Games with perfect information

Lecture notes

June 3, 2019

Sebastian Muskalla

TU Braunschweig

Summer term 2019

Preface

These are the lecture notes accompanying the course "Games with perfect information" taught at TU Braunschweig in the summer term of 2019 (and previously in the summer terms of 2017 and 2018).

I cannot guarantee the correctness of these notes. In case you spot a bug, please send me a mail: `s.muskalla@tu-bs.de`.

I would like to thank Lea, Pascal, Peter, Nora, Jonas, Patrick, Alexander, and all other people that have provided feedback and helped fixing errors in these notes. I am grateful for their help!

Sebastian Muskalla

Braunschweig, June 3, 2019

Contents

Literature

Unfortunately, it seems that there is no single book containing the contents of this lecture.

- There is a plethora of books on game theory, but they mostly study games with imperfect information (which are of interest for economic science). These books usually treat games with perfect information hardly or not at all.
- There are books on perfect-information games that consider them from a purely mathematical perspective, i.e. with an emphasis on theoretical concepts like determinacy and without caring about the algorithmics.
- Parts of the lecture can be found, for example, in books on automata theory, where certain games are introduced as tools to obtain automata-theoretic results. In contrast to this approach, we will focus on game-theoretic results and see their automata-theoretic consequences as applications.

Therefore, I have to refer the reader to a collection of books and papers for the different types of games considered in this lecture. The later sections will contain references to the books and papers that I used to prepare the corresponding lecture. A full list of references can be found at the end of this document.

One should note that the basic definitions, e.g. those of games and plays, differ between different books and papers. For example, in parts of the literature, games are deadlock-free by definition, while we try to avoid making such an assumption. These differences can usually be overcome by minor tweaking.

Other people have taught lectures on games with perfect information whose syllabus overlaps with the one of this lecture. In particular, I want to refer the reader to the lecture notes for a lecture on games given by Martin Zimmermann at the University of Saarland [ZKW].

Part I. Introduction & preliminaries

Contents

1. Introduction

Games, in particular board games, have been a recreational activity for humans for thousands of years. While this fact alone might justify that they are studied in science, one may ask: Why exactly do theoretic computer scientists study board games? A second question that may arise when looking at the title of this lecture is what distinguishes games *with perfect information* from other types of games.

Perfect vs. imperfect information

Let us first answer the second question, then the first. Most games that are played by humans are actually not perfect-information games: In some games, a part of the information is only visible to one of the players, e.g. in Battleships. In others, randomness plays a role, e.g. in Mensch ärgere dich nicht. Most card games, e.g. Poker, combine both: Initially, the cards are shuffled randomly, and later, each player has a set of cards on her hand that is not visible to the other players.

It turns out that randomness can usually be modeled by "hidden information". Thus, all such games are called games with **imperfect information**. These games are widely studied in science, in particular in economic science. The players of a game can model companies and the rules of the game model a market, and thus finding an optimal way to play the game corresponds to finding an optimal behavior for a company in a certain market situation.

The concepts and methods used to study games with imperfect information differ widely from the ones used to study games with perfect information. Therefore, the presentation of games with imperfect information in this lecture will be limited to this paragraph. We present the most famous (and most simple) example of a game with imperfect information that is studied in science. The rules of **prisoner's dilemma** are as follows: Two criminals are caught after a robbery by the police and interrogated separately, with no means of communication. If both remain silent, they can only be convicted for a lesser crime, and have to serve 2 years in prison each. The prosecutor makes them an offer: If one of them confesses the crime (and thereby betrays the other), the traitor only has to serve one year in prison, while the other criminal can be convicted for robbery and has to go to prison for 4 years. The catch is that if both confess, both serve 4 years in prison. Obviously, the sum of the years in prison is minimized if both stay silent, then they have to serve 4 years in total. This value is usually called the **social optimum**. This solution does not take selfishness into account: One could argue that the optimal solution is for both to confess and betray their partner: They now serve 8 years in total, but none of the players can improve their personal situation by changing their behavior. Such a situation is called **Nash equilibrium**, and finding such equilibria is one of the goals of the studies that are made. The factor between Nash equilibrium and social optimum, $\frac{8}{4} = 2$ in our case, is called the **price of anarchy**. These concept can for example be applied to study traffic flow. One can show that under the assumption of the drivers being selfish, there are situations in which the travel time decreases for all drivers if a road is closed.

