be used to give her n' first moves in $\Gamma_{\langle 2n'-1\rangle}$. Thereby, we obtain n' new bits by Player I which can be used to simulate his next move u_2 in the block game induced by f_0, and so forth.

Since Player O plays a winning strategy in Γ'_{f_0} and the ω-words built up in Γ'_{f_0} and $\Gamma_{\langle 2n'-1\rangle}$ are exactly the same, she also wins $\Gamma_{\langle 2n'-1\rangle}$.

Conversely, let Player O win $\Gamma_{\langle 2n'-1\rangle}$ and let $g_0(i) := 2n'$, for all $i \in \mathbb{N}$. Since we have $g_0 \sqsupseteq \langle 2n'-1\rangle$, Player O wins Γ_{g_0}. Then, by Proposition 5.3, she also wins $\Gamma'_{g_0''}$. Given a winning strategy for Player O in $\Gamma'_{g_0''}$ we can specify one for her in Γ_{SG} as follows: a move $[u_i]$ of Player I is simulated by u_i in $\Gamma'_{g_0''}$. By Lemma 1.2, an appropriate representative u_i of length $g_0''(i) \le |u_i| \le 2g_0''(i)$ must exist because $g_0'' \sqsupseteq g_0$, and so $|[g_0''(i), 2g_0''(i)]| \ge n'$ for all $i \in \mathbb{N}$. We use Player O's answer v_{i-1} to choose $\left[\binom{u_{i-1}}{v_{i-1}}\right]$ in Γ_{SG}. This yields a play winning for Player O in Γ_{SG}. □

With Corollary 5.4 and Lemmas 5.13 and 5.14 we have shown that the problem whether $L(\mathcal{A})$ is solvable with finite delay is reducible to the question whether Player O wins Γ_{SG}. In the framework of Section 4.1 this means that we can decide whether there exists a continuous operator satisfying a regular specification (given by a DPA $\mathcal{A}$).

We estimated the game graph of Γ_{SG} to have at most $2^{2(mn)^n+1}mn$ vertices, where n is the number of states and m the number of colors of $\mathcal{A}$ (see at the end of Section 5.2.2). Since we require only m colors to express the parity winning condition of Γ_{SG}, its winner can be computed in time $\mathcal{O}((2^{2(mn)^n+1}mn)^m)$, according to [Sch07]. Let us summarize the obtained results.

Theorem 5.16. *Let $\mathcal{A}$ be a DPA over $\mathbb{B}^2$. The problem whether $L(\mathcal{A})$ is solvable with finite delay and the problem whether there is a continuous operator λ such that it holds $\{\binom{\alpha}{\lambda(\alpha)} \mid \alpha \in \mathbb{B}^\omega\} \subseteq L(\mathcal{A})$ are in* 2ExpTime.

Finally, Lemma 5.15 shows that $L(\mathcal{A})$ is solvable with finite delay if and only if it is solvable with constant delay. Moreover, the lemma exhibits a doubly exponential upper bound for the required constant.

Theorem 5.17. *Let $\mathcal{A}$ be a DPA over $\mathbb{B}^2$ with n states and m colors, and $n' := 2^{(mn)^{2n}}$. Then, the following are equivalent:*

1. *$L(\mathcal{A})$ is solvable with finite delay.*
2. *$L(\mathcal{A})$ is solvable with constant delay $2n'-1$.*
3. *There is a continuous operator λ such that $\{\binom{\alpha}{\lambda(\alpha)} \mid \alpha \in \mathbb{B}^\omega\} \subseteq L(\mathcal{A})$.*
4. *There is a $(2n'-1)$-delay operator λ such that $\{\binom{\alpha}{\lambda(\alpha)} \mid \alpha \in \mathbb{B}^\omega\} \subseteq L(\mathcal{A})$.*

5.3 Delay Values

For the class of regular ω-languages, we have established decidability of Problem 4.11. Moreover, we have shown that each regular specification allowing for a finite-delay solution also admits a constant-delay solution.

In our setting, we have assumed that the function f induces a delay on the moves of Player O: the greater the function f, the greater the advantage for her. However, we may as well consider the delay being imposed on the moves of Player I; then Player O has to move ahead and an increasing delay function is an increasing disadvantage for her. The purpose of the present section is to show that the heart of the problem, i.e., determining whether a regular ω-language has a finite-delay solution, is independent of the fact who of the two players moves ahead.

To this end, let us generalize the delay game Γ_f as follows: we modify the range of the function f such that the value $f(i)$ captures the increase of the delay in round i, from Player O's point of view; negative f-values are hence a convenient way for describing a delay of Player I's moves. For the rest of this section, we consider the set of possible delay functions to be $F_* := F_+ \cup F_-$ where

$$F_+ := \{f \mid f : \mathbb{N} \to \mathbb{Z}_+\} \text{ and } F_- := \{f \mid f : \mathbb{N} \to \mathbb{Z}_-\}.$$

If $f \in F_+$ then the delay game is played as before (note that $\mathbb{Z}_+ = \mathbb{N}_+$), i.e., Player I moves ahead. If $f \in F_-$ then the roles of the players are swapped as follows: in each round, Player O has to choose $|f(i)|$ bits and Player I chooses one bit, afterwards. Then, we may raise the question whether it is possible for Player I to delay his moves for a finite number of steps such that he has a winning strategy in the corresponding game. This is the same as to ask for the existence of a function $f \in F_-$ such that Player I wins $\Gamma_f(L)$. Thus, we can formulate a dual version of Problem 4.11 as follows.

Problem 5.18. Let L be an ω-language over $\mathbb{B}^2$. Does there exist a function $f \in F_-$ such that Player I wins $\Gamma_f(L)$?

If Player I wins $\Gamma_f(L)$ for some $f \in F_-$, then he can force the play into the complement language $\overline{L} := (\mathbb{B}^2)^\omega \setminus L$. Accordingly, we say that $\overline{L}$ is *solvable with finite shift*, and analogously for restricted classes of functions (cf. page 101). Following the notation of [HL72], a *shift* function $f \in F_-$ can also be interpreted as a function inducing *negative delay*; we may use each term in place of the other. Accordingly, the function of constant shift d is referred to as the function $\langle -d \rangle$, i.e., the function inducing constant delay $-d$, where $\langle -d \rangle(0) = -(d+1)$ and $\langle -d \rangle(i) = -1$ for $i \geq 1$. Note that our notation distinguishes between the function $\langle -0 \rangle$ of constant shift 0 and the function $\langle 0 \rangle$ of constant delay 0. In the first one Player O begins, whereas in the second one Player I begins. This may make a difference in who has a winning strategy, for example, if we consider the regular winning condition $L := \{\binom{a_0}{b_0}\binom{a_1}{b_1}\binom{a_2}{b_2}\cdots \in (\mathbb{B}^2)^\omega \mid a_0 = b_0\}$.

For $f \in F_*$, we can observe properties analogous to the ones made for the normal delay game in Remark 4.13: items 1 and 2 directly translate to the generalized setting, as the basic modeling of the delay game is not effected by changing the roles of the players. For item 3, we naturally extend the relation $\sqsubseteq$ to the set F_*. However, winning is still monotone. If $f, g \in F_*$ and $g : \mathbb{N} \to \mathbb{Z}_-$, then $f \sqsubseteq g$ means that $|g(i)| \leq |f(i)|$ for every i, and accordingly Player O in $\Gamma_g(L)$ has to give less bits than in $\Gamma_f(L)$, in each round. Note that this is an advantage for Player O, compared to $\Gamma_f(L)$, and that she can simulate each winning strategy for $\Gamma_f(L)$ in $\Gamma_g(L)$.

Remark 5.19. Let $L \subseteq (\mathbb{B}^2)^\omega$ be ω-regular and $f_0 \in F_*$. Then the following hold:

1. The generalized delay game $\Gamma_{f_0}(L)$ is determined.
2. If f_0 is of bounded delay, then the winner of $\Gamma_{f_0}(L)$ is decidable.
3. If Player O wins $\Gamma_{f_0}(L)$ then she also wins $\Gamma_f(L)$, for all $f \sqsupseteq f_0$. Analogously, if Player I wins $\Gamma_{f_0}(L)$ then he also wins $\Gamma_f(L)$, for all $f \sqsubseteq f_0$.

We show that Problem 5.18 can be reduced to Problem 4.11. To do so, we modify the winning condition L and swap the roles of the players. Then, we can restrict the range of the function f to F_+ again, i.e., we are back in the setting of the previous sections.