Let us turn back to games with perfect information. We say that a game is a **perfect-information game** if both players know the rules of the game (i.e. the possible states in which the game be in, and the moves that lead from one state to another), and whenever it is their turn, they know the current state and the full history, i.e. all states in which the game has been before. Among real life board games, many games in which no randomness is involved belong to this class, e.g. Chess and Go. Those two are actually simple examples: We will see in Section 4 that in principle, Chess and Go are easy to solve using a known algorithm. The only thing that prevents us from actually doing so is the huge number of possible states that cannot be handled by modern computers. (In fact, this will probably stay this way in the foreseeable future.) In principle, we can consider games that are far more complicated, because they are infinite: The plays might be of infinite length, the number of possible states can be infinite, or both.

Games with perfect information are special because they allow a reasoning of a special shape: Whenever a player has to pick a move, the consequences of each possible choice are clear to the players, e.g. which choices the opponent has in the next move. More formally, for each given initial position, the tree of all possible plays that can unfold when the game is played from the given position is known in principle. (But it may be infinite or at least very large.)

Examples of games with perfect information

To answer the first question and to motivate why such games are of interest for computer scientists, we consider three examples.

The first example is that games naturally occur whenever decisions in a system are made by several separate entities. In automata theory, non-determinism is often considered (e.g. in the form of NFAs, non-deterministic finite automata), but it is usually assumed to be either completely controllable (e.g. “Is state p reachable from state q?”, or, to highlight the contribution of non-determinism better, “Can the non-determinism be resolved such that we reach p from q?”), or to be completely uncontrollable (e.g. “Is state *p* unreachable from *q*, no matter how non-determinism is resolved?"). It is a natural extension to consider several types of non-determinism, say one controllable and one uncontrollable type. We then ask whether we can instantiate the controllable non-determinism such that a certain property holds, no matter how the uncontrollable non-determinism is resolved. Such a scenario can be seen as a two-player game, where each player represents one type of non-determinism, the desired property corresponds to the winning condition of the game, and the question is now whether one player can enforce that she wins the game, no matter how the other player acts.

This situation occurs for example in **synthesis**. In contrast to verification, where we want to check whether the run-time behavior of a program satisfies a given specification (which means that we have either no or just one type of non-determinism), we now have a program template (a program with "holes") and a specification. Here we want to know whether we can instantiate the template such that the resulting program satisfies the specification. The choices when instantiating the template form one type of non-determinism resp. one of the players, the environment in which the program should be executed in represents another type of non-determinism resp. the other player.

As a second example, games can be used as a powerful tool to obtain new theoretic results. **Rabin's tree theorem** essentially states that the class of tree-languages acceptable by a certain type of automata is closed under complement. It is a highly non-trivial result, and its easiest proof is using parity games as a tool. The idea is to see the branching of a tree as another form of non-determinism (in addition to the non-determinism from the automaton). This allows us to see the acceptance problem for these tree automata ("Does the given automaton accept a given tree?") as a game, in which one player picks the moves of the automaton, and the other player picks the branch of the tree on which the automaton should run. The *positional determinacy of parity games*, a deep result from game theory, states that exactly one of the players can enforce that she wins the game, and in fact do so in a very special way, via a so-called *uniform positional winning strategy*. On the trees not in the language of the automaton, the player representing the automaton cannot win the game. Consequently, the other player has a uniform positional winning strategy for these trees. This strategy can now be encoded into an automaton that will by construction accept the complement language of the original automaton, which proves the result.

The third example can be seen as a combination of the concepts in the first two examples. Verifying a non-deterministic system against a specification that is given by logical formula can be seen as a game: Existential quantifiers in the formula means that there has to be a move of the system such that the subsystem reached by the move satisfies the inner condition. We model this as a player in a game that should select the correct move. Universal quantifiers mean that all subsystems that can be reached by a move should satisfy some property. This is modeled by having a second player that can select a move of his choice to which the first player has to react. The verification question can now be answered by solving the game.

Altogether, we see that whenever multiple entities make decisions that influence the run of a system, we can model the system as a game in which the entities are the players. This is even true when the entities are initially not apparent, but rather are hidden, e.g. in the form of branching of trees, or the evaluation semantics of logical formulas. For many settings that originate in theoretic computer science and its subfields like verification and automata theory, games with perfect information have been successfully used as a suitable model. This enables us to use results from game theory to obtain deep results in these fields.