Definition 5.20. Let $L \subseteq (\mathbb{B}^2)^\omega$ be a language. The *complement-flip* language of L is defined as

$$\overline{L}^{\updownarrow} := \{\beta^\wedge\alpha \mid \alpha^\wedge\beta \in \overline{L}\}.$$

If L is a regular language recognized by the DPA $\mathcal{A} = (Q, q_0, \delta, c)$, then $\overline{L}^{\updownarrow}$ is recognized by the DPA $\overline{\mathcal{A}}^{\updownarrow} = (Q, q_0, \delta', c')$, where in $\overline{\mathcal{A}}^{\updownarrow}$ the transition labels of $\mathcal{A}$ are flipped and the color of every state is increased by one (cf. [GTW02]). For all $q, q' \in Q, x, y, \in \mathbb{B}$ we define

$$\delta'\Big(q, \binom{x}{y}\Big) := q' :\Longleftrightarrow \delta\Big(q, \binom{y}{x}\Big) = q'$$

and

$$c'(q) := c(q) + 1.$$

Note that the size of $\overline{\mathcal{A}}^{\updownarrow}$ is linear in the size of $\mathcal{A}$.

Lemma 5.21. *Let $f_- \in F_-, f_+ := -f_- \in F_+$ and $L \subseteq (\mathbb{B}^2)^\omega$ be a language. Player I wins $\Gamma_{f_-}(L)$ if and only if Player O wins $\Gamma_{f_+}(\overline{L}^{\updownarrow})$.*

Proof. Let u_i and v_i be the moves made by Player I and Player O, respectively, in $\Gamma_{f_-}(L)$, and analogously with u'_i, v'_i in $\Gamma_{f_+}(\overline{L}^{\updownarrow})$.

Assume Player I has a winning strategy in $\Gamma_{f_-}(L)$. This winning strategy is simulated in $\Gamma_{f_+}(\overline{L}^{\updownarrow})$ as follows: Player I chooses a word $u'_0 \in \mathbb{B}^{f_+(0)}$. We simulate u'_0 as Player O's first move in $\Gamma_{f_-}(L)$, i.e., we set $v_0 := u'_0$. Player I's answer is the bit u_0, and we take it as Player O's first move v'_0 in $\Gamma_{f_+}(\overline{L}^{\updownarrow})$: $v'_0 := u_0$. Player I answers by his second move u'_1, which we simulate as Player O's second move v_1 in $\Gamma_{f_-}(L)$, meaning $v_1 := u'_1$, and so forth.

We build up the play $\binom{u_0}{u'_0}\binom{u_1}{u'_1}\binom{u_2}{u'_2}\cdots$ in $\Gamma_{f_-}(L)$ and the play $\binom{u'_0}{u_0}\binom{u'_1}{u_1}\binom{u'_2}{u_2}\cdots$ in $\Gamma_{f_+}(\overline{L}^{\updownarrow})$. Since Player I plays a winning strategy in $\Gamma_{f_-}(L)$, the play built up is winning for him, i.e., it holds $\binom{u_0}{u'_0}\binom{u_1}{u'_1}\binom{u_2}{u'_2}\cdots \notin L$. This means that $\binom{u_0}{u'_0}\binom{u_1}{u'_1}\binom{u_2}{u'_2}\cdots \in \overline{L}$, therefore $\binom{u'_0}{u_0}\binom{u'_1}{u_1}\binom{u'_2}{u_2}\cdots \in \overline{L}^{\updownarrow}$. Thus, the play built up in $\Gamma_{f_+}(\overline{L}^{\updownarrow})$ is winning for Player O.

The implication from right to left is shown analogously. □

The above lemma means that every function $f_- \in F_-$ witnessing that $\overline{L}$ is solvable with finite shift has a counterpart $f_+ \in F_+$ such that $\overline{L}^{\updownarrow}$ is solvable with delay f_+. As a direct consequence, we get that Problem 5.18 is reducible to Problem 4.11 (replacing L by $\overline{L}^{\updownarrow}$). Since, for regular L, the language $\overline{L}^{\updownarrow}$ is regular as well, we can apply the techniques from the previous sections to decide whether $\overline{L}$ has a finite-shift solution, obtaining analogous results. Moreover, we obtain the same asymptotic running time because $\overline{\mathcal{A}}^{\updownarrow}$ can be constructed from $\mathcal{A}$ in linear time.

Theorem 5.22. *Let $\mathcal{A}$ be a DPA over $\mathbb{B}^2$ recognizing the regular ω-language L. Then the following hold:*

1. *The problem whether $\overline{L}$ is solvable with finite shift is in* 2ExpTime.
2. *If $\overline{L}$ is solvable with finite shift, then it is solvable with constant shift d, for some d doubly exponential in $|\overline{\mathcal{A}}^{\updownarrow}|$.*

For a DPA $\mathcal{A}$ over $\mathbb{B}^2$ recognizing the regular ω-language L, Theorems 5.17 and 5.22 mean the following: L is solvable with finite delay if and only if L is solvable with constant delay $2n'-1$, where $n' := 2^{(mn)^{2n}}$ is determined by the number n of states and the number m of colors of $\mathcal{A}$. Analogously, $\overline{L}$ is solvable with finite shift if and only if $\overline{L}$ is solvable with constant shift $2n'-1$. It may be the case that both $\overline{L}$ has a finite-shift solution and L has a finite-delay solution. Then, there exists a minimal constant d such that Player O wins $\Gamma_{\langle d\rangle}(L)$; d is called the *delay value* of L. We obtain the following generalized determinacy of regular Gale-Stewart games.

Theorem 5.23 (Generalized Determinacy). *Let L be recognized by the DPA $\mathcal{A}$ over $\mathbb{B}^2$ with n states and m colors, and let $n' := 2^{(mn)^{2n}}$. Then precisely one of the following holds:*

1. *Player I wins $\Gamma_f(L)$, for all $f \in F_*$.*
2. *Player O wins $\Gamma_f(L)$, for all $f \in F_*$.*
3. *There exists an integer[2] $d_L \in [-(2n'-1), 2n'-1]$ uniquely determined by L such that it holds*

$$d_L = \min\{d \mid \textit{Player O wins } \Gamma_{\langle d\rangle}(L)\}$$

and $d_L - 1 = \max\{d \mid \textit{Player I wins } \Gamma_{\langle d\rangle}(L)\}$.

Moreover, from Theorems 5.16 and 5.22 we can draw the conclusion that we can decide which case of Theorem 5.23 holds.

Theorem 5.24. *Let $\mathcal{A}$ be a DPA over $\mathbb{B}^2$ recognizing L. Then, the problem to decide which case of Theorem 5.23 holds is in* 2ExpTime. *If case 3 applies, then one can compute the delay value d_L in triply exponential time.*

Proof. Let $\mathcal{A}$ have n states and m colors. By Theorem 5.16, we can decide in 2ExpTime whether L is solvable with finite delay. To do so, we construct the semigroup game $\Gamma_{\mathrm{SG}}(\mathcal{A})$ and solve it. By our main reduction, L has a finite-delay solution if and only if Player O wins $\Gamma_{\mathrm{SG}}(\mathcal{A})$; if Player I wins then L is not solvable with finite delay, which means that case 1 of Theorem 5.23 holds.

Analogously, we can decide whether $\overline{L}$ admits a finite-shift solution (cf. Theorem 5.22), by solving the semigroup game $\Gamma_{\mathrm{SG}}(\overline{\mathcal{A}}^{\updownarrow})$. Recall that $\overline{\mathcal{A}}^{\updownarrow}$ is of size linear in $\mathcal{A}$, therefore a solution to $\Gamma_{\mathrm{SG}}(\overline{\mathcal{A}}^{\updownarrow})$ can be computed in 2ExpTime. If Player O wins $\Gamma_{\mathrm{SG}}(\overline{\mathcal{A}}^{\updownarrow})$, then $\overline{L}$ is solvable with finite shift. Otherwise, Player I wins $\Gamma_{\mathrm{SG}}(\overline{\mathcal{A}}^{\updownarrow})$ and case 2 of Theorem 5.23 applies.

The case where Player I wins both $\Gamma_{\mathrm{SG}}(\mathcal{A})$ and $\Gamma_{\mathrm{SG}}(\overline{\mathcal{A}}^{\updownarrow})$ is not possible, because the main reduction of the previous sections and Lemma 5.21 yield that both players win $\Gamma_f(L)$ for all $f \in F_*$, a contradiction.

If Player O wins both $\Gamma_{\mathrm{SG}}(\mathcal{A})$ and $\Gamma_{\mathrm{SG}}(\overline{\mathcal{A}}^{\updownarrow})$, then L is solvable with finite delay and $\overline{L}$ is solvable with finite shift. Hence, there exists a constant $d_+ \geq 0$ such that Player O wins $\Gamma_{\langle d_+\rangle}(L)$ and a constant $d_- \leq -0$ such that Player I wins $\Gamma_{\langle d_-\rangle}(L)$. By monotonicity, there must exist a minimal d^* with $d_- \leq d^* \leq d_+$ such that Player O wins $\Gamma_{\langle d^*\rangle}(L)$, i.e., Player I wins $\Gamma_{\langle d\rangle}(L)$ for all $d < d^*$;

[2] Recall that our notation distinguishes between the functions $\langle -0\rangle$ and $\langle 0\rangle$. To cope with this, let -1 be the predecessor of -0, and let -0 be the predecessor of 0.

note that $d^* < 0$ is possible. Since, for given $f \in F_*$, the winner of $\Gamma_f(L)$ depends only on L (but not on its representation $\mathcal{A}$), the constant d^* also depends only on L. Altogether, case 3 of Theorem 5.23 holds, and d^* is the delay value d_L.

To compute d_L we search through the interval $[-(2n'-1), 2n'-1]$ via binary search. Throughout the computation we assume that $d = \lceil \frac{d_- + d_+}{2} \rceil$, where we start with $d_- := -(2n'-1)$ and $d_+ := 2n'-1$.[3] If Player O wins $\Gamma_{\langle d \rangle}(L)$ then $d_+ := d$ and if Player I wins $\Gamma_{\langle d \rangle}(L)$ then $d_- := d$, such that the size of the remaining interval is halved. Afterwards, we update the value d and proceed analogously, until $d_- = d_+$.

By the above procedure, we obtain d_L in at most $\log n' + 2$ iterations. For each d, the game $\Gamma_{\langle d \rangle}(L)$ can be modeled as a parity game on a graph with $\mathcal{O}(n2^d)$ vertices, where we memorize the word one of the players moves ahead. Since d is at most doubly exponential and we require only m colors for the parity condition, the game $\Gamma_{\langle d \rangle}(L)$ can thus be solved in triply exponential time. Since the number of iterations is only exponential, the overall running time to compute d_L is also triply exponential. □

Let us reformulate our results in terms of delay operators: first of all, note that the determinacy of a regular condition L (established in [BL69]) means that either L or $\overline{L}$ is solvable by a 1-Lipschitz continuous operator, i.e., an operator for which the constant 1 witnesses Lipschitz continuity (cf. page 99). In other words, either L or $\overline{L}$ contains the graph of a 1-Lipschitz continuous operator. This is due to the fact that the i-th output bit depends only on the first i input bits and, therefore, for all $\alpha, \beta \in \mathbb{B}^\omega$ with $\alpha \neq \beta$ it holds

$$\frac{\operatorname{dist}(\lambda(\alpha), \lambda(\beta))}{\operatorname{dist}(\alpha, \beta)} \leq 1.$$

From Theorem 5.23 it follows that, even if L has no 1-Lipschitz continuous solution, it may nonetheless be solvable by a c-Lipschitz continuous operator, for some $c \leq 2^{2n'-1}$. Theorem 5.17 implies that there exists a continuous solution if and only if there exists a $2^{2n'-1}$-Lipschitz continuous solution. That means, for finding a continuous solution, it suffices to explore the set of operators of delay bounded by $2n'-1$. Note that this already yields decidability of the problem whether a regular condition is solvable by a continuous operator. Finally, Theorem 5.24 implies that one can compute the minimal Lipschitz constant required, if it exists, in triply exponential time.

[3]Sometimes, it may be necessary to consider $d = \lfloor \frac{d_- + d_+}{2} \rfloor$, if $d = \lceil \frac{d_- + d_+}{2} \rceil$ has already been considered.

Chapter 6

A Concurrent Setting

Concurrent games are used to model synchronous interaction in open systems [dAHM00, dAHM01]. The basic difference to turn-based games is that concurrent games normally cannot be won with deterministic strategies, i.e., they are non-determined in the classical sense. Consider the following example of a simple non-determined concurrent Gale-Stewart game.

Example 6.1. Let the players simultaneously choose one bit in each round of the game. The winning condition is given by the following regular expression:

$$r = \left(\left[\binom{0}{1} + \binom{1}{0}\right]^* \left[\binom{0}{0} + \binom{1}{1}\right]\right)^\omega$$

That means, a play is winning for Player O if and only if she chooses the same bit as her opponent, in infinitely many rounds. Clearly, since Player O does not know the bit a_i when choosing b_i, she cannot force to satisfy the winning condition. By an analogous argument, Player I cannot guarantee that the winning condition is violated. Thus, neither Player I nor Player O has a winning strategy, which means that the game is non-determined.

There are well-known techniques to decide whether a given concurrent game is determined with deterministic strategies (see for example [dAHM00, dAHM01]). Additionally, more sophisticated notions of strategies have been invented which ensure winning with a desired probability, possibly involving both random choice of moves and infinite memory [dAH00]. In the game of Example 6.1, Player O has an *almost-sure* winning strategy (cf. [JKH02]), i.e., a *mixed* strategy which is winning for her with probability 1. Here, we abstain from considering mixed strategies. Instead, we conduct the analysis of games with finite delay in a concurrent setting.

First of all, we introduce a concurrent game with delay and show that any winning strategy for one of the players can be simulated by a winning strategy in a turn-based game, and vice versa (see Propositions 6.2 and 6.3). As a corollary, we get that winning is still monotone with respect to the delay function.

For ω-regular specifications, we convey the techniques used in the previous chapter to prove analogous results for the concurrent setting. As it will turn out, a solution of finite delay can still be reduced to one of constant delay, which means that the delay value of a regular language is still computable in triply exponential time. Moreover, we can decide whether there exist delay functions high enough such that our game becomes non-determined, and we show how to compute upper bounds for these functions.

This chapter is organized as follows. First, we introduce a concurrent game with finite delay. In Section 6.1, we reduce the concurrent setting to the turn-based case; the findings presented hold for arbitrary classes of conditions. Finally, in Section 6.2 we consider the class of regular specifications and show results analogous to those in Chapter 5.

Definition of Γ^{cc}. Let us informally describe the *concurrent delay game* $\Gamma^{cc}_{f,g}(L)$. It is induced by an ω-language L over $\mathbb{B}^2$ and two functions $f, g : \mathbb{N} \to \mathbb{N}_+$. The function f imposes a delay on the moves of Player O, and analogously for the function g and the moves of Player I. (The concurrency inherent in the new setting makes an analysis of possible shift solutions meaningless, therefore f, g map into $\mathbb{N}_+$.)

A play proceeds as follows: in each round, i.e., for $i \geq 0$, each of the two players simultaneously chooses a non-empty word of predefined length. Player I chooses a word of length $f(i)$, and Player O chooses a word which has length $g(i)$. Thereby, each player builds up an infinite sequence; Player I builds up $\alpha = a_0a_1a_2\cdots$ and Player O builds up $\beta = b_0b_1b_2\cdots$ ($a_i, b_i \in \mathbb{B}$). The important difference to the normal delay game from Chapter 5 is that the moves of each player are hidden to the opponent, and they are only uncovered bit by bit during the course of the play: the bits a_i and b_i are communicated to Player O and Player I, respectively, after round i. That means, in each round both players have to commit to possibly several bits, but each of them gets the information about only one bit chosen by the opponent.

The winning condition is the same as for the normal delay game, i.e., the play built up is winning for Player O if $\alpha^\wedge\beta := \binom{a_0}{b_0}\binom{a_1}{b_1}\binom{a_2}{b_2}\cdots$ is contained in L. Otherwise, it is winning for Player I.

6.1 Reduction to the Turn-based Setting

The aim of this section is to reduce the concurrent setting to the turn-based one. Therefor, we show in the following two propositions that, for each player, a winning strategy in Γ^{cc} can be simulated by a winning strategy for Player I in the normal delay game. (Depending on which player we consider in Γ^{cc}, we may have to adapt the winning condition for the normal delay game.) The reason for this is as follows: in each of the two settings, Player I is shown one bit of Player O in each round. If Player I has a winning strategy in one of the games, then it makes no difference for him whether or not the opponent chooses concurrently, or whether the opponent, when taking a decision, knows about all of Player I's previous choices. Hence, Player I also has a winning strategy in the other game.