What it means to "solve" a game

When we talk about solving a game, what do we actually mean? Solving a game means essentially determining the winner of the game. The winner of one concrete play is determined by the winning condition of the game, and thus easy to find. To be the winner of the whole game, a player has to be able to enforce the winning condition to hold in all plays, no matter how the other player acts.

The questions that we are usually asking are the following:

- **Determinacy**: Is there a winner? (This may sound counter-intuitive, but there are games in which there is no winner, although "draw" is not a possible outcome of a play.)

- **Decidability/Computability**: Is there an algorithm (and can we explicitly implement it) that computes the winner?

- **Strategies**: How does the winner have to play to ensure that she does indeed win a play? How can such a *strategy* be implemented such that executing it uses a minimal amount of space and computation time?

Structure of the lecture

The lecture is structured in four parts.

In the first part, we start by considering Nim, a very simple game with perfect information. We then move on and define the basic notations needed in the rest of the lecture: games on graphs, plays, winning conditions and strategies.

In the second part, we consider various types of winning conditions for games on graphs. We start with simple reachability conditions and continue with conditions that work on plays of infinite length, like Büchi, parity, and Muller conditions. Although parts of the theory also work for games on infinite graphs, our focus is on finite graphs as for them, the theory immediately gives rise to algorithms that allows us to compute the winner of the game. We conclude the part by considering games that are not about winning or losing, but about optimizing the payoff (which is a number associated to a play of the game). We study zero-sum games of bounded length with arbitrary payoff functions and mean payoff games in which the payoff is some sort of average value of an infinite play.

We then turn towards studying games on infinite graphs in Part III. We will see that if we do not restrict the game arena and the winning condition, we might obtain games that are undetermined: Although each play has a winner, none of the players has a systematic way of winning. We continue with games whose underlying graph is infinite, but has a finite representation by an automaton. Such games have a winner, and we have the hope that we are able to compute it by working on the finite represenation. Deciding the winner algorithmically will of course not work for automata models for which verification problems are undecidable, like Turing machines and counter machines. Surprisingly, the problem remains undecidable if we restrict counter machines to counter nets, for which verification problems like control state reachability are decidable. In contrast to this result, pushdown games, games on the configuration graphs of pushdown automata, can be decided. We conclude the part by briefly mentioning the Borel determinacy theorem and the resulting Borel hierarchy of winning conditions for which the associates games are guaranteed to be decidable.

As mentioned earlier, game theory has numerous applications. In the course of the lecture, we study two of them, both bundled together in these notes in the form of the fourth part. As a practical application, we see that reachability games can be used to model online scheduling problems. To this end, the tasks that are generated at runtime are seen as one player and the scheduler that should be constructed as the other. A theoretic application of game theory is the above-mentioned Rabin's tree theorem from automata theory which we will state and prove.

Further reading

There are a lot of topics in the research on games with perfect information that are not covered in this lecture. The content of this lecture has hopefully laid the foundation for the interested reader to explore these topics in self-study. We point out a few possible directions and give corresponding references.

- **Algorithmics of parity games on finite graphs:**
  There is active research on finding algorithms for solving parity games. In Section 6, we already mentioned the breakthrough result [Cal+17; JL17] that parity games can be solved by an algorithm that is quasi-polynomial and only exponential in the highest priority. Whether solving parity games is a problem in P remains an open problem.

- **Algorithmics for pushdown games:**
  Walukiewicz's reduction which we discussed in Section 12 shows that parity games on pushdown automata can be decided. However, the resulting algorithm is not suitable for practical usage (although it has the optimal time complexity). There are different techniques for solving various types of Pushdown games that work e.g. by saturating automata[Cac02] or by computing the least solution to a system of equations [HMM16]. Parity games can be turned into safety games by adding a counter (with bounded value) to the control state (see e.g. [FZ12]). In the case of Pushdown games, this even gives a polynomial-time reduction [Hag+18].

- **Higher order pushdown games:**
  Walukiewicz's reduction and some of the other techniques for solving pushdown games can be extended to work on larger classes of systems. Namely, they work for higher levels of the pushdown hierarchy: for higher-order recursion schemes and for higher-order (collapsible) pushdown automata [CW07; HMM17].

- **Game semantics:**
  We have discussed in Section 15 the correspondence between logics and automata, and that algorithmic problems for the latter can be dealt with by solving games. A more direct correspondence is given by the game semantics for certain kinds of logics. For example, the problem of model checking a μ-calculus formula on a system usually corresponds to solving parity games on the systems [Wal01; KO09].