Proposition 6.2. *Let L be an ω-language over $\mathbb{B}^2$ and $f : \mathbb{N} \to \mathbb{N}_+$. Then, for all $g : \mathbb{N} \to \mathbb{N}_+$ it holds:*

$$\textit{Player I wins } \Gamma^{\mathrm{cc}}_{f,g}(L) \iff \textit{Player I wins } \Gamma_f(L)$$

Proof. We accomplish the proof by simulation of the winning strategies. Let u_i and v_i be the words chosen by Player I and Player O, respectively, in round i of $\Gamma^{\mathrm{cc}}_{f,g}(L)$, and analogously with u'_i, v'_i for $\Gamma_f(L)$. (Note that $v'_i \in \mathbb{B}$, for all $i \in \mathbb{N}$.)

Let Player I have a winning strategy in $\Gamma^{\mathrm{cc}}_{f,g}(L)$. It yields $u_0 \in \mathbb{B}^{f(0)}$, and we can choose $u'_0 := u_0$. Player O's answer $v'_0 \in \mathbb{B}$ in $\Gamma_f(L)$ is simulated in $\Gamma^{\mathrm{cc}}_{f,g}(L)$, such that after the first round the bit v'_0 is uncovered. Player I's winning strategy yields the word $u_1 \in \mathbb{B}^{f(1)}$, which is taken as u'_1. Player O's answer v'_1 is taken as her next bit in $\Gamma^{\mathrm{cc}}_{f,g}(L)$. (Note that this bit may belong to either v_0 or v_1, depending on g.) Player I's winning strategy in $\Gamma^{\mathrm{cc}}_{f,g}(L)$ yields $u_2 \in \mathbb{B}^{f(2)}$, and so on. The plays built up this way coincide. Thus, since Player I wins $\Gamma^{\mathrm{cc}}_{f,g}(L)$, he also wins $\Gamma_f(L)$.

The implication from right to left is accomplished analogously. □

For the simulation of a winning strategy for Player O in $\Gamma^{\mathrm{cc}}_{f,g}(L)$, we utilize the complement-flip language $\overline{L}^{\updownarrow}$ as winning condition for the turn-based game.

Proposition 6.3. *Let L be an ω-language over $\mathbb{B}^2$ and $g : \mathbb{N} \to \mathbb{N}_+$. Then, for all $f : \mathbb{N} \to \mathbb{N}_+$ it holds:*

$$\textit{Player O wins } \Gamma^{\mathrm{cc}}_{f,g}(L) \iff \textit{Player I wins } \Gamma_g(\overline{L}^{\updownarrow})$$

Proof. Again, we argue by simulation of the winning strategies. (To this end, let the words u_i, v_i, u'_i, v'_i have the same meaning as in the proof of Proposition 6.2.)

Assume Player O has a winning strategy in $\Gamma^{\mathrm{cc}}_{f,g}(L)$, which means that we obtain $v_0 \in \mathbb{B}^{g(0)}$, without any information about the move u_0 of Player I; set $u'_0 := v_0$. Player O's answer v'_0 is just one bit, and we simulate it in $\Gamma^{\mathrm{cc}}_{f,g}(L)$

such that it is taken as the first bit of u_0. This yields Player O's second move $v_1 \in \mathbb{B}^{g(1)}$ in $\Gamma^{\mathrm{cc}}_{f,g}(L)$, which we use as second move of Player I in $\Gamma_g(\overline{L^{\updownarrow}})$, i.e., $u'_1 := v_1$. Player O's answer v'_1 is taken as the next bit of Player I in $\Gamma^{\mathrm{cc}}_{f,g}(L)$; it belongs to either u_0 or u_1, depending on f. We obtain $v_2 \in \mathbb{B}^{g(2)}$, and proceed analogously.

We get $u_0u_1u_2\cdots = v'_0v'_1v'_2\cdots$ and $v_0v_1v_2\cdots = u'_0u'_1u'_2\cdots$, by construction. Since Player O plays a winning strategy in $\Gamma^{\mathrm{cc}}_{f,g}(L)$, we have $\binom{u_0}{v_0}\binom{u_1}{v_1}\binom{u_2}{v_2}\cdots \in L$. Hence, it also holds $\binom{u'_0}{v'_0}\binom{u'_1}{v'_1}\binom{u'_2}{v'_2}\cdots \notin \overline{L^{\updownarrow}}$, which means that the play in $\Gamma_g(\overline{L^{\updownarrow}})$ is winning for Player I.

The proof for the other direction works analogously. $\square$

A direct consequence of the above propositions and Remark 4.13 is that, for both players, winning the game Γ^{cc} is a monotone property with respect to decreasing delay.

Remark 6.4. Let $L \subseteq (\mathbb{B}^2)^\omega$ be an ω-language and $f_0, g_0 : \mathbb{N} \to \mathbb{N}_+$. If Player I wins $\Gamma^{\mathrm{cc}}_{f_0,g_0}(L)$, then he also wins $\Gamma^{\mathrm{cc}}_{f,g}(L)$, for all $f, g : \mathbb{N} \to \mathbb{N}_+$ with $f \sqsubseteq f_0$. Analogously, if Player O wins $\Gamma^{\mathrm{cc}}_{f_0,g_0}(L)$, then she also wins $\Gamma^{\mathrm{cc}}_{f,g}(L)$, for all $f, g : \mathbb{N} \to \mathbb{N}_+$ with $g \sqsubseteq g_0$.

Basically, the following theorem is a consequence of the previous remark. For case 2a, note that there cannot exist functions f, g such that Player O wins $\Gamma^{\mathrm{cc}}_{f,g}(L)$, because, otherwise, it follows from monotonicity that both players win $\Gamma^{\mathrm{cc}}_{\langle 0\rangle,\langle 0\rangle}(L)$; a contradiction. For case 2b, the argument is analogous.

Theorem 6.5. *Let L be an ω-language over $\mathbb{B}^2$. Then precisely one of the following holds:*

1. *Global Determinacy: the game $\Gamma^{\mathrm{cc}}_{f,g}(L)$ is determined, for all $f, g : \mathbb{N} \to \mathbb{N}_+$.*
   a) *Player I wins $\Gamma^{\mathrm{cc}}_{f,g}(L)$, for all f, g.*
   b) *Player O wins $\Gamma^{\mathrm{cc}}_{f,g}(L)$, for all f, g.*
2. *Delay-dependent Determinacy: there exist functions $f_0, g_0, f_1, g_1 : \mathbb{N} \to \mathbb{N}_+$ such that $\Gamma^{\mathrm{cc}}_{f_0,g_0}(L)$ is determined and $\Gamma^{\mathrm{cc}}_{f_1,g_1}(L)$ is non-determined.*
   a) *Player I has a winning strategy in $\Gamma^{\mathrm{cc}}_{f,g}(L)$ for all f, g with $f \sqsubseteq f_0$, and $\Gamma^{\mathrm{cc}}_{f,g}(L)$ is non-determined for all f, g with $f \sqsupseteq f_1$. Player O does not have a winning strategy in $\Gamma^{\mathrm{cc}}_{f,g}(L)$, for all f, g.*
   b) *Player O has a winning strategy in $\Gamma^{\mathrm{cc}}_{f,g}(L)$ for all f, g with $g \sqsubseteq g_0$, and $\Gamma^{\mathrm{cc}}_{f,g}(L)$ is non-determined for all f, g with $g \sqsupseteq g_1$. Player I does not have a winning strategy in $\Gamma^{\mathrm{cc}}_{f,g}(L)$, for all f, g.*
3. *Global Non-Determinacy: $\Gamma^{\mathrm{cc}}_{f,g}(L)$ is non-determined, for all $f, g : \mathbb{N} \to \mathbb{N}_+$.*

6.2 Regular Specifications

From now on, we deal only with ω-regular specifications. We use the results from Chapter 5 to decide, for which functions, which player has a winning strategy in the concurrent delay game. Moreover, we show that one can decide whether there exist delay functions (high enough) such that the game becomes non-determined.

As a start, we reformulate case 2 of Theorem 6.5 such that the delay functions f_0, g_0, f_1, g_1 can be assumed to induce constant delay.