- **Determinacy:**
  A line of studies that is more oriented towards pure mathematics is trying to find

sufficient conditions for the determinacy of infinite games. The big result in this area is the Borel determinacy theorem [Mar75; Mar82] which we have stated but not proven in Section 13.

2. Nim – A warm-up

Before formally introducing the basic definitions, we will work on a toy example. It is of no practical use, but a very famous example of a perfect information game, and one of the first games that have been implemented on a computer. When doing the general theory later, we will eventually see that many steps of the general solutions for games corresponds to the steps that we take in the following to solve the example.

Sources
The content of this section is based on Roland Meyer's notes on the topic.
They can be found here:
tcs.cs.tu-bs.de/documents/ComplexityTheory_WS_20152016/landnl.pdf

Nim

2.1 Definition: Nim
The state of a game of **Nim** is given by a list of piles, each containing a (non-negative) number of coins.

During a play of the game, the players take turns alternately. In each turn, the active player has to select a non-empty pile, and take coins from this pile. She has to take at least one coin, but other than that, she may take arbitrarily many coins, up to the whole pile.

The player that takes the very last coin such that all piles are empty after the move, wins the play of the game.

2.2 Example
Consider a state of a game of Nim that has three piles, two consisting of two coins each, one consisting of just one coin. In the following, we write states as tuples, e.g. as $(2, 2, 1)$. Assume the first player makes takes two coins from the first pile, resulting in state $(0, 2, 1)$. The second player now takes the whole second pile, resulting in $(0, 0, 1)$ and thus enabling the first player to win the play of the game by taking the very last coin.

We write plays as a sequences of transitions between states, e.g. as

$$(2, 2, 1) \xrightarrow{\text{player 1}} (0, 2, 1) \xrightarrow{\text{p2}} (0, 0, 1) \xrightarrow{\text{p1}} (0, 0, 0)\,.$$

So we have seen that this concrete play ends with a win for player 1. Is the fact that player 1 has won an inherent property of the initial position $(2, 2, 1)$ or could player 2 have won by playing more cleverly?

Given some fixed initial position $(c_1, \ldots, c_k)$ (i.e. k piles of coins, where each pile i consists of c_i coins), we would like to check which player can enforce a win, and how she has to play to do this.

One could use the fact that each play of Nim has bounded length: Since each player has to take at least one coin whenever it is her turn, the play consists of at most

$$C = \sum c_i = c_1 + \ldots + c_k$$

moves. Furthermore, in each state, there are only up to C possible moves. Combining these insights, we obtain that all possible plays can be arranged in a tree of height at most C and of out-degree at most C, i.e. a tree with at most C^C nodes.

We could explicitly construct the tree and do the following procedure to check whether player 1 can win:

1. Mark all occurrences of the state $(0, \ldots, 0)$ in which player 1 took the last turn as winning.

2. Mark all states in which player 2 *has to* move to a winning state as winning.

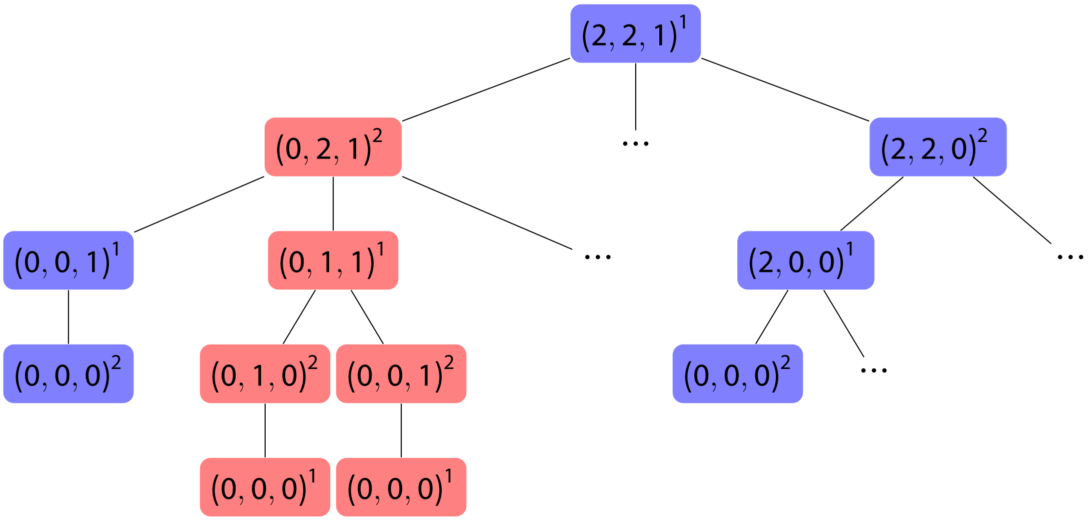
3. Mark all states in which player 1 *can* move to a winning state as winning.