Lemma 6.6. *Let $\mathcal{A}$ be a DPA over $\mathbb{B}^2$ recognizing L, where $\mathcal{A}$ has n states and m colors, and $n' := 2^{(mn)^{2n}}$. Moreover, let $f_0, g_0, f_1, g_1 : \mathbb{N} \to \mathbb{N}_+$ be functions such that $\Gamma^{\mathrm{cc}}_{f_0,g_0}(L)$ is determined and $\Gamma^{\mathrm{cc}}_{f_1,g_1}(L)$ is non-determined. Then precisely one of the following holds:*

1. *There exists a maximal $d_\mathrm{I} \leq 2n'-1$ uniquely determined by L such that Player I wins $\Gamma^{\mathrm{cc}}_{\langle d_\mathrm{I}\rangle,g}(L)$, for all g. Moreover, $\Gamma^{\mathrm{cc}}_{f,g}(L)$ is non-determined for all f, g with $f \sqsupseteq \langle d_\mathrm{I}\rangle$.*
2. *There exists a maximal $d_\mathrm{O} \leq 2n'-1$ uniquely determined by L such that Player O wins $\Gamma^{\mathrm{cc}}_{f,\langle d_\mathrm{O}\rangle}(L)$, for all f. Moreover, $\Gamma^{\mathrm{cc}}_{f,g}(L)$ is non-determined for all f, g with $g \sqsupseteq \langle d_\mathrm{O}\rangle$.*

Proof. Assume that Player I wins $\Gamma^{\mathrm{cc}}_{f_0,g_0}(L)$. Then it follows from Proposition 6.2 that Player I wins $\Gamma_{f_0}(L)$. By Remark 4.13, Player I also wins $\Gamma_{\langle 0\rangle}(L)$ and thus, by Proposition 6.2, he wins $\Gamma^{\mathrm{cc}}_{\langle 0\rangle,g}(L)$ for all g.

Moreover, if $\Gamma^{\mathrm{cc}}_{f_1,g_1}(L)$ is non-determined, then Player I does not win $\Gamma_{f_1}(L)$, by Proposition 6.2. Thus, since $\Gamma_{f_1}(L)$ is determined, Player O wins $\Gamma_{f_1}(L)$. By the results of Chapter 5, she also wins $\Gamma_{\langle d'\rangle}(L)$, for some $d' \leq 2n'-1$, which means that Player I does not win $\Gamma_{\langle d'\rangle}(L)$. Using Proposition 6.2 again, we obtain that Player I does not win $\Gamma^{\mathrm{cc}}_{\langle d'\rangle,g}(L)$, for all g.

Altogether, Player I wins $\Gamma^{\mathrm{cc}}_{\langle 0\rangle,g}(L)$ and does not win $\Gamma^{\mathrm{cc}}_{\langle d'\rangle,g}(L)$, for some $d' \leq 2n'-1$ and all g. Hence, by monotonicity, there must exist a unique constant $d_\mathrm{I} \in [0, 2n'-1]$ such that Player I wins $\Gamma^{\mathrm{cc}}_{\langle d\rangle,g}(L)$ for all $d \leq d_\mathrm{I}$ and all g, and $\Gamma^{\mathrm{cc}}_{\langle d\rangle,g}(L)$ is non-determined for all $d > d_\mathrm{I}$ and all g. Clearly, d_I depends only on L. Thus, item 1 of the lemma holds.

Item 2 holds analogously, assuming that Player O wins $\Gamma^{\mathrm{cc}}_{f_0,g_0}(L)$. □

Theorem 6.7. *Let $\mathcal{A} = (Q, q_0, \delta, c)$ be a DPA over $\mathbb{B}^2$ recognizing L, where $\mathcal{A}$ has n states and m colors, and $n' := 2^{(mn)^{2n}}$. Then, the problem to decide which case of Theorem 6.5 holds is in* 2ExpTime. *If case 2 applies, then one can compute d_I (or d_O) in* 3ExpTime.

Proof. We construct both the semigroup games $\Gamma_{\mathrm{SG}}(\mathcal{A})$ and $\Gamma_{\mathrm{SG}}(\overline{\mathcal{A}}^{\updownarrow})$ (see Section 5.2) and distinguish four different cases.

1. Player I wins both $\Gamma_{\mathrm{SG}}(\mathcal{A})$ and $\Gamma_{\mathrm{SG}}(\overline{\mathcal{A}}^{\updownarrow})$. By Lemmas 5.13 and 5.14, Player I wins $\Gamma_f(L)$ for all f. From Proposition 6.2 it follows that Player I wins $\Gamma^{\mathrm{cc}}_{f,g}(L)$, for all f, g. Analogously (with Proposition 6.3), if Player I wins $\Gamma_{\mathrm{SG}}(\overline{\mathcal{A}}^{\updownarrow})$, then Player O wins $\Gamma^{\mathrm{cc}}_{f,g}(L)$, for all f, g. This yields a contradiction, because at most one player can have a winning strategy in Γ^{cc}; consequently, this case cannot occur. Moreover, we conclude that if Player I wins one of the two semigroup games, then Player O wins the other one (see cases 2 and 3 below).

2. Player I wins $\Gamma_{\mathrm{SG}}(\mathcal{A})$ and Player O wins $\Gamma_{\mathrm{SG}}(\overline{\mathcal{A}}^{\updownarrow})$. By Lemmas 5.13 and 5.14, Player I wins $\Gamma_f(L)$ for all f. From Proposition 6.2 it follows that Player I wins $\Gamma^{\mathrm{cc}}_{f,g}(L)$, for all f, g, which means that case 1a of Theorem 6.5 holds.

3. Player O wins $\Gamma_{\mathrm{SG}}(\mathcal{A})$ and Player I wins $\Gamma_{\mathrm{SG}}(\overline{\mathcal{A}}^{\updownarrow})$. This case is analogous to case 2, yielding that Player O wins $\Gamma^{\mathrm{cc}}_{f,g}(L)$, for all f, g. Thus, it holds case 1b of Theorem 6.5.

4. Player O wins both $\Gamma_{\mathrm{SG}}(\mathcal{A})$ and $\Gamma_{\mathrm{SG}}(\overline{\mathcal{A}}^{\updownarrow})$. Lemma 5.15 yields that Player O wins $\Gamma_{\langle 2n'-1\rangle}(L)$, and Proposition 6.2 implies that Player I does not have a winning strategy in $\Gamma^{\mathrm{cc}}_{\langle 2n'-1\rangle,g}(L)$, for all g. Analogously, we obtain that Player O wins $\Gamma_{\langle 2n'-1\rangle}(\overline{L}^{\updownarrow})$, and Proposition 6.3 implies that Player O does not have a winning strategy in $\Gamma^{\mathrm{cc}}_{f,\langle 2n'-1\rangle}(L)$, for all f. As a consequence, the game $\Gamma^{\mathrm{cc}}_{\langle 2n'-1\rangle,\langle 2n'-1\rangle}(L)$ is non-determined; monotonicity yields that $\Gamma^{\mathrm{cc}}_{f,g}(L)$ is non-determined, for all $f, g \sqsupseteq \langle 2n'-1\rangle$.

   To decide whether it holds item 2 or 3 of Theorem 6.5, we check whether $\Gamma^{\mathrm{cc}}_{\langle 0\rangle,\langle 0\rangle}(L)$ has a winner. For each $\sigma \in \{\mathrm{I}, \mathrm{O}\}$, we consider the game $\mathcal{G}_\sigma$, which is defined as the normal Gale-Stewart game (cf. Section 1.4.3), with the only difference that Player σ begins: in each round, Player σ chooses one bit x and Player $\overline{\sigma}$ answers by one bit y, afterwards. Clearly, $\mathcal{G}_\sigma$ can be modeled as a parity game on a finite graph and we can apply standard techniques to decide its winner [GTW02]. The important observation is that each winning strategy for Player σ in $\Gamma^{\mathrm{cc}}_{\langle 0\rangle,\langle 0\rangle}(L)$ also is a winning strategy for him in $\mathcal{G}_\sigma$, and vice versa. This is due to the fact that, in both games, Player σ has to choose x before knowing about y. For Player $\overline{\sigma}$ there is a difference: whereas in $\mathcal{G}_\sigma$ he knows about x before choosing y, in $\Gamma^{\mathrm{cc}}_{\langle 0\rangle,\langle 0\rangle}(L)$ he knows about x only after choosing y, which is a disadvantage compared to $\mathcal{G}_\sigma$. Altogether, Player $\overline{\sigma}$ wins $\mathcal{G}_\sigma$ if and only if he wins $\Gamma^{\mathrm{cc}}_{\langle 0\rangle,\langle 0\rangle}(L)$ or $\Gamma^{\mathrm{cc}}_{\langle 0\rangle,\langle 0\rangle}(L)$ is non-determined. We can deduce the following.

Remark 6.8. The game $\Gamma^{\mathrm{cc}}_{\langle 0\rangle,\langle 0\rangle}(L)$ is non-determined if and only if Player I wins $\mathcal{G}_{\mathrm{O}}$ and Player O wins $\mathcal{G}_{\mathrm{I}}$.

To that effect, solving $\mathcal{G}_{\mathrm{I}}$ and $\mathcal{G}_{\mathrm{O}}$ we can decide whether it holds case 2a, 2b or 3 (of Theorem 6.5) as follows: if $\Gamma^{\mathrm{cc}}_{\langle 0\rangle,\langle 0\rangle}(L)$ is non-determined, then $\Gamma^{\mathrm{cc}}_{f,g}(L)$ is non-determined for all f, g, by Remark 6.4. Accordingly, it holds case 3 of Theorem 6.5. Otherwise, the winner of $\Gamma^{\mathrm{cc}}_{\langle 0\rangle,\langle 0\rangle}(L)$ determines whether it holds case 2a or 2b.

We have proven that, for regular $L \subseteq (\mathbb{B}^2)^\omega$, it is decidable which case of Theorem 6.5 holds. The construction and solving of the semigroup games $\Gamma_{\mathrm{SG}}(\mathcal{A})$ and $\Gamma_{\mathrm{SG}}(\overline{\mathcal{A}^{\updownarrow}})$ can be done in doubly exponential time, which suffices to deal with cases 1 through 3 of the above case distinction.

For case 4, we need to construct and solve both $\mathcal{G}_{\mathrm{I}}$ and $\mathcal{G}_{\mathrm{O}}$. Each of these two games can be modeled by a parity game on a graph with $3n$ vertices and m colors and, thus, is solvable in time $\mathcal{O}((3n)^m)$. If the solutions to $\mathcal{G}_{\mathrm{I}}$ and $\mathcal{G}_{\mathrm{O}}$ yield that $\Gamma^{\mathrm{cc}}_{\langle 0\rangle,\langle 0\rangle}(L)$ does not have a winner, then we are done. Otherwise, we make use of Lemma 6.6 and compute d_{I} (or d_{O}), analogously to the proof of Theorem 5.24. Let us exemplarily consider the case where Player I wins $\Gamma^{\mathrm{cc}}_{\langle 0\rangle,\langle 0\rangle}(L)$, i.e., we have to compute d_{I}. We execute a binary search through the interval $[0, 2n'-1]$, finding the maximal d such that Player I wins $\Gamma^{\mathrm{cc}}_{\langle d\rangle,\langle 0\rangle}(L)$, in at most $\log n'+2$ iterations. For every considered constant d, we construct the generalization $\mathcal{G}_{\mathrm{I},d}$ of $\mathcal{G}_{\mathrm{I}}$, where in his first move Player I chooses $d+1$ bits, instead of only one bit. (The game $\mathcal{G}_{\mathrm{O},d}$ needs not be considered because we know that Player O does not win $\Gamma^{\mathrm{cc}}_{f,g}(L)$, for all f, g.) $\mathcal{G}_{\mathrm{I},d}$ can be modeled as a parity game on a graph with at most $\mathcal{O}(n2^d)$ vertices and m colors, hence it can be solved in exponential time [Sch07]. Since d is at most doubly exponential, the overall time needed to compute d_{I} is triply exponential. □

Conclusion of Part II

Summary

In the second part of the thesis, we have explored the topological properties of operators that are needed to solve regular conditions. Our motivation has been initiated by the work [BL69] of Büchi and Landweber, who showed that the existence of solutions with delay zero is decidable, and by the work [HL72] of Hosch and Landweber, who extended the former result to bounded-delay operators.

Our Approach. We have considered regular Gale-Stewart games with the notion of a strategy corresponding to the class of continuous operators. One of the players is allowed to delay each of his moves for a finite number of steps, thereby obtaining the possibility to get an unbounded lookahead on the moves of the opponent. In our approach, we have made use of the fact that the behavior of a deterministic parity automaton (recognizing the winning condition) can be described by a finite number of equivalence classes, each of which contains all words which induce the same behavior on the automaton. The set of equivalence classes induces a finite semigroup on the set of all possible inputs to the automaton. On this basis, we have defined a parity game on a finite graph, called semigroup game, in which a move of a player is a behavior of the given parity automaton.

Results. We have shown that the output player wins the semigroup game if and only if she wins the Gale-Stewart game with some finite delay. Thereby, the problem whether a given regular specification is solvable with finite delay turned out to be decidable (in doubly exponential time). Moreover, we have proven that each finite-delay solution can be reduced to one of constant delay where the required constant is at most doubly exponential in the size of the automaton representing the winning condition. This follows from the fact that the length of a shortest representative of any behavior induced by the automaton is at most doubly exponential.

From our results we have derived a generalization of the determinacy of regular conditions (shown by Büchi and Landweber), now provided in Theorem 5.23. In topological terms, our results mean that every regular condition which is solvable by a continuous operator is also solvable by a Lipschitz continuous operator.

We have obtained analogous results for a concurrent setting.

Further Prospects

Lower Bound. We have skipped a proof of a lower bound for the required delay, which obviously is at least linear in the size of the automaton. For example, the condition that the first output bit has to coincide with the k-th input bit (for some fixed k), is recognizable by a DPA with $\mathcal{O}(k)$ states, and it requires a delay of k to be solved.

We believe that there exists no regular condition requiring a delay which asymptotically exceeds the number of states of an automaton recognizing the condition. The reason for this is as follows: if in a regular specification the i-th output letter depends on the first j letters of the input sequence, then an automaton recognizing the specification has to count up to positions i and j; this requires at least $\max\{i, j\}$ states. One idea could be to see whether a representation by logic formulas helps in the issue of finding lower bounds.

Infinite Delay. Note that there are regular conditions which require to know about the whole input sequence, to be able to decide on the first output letter. For example, consider the specification over $\mathbb{B}^2$ where the first output bit should be 1 if the input sequence contains 1 infinitely often, and 0 otherwise. This condition is recognizable by a DPA with five states and three colors. Clearly, Player O does not win with any finite delay.

However, one could think of a game with infinite delay, such that Player I has to give a full ω-word in his first move. We have spared to consider this setting as the question whether there exists a solution with infinite delay can obviously be answered in the following way: if there exists a word α such that, for no β, the pair (α, β) is contained in L, then Player I wins by choosing α. Otherwise, for each α, Player O can build up some β such that $(\alpha, \beta) \in L$, which means that Player O has a winning strategy, because she knows α before she has to choose the first letter of β. The problem to decide which one of the aforementioned situations holds can effectively be solved as follows: first, we project the transitions of the automaton $\mathcal{A}$ on the first component; afterwards, the language of the obtained automaton, say $\mathcal{A}'$, is tested for universality. Clearly, Player O wins with infinite delay if and only if $L(\mathcal{A}')$ is universal.

One may argue that the above notion of infinite delay is inappropriate for algorithmic synthesis. On the one hand, it is not clear how to model it as an infinite game because Player I had to move only once. On the other hand, depending on the kind of word Player I chooses, Player O's winning strategy needs not be finitely representable; therefore, we might not be able to implement it.

However, it is also possible to think of infinite lookahead such that the second player may use information about the first player's sequence up to a partition of the space of sequences into regular sets.

Non-regular Winning Conditions. Whereas Walukiewicz has shown that infinite games with deterministic context-free winning condition are determined with (deterministic) pushdown-strategies [Wal96], neither our results nor the results in [HL72] translate to non-regular winning conditions. Recent work of Fridman et al. has shown that the problem whether a deterministic context-free specification is solvable with finite delay is undecidable [FLZ10].

The proof is accomplished by a reduction from the halting problem for 2-register machines, which is known to be undecidable [SS63]. Player I builds up a sequence of encodings of configurations, each of which is a triple (l, c_1, c_2), where l denotes the current line of the machine and c_1, c_2 are the contents of the counters. Player O verifies whether or not the adversary's choices comply with the transitions of the given machine. She wins if she detects an error in the computation given by Player I, i.e., at a particular position the next but one configuration is not the successor configuration of the next one. The latter condition can be checked by a deterministic pushdown automaton.