Now repeat steps 2. and 3. until no new states are marked as winning anymore. Whenever the play reaches a winning state, player 1 can win by picking a move that again leads to a winning state whenever it is her turn. The manner in which the states were marked ensures that player 2 will never have a move to reach a state that is not winning. A play played like this will end in a node $(0, \ldots, 0)$ in which player 1 did the last move, and is thus won by player 1.

A similar argumentation can be used to show that whenever a state is not winning, player 2 can ensure that the not-winning property is maintained, and she wins the play of the game.

Checking which player is the winner of the game for a given initial state now can be done by constructing and marking the tree of plays and then checking whether its root note (corresponding to the initial state) is winning.

2.3 Example
We show a part of the tree of plays for the initial state $(2, 2, 1)^1$. Here, the superscripts (e.g. ^{1}) denotes which player has to make the next move. In the base case, states $(0, 0, 0)^2$ are winning for player 1. Winning nodes have a blue, losing nodes have a red background.



The algorithm works, but it has two severe disadvantages: Firstly, it needs to build a tree that is exponential in the size of the initial position. (To be precise: Exponential even in the unary encoding of the numbers!) Secondly, it has to be rerun for every initial position.

Bouton's theorem

We would prefer an algorithm that identifies whether a state is winning without explicitly building the tree.

In the following, we will use the fact that Nim is an **impartial** game: The tuple $(c_1, \ldots, c_k)$ representing the current state uniquely determines all possible moves, and it does not matter which player is currently moving. We will give a condition that is fulfilled if and only if the **active** player, i.e. the player whose turn it is, wins the play.

The desired algorithm was first presented by Bouton in 1901 [Bou01]. The condition is dependent on a property of a binary representation of the c_i, defined as follows.

2.4 Definition: Nim sum
Let $(c_1, \ldots, c_k)$ be a state of a Nim play. We consider a binary, most significant bit first representation of the c_i.

Let j_{max} be the length of the binary representation of the greatest c_i. Let $c_{ij} \in \{0, 1\}$ for $i \in \{1, \ldots, k\}, j \in \{1, \ldots, j_{max}\}$ be the j^{th} bit of the binary representation of c_i.

The **Nim sum** $\mathrm{Nim}\Sigma(c_1, \ldots, c_k)$ of $(c_1, \ldots, c_k)$ is a vector in $\mathbb{N}^{j_{max}}$ such that the j^{th} component is the sum of the j^{th} bits of the binary representations of the c_i, i.e.

$$\mathrm{Nim}\Sigma(c_1, \ldots, c_k)_j = \sum_{i=1}^{k} c_{ij} .$$

We call a state $(c_1, \ldots, c_k)$ **balanced** if every component of $\mathrm{Nim}\Sigma(c_1, \ldots, c_k)$ is even.

2.5 Example

The Nim sum of (2,2,1) is unbalanced.

c_i	c_{i1}	c_{i2}
$c_1 = 2_{10} =$	1	0
$c_2 = 2_{10} =$	1	0
$c_3 = 1_{10} =$	0	1
$\mathrm{Nim}\Sigma =$	2	1

2.6 Theorem: Solving Nim (Bouton 1901 [Bou01])

The active player can enforce that she wins from a state $(c_1, \ldots, c_k)$ if and only if $(c_1, \ldots, c_k)$ is not balanced.

Crucial to the proof of the theorem will be the following three lemmata.

2.7 Lemma

Let $(c_1, \ldots, c_k)$ be a balanced state. There is no move from this state to $(0, \ldots, 0)$.

Proof:

If the position is $(0, \ldots, 0)$, there is no move, in particular no move to $(0, \ldots, 0)$.