The argument for the correctness of the reduction is as follows: if the machine halts, then its computation is finite and of bounded length. Accordingly, there exists a constant d such that, in the game with delay d, Player I has to choose the whole computation (plus a successor configuration of the stop-configuration) in the first round. Since the stop-configuration has no valid successor configuration, Player O can claim incorrectness, and hence wins with delay d. Conversely, if the machine does not halt, then Player I can build up an infinite computation without making a mistake. Accordingly, he wins with each finite delay. Note that the given arguments yield undecidability already for the case where only bounded delay is considered.

Moreover, Fridman et al. have shown in [FLZ10] that there exists a deterministic context-free specification such that Player O wins the corresponding game with finite delay, but she requires a delay which cannot be bounded by some k-fold exponential function, for any fixed k.

Bibliography

[ALW89] Martín Abadi, Leslie Lamport, and Pierre Wolper. Realizable and unrealizable specifications of reactive systems. In Giorgio Ausiello, Mariangiola Dezani-Ciancaglini, and Simona Ronchi Della Rocca, editors, *ICALP*, volume 372 of *LNCS*, pages 1–17. Springer, 1989.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[BL69] J. Richard Büchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the AMS*, 138:295–311, 1969.

[BLV96] Nils Buhrke, Helmut Lescow, and Jens Vöge. Strategy construction in infinite games with streett and rabin chain winning conditions. In Tiziana Margaria and Bernhard Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *LNCS*, pages 207–224. Springer Berlin / Heidelberg, 1996.

[Bra84] Wilfried Brauer. *Automatentheorie: Eine Einführung in die Theorie endlicher Automaten; mit 60 Beispielen und 111 Übungsaufgaben*. Leitfäden und Monographien der Informatik. Teubner, Stuttgart, 1984.

[BSW03] Alexis-Julien Bouquet, Olivier Serre, and Igor Walukiewicz. Pushdown games with unboundedness and regular conditions. In Paritosh K. Pandya and Jaikumar Radhakrishnan, editors, *FSTTCS*, volume 2914 of *LNCS*, pages 88–99. Springer, 2003.

[Cac03] Thierry Cachat. Higher order pushdown automata, the caucal hierarchy of graphs and parity games. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *ICALP*, volume 2719 of *LNCS*, pages 556–569. Springer, 2003.

[Cha06] Krishnendu Chatterjee. Linear Time Algorithm for Weak Parity Games. Technical report, EECS Department, University of California, Berkeley, 2006.

[Chu57] Alonzo Church. Applications of recursive arithmetic to the problem of circuit synthesis. In *Summaries of the Summer Institute of Symbolic Logic*, volume 1, pages 3–50. Cornell University, Ithaca, New York, 1957.

[Chu63] Alonzo Church. Logic, arithmetic, and automata. In *Proceedings of the International Congress of Mathematicians, 1962*, pages 23–35. Institute Mittag-Leffler, Djursholm, Sweden, 1963.

[CMJ04] Krishnendu Chatterjee, Rupak Majumdar, and Marcin Jurdzinski. On nash equilibria in stochastic games. In Jerzy Marcinkowski and Andrzej Tarlecki, editors, *CSL*, volume 3210 of *LNCS*, pages 26–40. Springer, 2004.

[dAH00] Luca de Alfaro and Thomas A. Henzinger. Concurrent Omega-Regular Games. In *15th Symposium on Logic in Computer Science (LICS' 00)*, pages 141–156, Washington - Brussels - Tokyo, 2000. IEEE.

[dAHM00] Luca de Alfaro, Thomas A. Henzinger, and Freddy Y. C. Mang. The control of synchronous systems. In Catuscia Palamidessi, editor, *CONCUR 2000 – Concurrency Theory (11th CONCUR'2000)*, volume 1877 of *LNCS*, pages 458–473. Springer-Verlag (New York), Pennsylvania State University, University Park, PA, USA, 2000.

[dAHM01] Luca de Alfaro, Thomas A. Henzinger, and Freddy Y. C. Mang. The control of synchronous systems, part II. In Kim G. Larsen and Mogens Nielsen, editors, *CONCUR 2001 – Concurrency Theory (12st CONCUR'01)*, volume 2154 of *LNCS*, pages 566–582. Springer-Verlag (New York), Aalborg, Denmark, 2001.

[DJW97] Stefan Dziembowski, Marcin Jurdziński, and Igor Walukiewicz. How Much Memory is Needed to Win Infinite Games? In *Proceedings of the 12th LICS*, pages 99–110, Washington - Brussels - Tokyo, 1997. IEEE.

[EH00] Kousha Etessami and Gerard J. Holzmann. Optimizing Büchi Automata. In Catuscia Palamidessi, editor, *CONCUR 2000 – Concurrency Theory*, volume 1877 of *LNCS*, pages 153–168. Springer Berlin/Heidelberg, 2000.

[EJ91] E. Allen Emerson and Charanjit S. Jutla. Tree Automata, Mu-Calculus and Determinacy (Extended Abstract). In *32nd Annual*

Symposium on Foundations of Computer Science, pages 368–377, San Juan, Puerto Rico, 1991. IEEE.

[EL85] E. Allen Emerson and Chin-Laung Lei. Modalities for model checking: Branching time strikes back. In *Conference Record of the Twelfth Annual ACM Symposium on Principles of Programming Languages*, pages 84–96, New Orleans, Louisiana, January 13–16, 1985. ACM SIGACT-SIGPLAN, ACM Press. Extended abstract.

[EM69] Shimon Even and Albert R. Meyer. Sequential boolean equations. *IEEE Transactions on Computers*, C-18(3):230–240, 1969.

[EWS05] Kousha Etessami, Thomas Wilke, and Rebecca A. Schuller. Fair Simulation Relations, Parity Games, and State Space Reduction for Büchi Automata. *SIAM Journal on Computing*, 34(5):1159–1175, 2005.

[FLZ10] Wladimir Fridman, Christof Löding, and Martin Zimmermann. Degrees of lookahead in context-free infinite games. Technical Report AIB-2010-20, RWTH Aachen University, December 2010.

[Fri05] Carsten Fritz. *Simulation-Based Simplification of omega-Automata*. PhD thesis, Christian-Albrechts-Universität zu Kiel, 2005.

[Fri09] Oliver Friedmann. An exponential lower bound for the parity game strategy improvement algorithm as we know it. In *LICS*, pages 145–156. IEEE Computer Society, 2009.

[FW02] Carsten Fritz and Thomas Wilke. State space reductions for alternating büchi automata quotienting by simulation equivalences. In Manindra Agrawal and Anil Seth, editors, *FST TCS 2002: Foundations of Software Technology and Theoretical Computer Science*, volume 2556 of *Lecture Notes in Computer Science*, pages 157–168. Springer Berlin / Heidelberg, 2002.

[FW06] Carsten Fritz and Thomas Wilke. Simulation Relations for Alternating Parity Automata and Parity Games. In *Proceedings of the 10th DLT*, volume 4036 of *LNCS*, pages 59–70. Springer, 2006.

[GH82] Yuri Gurevich and Leo Harrington. Trees, Automata and Games. In *Proceedings of the 14th STOC*, pages 60–65, San Francisco, CA, 1982.

[GH11] Marcus Gelderie and Michael Holtmann. Memory reduction via delayed simulation. In Johannes Reich and Bernd Finkbeiner, editors, *International Workshop on Interactions, Games and Protocols (iWIGP), Saarbrücken, Germany*, volume 50 of *EPTCS*, pages 46–60, 2011.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Co., San Francisco, CA, 1979.

[Gri73] David Gries. Describing an Algorithm by Hopcroft. *Acta Informatica*, 2:97–109, 1973.

[GS53] David Gale and Frank M. Stewart. Infinite games with perfect information. *Annals of Mathematical Studies*, 28:245–266, 1953.

[GTW02] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics and Infinite Games*, volume 2500 of *LNCS*. Springer, 2002.

[GW06] Erich Grädel and Igor Walukiewicz. Positional determinacy of games with infinitely many priorities. *Logical Methods in Computer Science*, 2(4), 2006.

[HD05] Paul Hunter and Anuj Dawar. Complexity bounds for regular games. In Joanna Jedrzejowicz and Andrzej Szepietowski, editors, *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science 2005, MFCS 2005*, volume 3618 of *LNCS*, pages 495–506. Springer, 2005.

[HHK95] Monika R. Henzinger, Thomas A. Henzinger, and Peter W. Kopke. Computing simulations on finite and infinite graphs. In IEEE, editor, *36th Annual Symposium on Foundations of Computer Science: October 23–25, 1995, Milwaukee, Wisconsin*, pages 453–462, pub-IEEE:adr, 1995. IEEE Computer Society Press.

[HKR97] Thomas A. Henzinger, Orna Kupferman, and Sriram K. Rajamani. Fair simulation. In Antoni Mazurkiewicz and Józef Winkowski, editors, *CONCUR '97: Concurrency Theory, 8th International Conference*, volume 1243 of *LNCS*, pages 273–287, Warsaw, Poland, 1997. Springer-Verlag.

[HKT10] Michael Holtmann, Łukasz Kaiser, and Wolfgang Thomas. Degrees of Lookahead in Regular Infinite Games. In Luke Ong, editor, *Foundations of Software Science and Computational Structures*, volume 6014 of *LNCS*, pages 252–266. Springer, 2010.

[HL72] Frederick A. Hosch and Lawrence H. Landweber. Finite delay solutions for sequential conditions. In M. Nivat, editor, *Automata, Languages and Programming*, pages 45–60, Paris, France, 1972. North-Holland, Amsterdam.

[HL07] Michael Holtmann and Christof Löding. Memory Reduction for Strategies in Infinite Games. In Jan Holub and Jan Zdárek, editors, *CIAA*, volume 4783 of *LNCS*, pages 253–264. Springer, 2007.

[Hol07] Michael Holtmann. Memory Reduction for Strategies in Infinite Games. Diploma Thesis (revised version), RWTH Aachen, 2007.

[Hop71] John E. Hopcroft. An $n \log n$-Algorithm for Minimizing States in a Finite Automaton. Technical report, Stanford University, Department of Computer Science, 1971.

[Hor05] Florian Horn. Streett Games on Finite Graphs. *GDV*, 2005.

[Hor08] Florian Horn. Explicit Muller games are PTIME. In Ramesh Hariharan, Madhavan Mukund, and V. Vinay, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2008)*, volume 2 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 235–243, Dagstuhl, Germany, 2008. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[JKH02] Marcin Jurdziński, Orna Kupferman, and Thomas A. Henzinger. Trading probability for fairness. In Julian C. Bradfield, editor, *CSL*, volume 2471 of *LNCS*, pages 292–305. Springer, 2002.

[JPZ06] Marcin Jurdziński, Mike Paterson, and Uri Zwick. A deterministic subexponential algorithm for solving parity games. In *SODA*, pages 117–123. ACM-SIAM, 2006.

[Jur98] Marcin Jurdziński. Deciding the winner in parity games is in $\mathrm{UP} \cap \mathrm{co-UP}$. *Information Processing Letters*, 68(3):119–124, 1998.

[Jur00] Marcin Jurdziński. Small progress measures for solving parity games. In Horst Reichel and Sophie Tison, editors, *STACS*, volume 1770 of *LNCS*, pages 290–301. Springer, 2000.

[Koh70] Zvi Kohavi. *Switching and Finite Automata Theory*. McGraw-Hill, New York, 1970.

[Kuč11] Antonín Kučera. Turn-based stochastic games. In Krzysztof R. Apt and Erich Grädel, editors, *Lectures in Game Theory for Computer Scientists*, pages 146–184. Cambridge University Press, 2011.

[Löd01] Christof Löding. Efficient minimization of deterministic weak ω-automata. *IPL*, 79:105–109, 2001.

[McN65] Robert McNaughton. Finite-state infinite games. Technical report, Project MAC, Massachusetts Institute of Technology, USA, 1965.

[McN66] Robert McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9(5):521–530, 1966.

[McN93] Robert McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65(2):149–184, 1993.

[Mea55] George H. Mealy. A method for synthesizing sequential circuits. *Bell System Technical Journal*, 34(5):1045–1079, 1955.

[Mos80] Yiannis N. Moschovakis. *Descriptive Set Theory*, volume 100 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Company, 1980.

[Mos91] Andrzej W. Mostowski. Games with forbidden positions. Technical Report 78, University of Gdańsk, 1991.

[MP95] Zohar Manna and Amir Pnueli. *Temporal Verification of Reactive Systems - Safety -*. Springer-Verlag, New York, 1995.

[MS95] David E. Muller and Paul E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141(1-2):69–107, 1995.

[Mul63] David E. Muller. Infinite sequences and finite machines. In *Proceedings of the Fourth Annual Symposium on Switching Circuit Theory and Logical Design*, pages 3–16, Chicago, Illinois, 1963. IEEE.

[Pin95] Jean-Éric Pin. Finite semigroups and recognizable languages: An introduction, 1995.

[Pöt10] Frank Pöttgen. Unendliche Zweipersonenspiele mit Verzögerter Information. Diploma Thesis, RWTH Aachen, 2010.

[PP95] Dominique Perrin and Jean-Éric Pin. Semigroups and automata on infinite words. In J. Fountain, editor, *NATO Advanced Study Institute Semigroups, Formal Language and Groups*, pages 49–72. Kluwer academic publishers, 1995.

[PP04] Dominique Perrin and Jean-Éric Pin. *Infinite words*, volume 141 of *Pure and Applied Mathematics*. Elsevier, Amsterdam, 2004.

[PR89] Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Programming Languages*, pages 179–190. ACM Press, 1989.

[Sch07] Sven Schewe. Solving parity games in big steps. In Vikraman Arvind and Sanjiva Prasad, editors, *FSTTCS*, volume 4855 of *LNCS*, pages 449–460. Springer, 2007.

[Sed91] Robert Sedgewick. *Algorithmen*. Addison-Wesley Publishing Company, Reading, MA, 1991.

[SS63] John C. Shepherdson and Howard E. Sturgis. Computability of recursive functions. *Journal of the ACM*, 10:217–255, 1963.

[SW74] Ludwig Staiger and Klaus Wagner. Automatentheoretische und automatenfreie Charakterisierungen topologischer Klassen regulärer Folgenmengen. *Elektronische Informationsverarbeitung und Kybernetik*, 10(7):379–392, 1974.

[TB73] Boris A. Trakhtenbrot and Janis M. Barzdin. *Finite Automata, Behavior and Synthesis*. North Holland, Amsterdam, 1973.

[Tho95] Wolfgang Thomas. On the synthesis of strategies in infinite games. In *Proceedings of the 12th STACS*, volume 900 of *LNCS*, pages 1–13, Munich, Germany, 1995. Springer.

[Tho97] Wolfgang Thomas. Languages, automata and logic. In A. Salomaa and G. Rozenberg, editors, *Handbook of Formal Languages*, volume 3, Beyond Words. Springer, Berlin, 1997.

[Tho02] Wolfgang Thomas. Infinite Games and Verification. In *Proceedings of the 14th CAV*, volume 2404 of *LNCS*, pages 58–64. Springer-Verlag, 2002.

[TL93] Wolfgang Thomas and Helmut Lescow. Logical specifications of infinite computations. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *REX School/Symposium*, volume 803 of *LNCS*, pages 583–621. Springer, 1993.

[VJ00] Jens Vöge and Marcin Jurdziński. A discrete strategy improvement algorithm for solving parity games. In E. Allen Emerson and A. Prasad Sistla, editors, *CAV*, volume 1855 of *LNCS*, pages 202–215. Springer, 2000.

[Wal96] Igor Walukiewicz. Pushdown processes: Games and model checking. In Rajeev Alur and Thomas A. Henzinger, editors, *CAV*, volume 1102 of *LNCS*, pages 62–74. Springer, 1996.

[Wal03] Nico Wallmeier. Symbolische Synthese zustandsbasierter reaktiver Programme. Diplomarbeit, RWTH Aachen, 2003.

[Wal08] Nico Wallmeier. *Strategien in unendlichen Spielen mit Liveness-Gewinnbedingungen: Syntheseverfahren, Optimierung und Implementierung*. PhD thesis, RWTH Aachen, 2008.

[WHT03] Nico Wallmeier, Patrick Hütten, and Wolfgang Thomas. Symbolic Synthesis of Finite-State Controllers for Request-Response Specifications. In *Proceedings of the 8th CIAA*, volume 2759 of *LNCS*, pages 11–22. Springer, 2003.

[Zie98] Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1-2):135–183, 1998.

Index