Assume there is at least one c_i that is not equal to 0, say c_{i_0}. We prove that there is some index $i_1 \neq i_0$ such that $c_{i_1} \neq 0$:

Towards a contradiction, assume we have $c_i = 0$ for all $i \neq i_0$. As a result, we have $c_{ij} = 0$ for all $i \neq i_0$ and all j. Since $c_{i_0} \neq 0$, there is at least one j, say j_0, such that $c_{i_0 j_0} = 1$. Then we have

$$\mathrm{Nim}\Sigma(c_1, \ldots, c_k)_{j_0} = \sum_{i=1}^{k} c_{ij} = 0 + c_{i_0 j_0} = 1 .$$

This would mean that the Nim sum is not balanced, a contradiction.

Now we know that there are two piles on which at least one coin is present. Since in the next move, the active player can empty at most one of the piles, she will not be able to reach state $(0, \dots, 0)$: Coins remain on at least one pile. ■

2.8 Lemma
Let $(c_1, \dots, c_k)$ be a balanced state. Every successor state (i.e. a state to which we can go with one single move) is unbalanced.

Proof:
If the position is $(0, \dots, 0)$, there is nothing to show since there is no successor.

Assume that $c_i \neq 0$ for some i, and consider an arbitrary successor state $(c'_1, \dots, c'_k)$. When doing a move, exactly one of the c_i is changed, say c_{i_0}. Thus, at least one bit of the binary representation of this c_{i_0} is changed, i.e. there is j_0 such that $c'_{i_0 j_0} \neq c_{i_0 j_0}$.

Now consider the Nim sum of the successor state. It is easy to see that if $\text{Nim}\Sigma(c_1, \dots, c_k)_{j_0}$ was even, then $\text{Nim}\Sigma(c'_1, \dots, c'_k)_{j_0}$ is now odd: $c_{i_0 j_0}$ and $c'_{i_0 j_0}$ differ by one, and c_{ij_0} is unchanged for all $i \neq i_0$. This means that the new Nim sum is not balanced. ■

Note that Lemma 2.8 in fact implies Lemma 2.7. We chose to present them separately for didactic reasons.

2.9 Lemma
Let $(c_1, \dots, c_k)$ be a unbalanced state. There is a successor state (i.e. a state to which we can go with one single move) that is balanced.

Proof: Exercise 2.13. ■

Now we are ready to give to prove the theorem.

Proof of Theorem 2.6:
For one direction of the proof, assume that the initial position $(c_1, \dots, c_k)$ is not balanced. We present a winning strategy for the active player, i.e. a systematic way of playing that ensures that the player that is active in the initial positions wins.

The winning strategy maintains the invariant that whenever it is the turn of the player, the state of the game is not balanced. Whenever it is her turn, she picks a move that makes the resulting state balanced, which is possible by Lemma 2.9. Whenever it is the turn of the opponent, she has to make a move that makes the state unbalanced again by Lemma 2.8. Each play that is played like this is winning for the player that is initially active: Whenever the opponent has to move, she is in a balanced state and thus cannot directly reach the winning state $(0,\ldots,0)$ by Lemma 2.7. Since every play of Nim is finite, $(0,\ldots,0)$ has to be reached at some point. This proves that eventually, the initially active player wins by reaching $(0,\ldots,0)$ with her move.

For the other direction of the proof, assume that the initial position is $(c_1,\ldots,c_k)$ balanced. We prove that the player that is not active then has a winning strategy. This is sufficient to show that the active player cannot enforce that she wins (see Lemma 3.9).

By Lemma 2.8, the active player has no choice but to go to an unbalanced state. In this state, the opponent is now the active player, and she can use the above strategy from the first part of the proof to ensure that she wins the play. ■

2.10 Example
The theorem shows that $(2,2,1)$ is indeed a good position for player 1. But the move that player 1 made in Example 2.2 is not optimal, it leads to the unbalanced state $(2,2,1)$ with Nim sum 1 1. To ensure that she wins, she would have to take the single coin on the last pile, leading to state $(2,2,0)$ with Nim sum 2 0. If the other player now takes a whole pile (state $(2,0,0)$, Nim sum 1 0), player 1 wins by taking the other pile. If the other player takes only one coin from one pile (state $(2,1,0)$, Nim sum 1 1), player 1 can get to a balanced state by taking one coin from the other pile (state $(1,1,0)$, Nim sum 0 2). From this position one, it is easy to see that player 2 has to take the second to last coin, and player 1 can take the last coin.

2.11 Remark
As mentioned above, Nim is a so-called **impartial** game. This means that

- the possible moves from a state of the game are independent of which player is active,
- all plays have finite length,
- the player who cannot move anymore loses.

The **Sprague–Grundy theorem** shows that for every such impartial game, there is an initial state of Nim that is equivalent to it.

Exercises

2.12 Exercise
Complete the tree from Example 2.3, i.e. draw the full tree of plays for the initial state

$(2, 2, 1)^1$, where we assume that player 1 has to move first. For every node, write down the Nim sum. Furthermore, mark all winning states in the tree.

2.13 Exercise

Prove Lemma 2.9: Let $(c_1, \ldots, c_k)$ be an unbalanced state. There is a successor state (i.e. a state to which we can go with one single move) that is balanced.

Hint: Consider the smallest index j such that $\mathrm{Nim}\Sigma(c_1, \ldots, c_k)_j$ is odd. (Note that "smallest" means that the corresponding bit is most significant.) Prove that there is an index i with $c_{ij} = 1$ that can be modified to get to a balanced state.

3. Games with perfect information – Basic definitions

The goal of this section is to provide the basic definitions. The rest of the lecture will be based on them. We need to define games, plays, and the winner of plays. Furthermore, we consider strategies, systematic ways of playing.

Games and Plays

3.1 Definition: Game

A **sequential two-player board game with perfect information** $\mathcal{G}$, shortly referred to as **game** in the rest of the lecture, consists of a **game arena** and a **winning condition**.

A **game arena** is a directed graph $G = (V, R)$ together with a function

$$owner \colon V \to \{\bigcirc, \square\}$$

that assigns to each vertex in V an owner, either the **universal player** $\square$ or the **existential player** $\bigcirc$.

We postpone the definition of the **winning condition** as it needs more notation.

The **vertices** V of the graph are the possible states of the game, we will mostly call them **positions** (or sometimes also **configurations**) in this lecture. The **arcs** R of the graph are the **moves** or **transitions** of the game that connect the positions.

We usually write a game arena as $G = (V_\square \uplus V_\bigcirc, R)$, i.e. instead of explicitly specifying the ownership function, we give an implicit definition that is based on a partition of the positions into the positions owned by each player.

We will assume throughout the lecture that R contains no parallel arcs (arcs that have the same origin and destination). Consequently, each arc is uniquely specified by a tuple $(o, d) \in V \times V$ consisting of its origin o and its destination d, and we can see $R \subseteq V \times V$ as a set of such tuples. We allow self-loops, i.e. arcs (o, d) with $o = d$.

In the rest of this section, we assume $G = (V_\square \uplus V_\bigcirc, R)$ to be some fixed game arena.

Before we can formally define what a winning condition is, we need to understand how a game is played.

Intuitively, we assume that at each point in time, a token is placed on one position of the game arena. Then, the owner of this position picks an arc of the game arena originating in the current position and moves the token to its destination. This continues ad infinitum or until the token is in a position for which there is no leaving arc. The resulting path of the token in the game arena is called a play.

3.2 Definition: Play
A **play** of a game is a finite or infinite path in its game arena.

Each play is uniquely identified by a finite or infinite sequence of positions $p = p_0p_1p_2\dots$ such that (p_i, p_{i+1}) is an arc of the arena for all i. (Here, we use that R is parallel-free.)

The length of a finite play $p_0 \dots p_k$ is $|p| = k$, meaning we count the number of moves that have been made. In this case, we also write p_{last} to denote the last position p_k. We write $|p| = \omega$ for infinite plays.

The "for all i" above should be read as: For all $i \in \{0, \dots, k-1\}$ if the play is finite and has length k, and for all $i \in \{0, \dots, \omega\} = \mathbb{N}$ if the play is infinite.

A position x is **live** if it has at least one successor in the game arena (i.e. there is an arc $(x, y) \in R$ for some $y \in V$). If a position has no successor it is called **dead** or a **deadlock**.

We call a finite play **alive** resp. **dead** or **deadlocked** if its last position is live resp. dead.

We call a play **maximal** if it cannot be prolonged, i.e. if it is infinite or finite but dead.

For a play that is alive, we call the player **active** that owns the last position. Intuitively, this player should make the next move.

In a play p, we think of a move (p_i, p_{i+1}) as chosen by the owner of p_i, i.e. each player chooses the next position whenever she owns the current position.

We write

- $Plays$ for the set of all plays,
- $Plays_{inf}$ for the set of all infinite plays,
- $Plays_{max}$ for the set of all maximal plays,
- $Plays_{\Box}$ resp. $Plays_{\bigcirc}$ for the finite plays in which player $\Box$ resp. $\bigcirc$ is active.

Sometimes, we only want to consider the plays that start in some fixed initial position x, i.e. plays p with $p_0 = x$. We call such plays the **plays from** x, and write $\mathit{Plays}(x), \mathit{Plays}_{\mathit{inf}}(x), \ldots$

3.3 Remark

a) We only consider **two-player** games, but extending the definitions to k-player games is straightforward. Luckily, as we will see in Exercise 3.16, any perfect-information game for $k > 2$ players can be reduced to two-player games. Note that this is not true for games with imperfect information.

b) Our games are called **sequential** because one move happens after the other. There are other types of games in which the players move simultaneously. While some of these games can be easily sequentialized, recall that in prisoner's dilemma, it was important that both players moved simultaneously without any knowledge of the move of the other players. Simultaneous moves may introduce an aspect of hidden information, a case which is not considered in this lecture.

c) We assume that a game is essentially given by its set of positions and set of moves. In game theory, this is sometimes called the **extensive form**. To handle games in which the set of positions V is infinite, one needs a finite representation of the game arena to handle them algorithmically.

3.4 Definition: Winning condition

The **winning condition** *win* of a game is a function

$$\mathit{win}\colon \mathit{Plays}_{\mathit{max}} \to \{\bigcirc, \square\}$$

that assigns each maximal play p its winner $\mathit{win}(p) \in \{\bigcirc, \square\}$.

We say that a maximal play p is won by the universal resp. existential player if $\mathit{win}(p) = \square$ resp. $\mathit{win}(p) = \bigcirc$.

With this definition, a game can be seen as a tuple $\mathcal{G} = (G, \mathit{win})$ consisting of a game arena and of a winning condition for maximal plays on this arena.

3.5 Remark

According to our definition of *winning*, each maximal play has a unique winner, i.e. there is a winner, and at most one player wins. In particular, we do not allow a draw as a possible outcome. Many games that you know from real life allow a draw as a possible outcome, e.g. chess. Such games cannot be directly studied using our methods. It is a common technique to consider variants of the game in which a draw is seen as a win for one of the players. We apply this trick to chess in Example 3.13.

Strategies

The goal of each player is to pick her moves such that the resulting maximal play is winning for her. Since the maximal plays are partitioned into the plays won by each of the players, both cannot reach their goal at the same time.

For one maximal play, the winning function determines the winner. Instead of just considering one play at a time, we are interested in checking whether a player can enforce that she wins always by playing cleverly, no matter what her opponent does. This is formalized using the concept of strategies.

In the rest of this course, we assume that $\star \in \{\bigcirc, \square\}$ is one of the players and $\overline{\star}$ is the other player, i.e. $\{\star, \overline{\star}\} = \{\bigcirc, \square\}$.

3.6 Definition: Strategy
A **strategy** for player $\star \in \{\bigcirc, \square\}$ is a function

$$s_{\star} \colon \mathit{Plays}_{\star} \to V$$

that assigns each finite play p such that $\star$ is active in p_{last} a vertex $s_{\star}(p) \in V$ such that $(p_{last}, s_{\star}(p)) \in R$ is a valid move in the arena.

A strategy for player $\star$ fixes the behavior of $\star$ during a play: Whenever it is her turn, she executes the move that is the value for the play up to this point returned by the strategy. If all such plays are won by $\star$, we call the strategy a winning strategy.

3.7 Definition: Conforming, Winning strategy
A play $p \in \mathit{Plays}$ **conforms** to a strategy $s_{\star}$ if for all $p_i \neq p_{last}$ such that $p_i \in V_{\star}$ is owned by $\star$, we have $p_{i+1} = s_{\star}(p_0 \dots p_i)$.

A strategy $s_{\star}$ is a **winning strategy for player $\star$ from position** x if every maximal play $p \in \mathit{Plays}_{max}(x)$ from x that conforms to $s_{\star}$ is won by $\star$.

When we say that we want to **solve** a game from a certain position x, we mean that we want to check which player has a winning strategy from x. Similarly, solving a game means that we want to characterize for each of the players the positions from which she has a winning strategy.

3.8 Definition
A vertex is **winning** for player $\star$ if she has a winning strategy from position x.

The set of all such vertices is called the **winning region** $W_{\star} \subseteq V$.

Naively, it seems that for each position $x \in V$, there are four cases:

- None of the players could have a winning strategy, i.e. $x \notin W_{\bigcirc}, x \notin W_{\square}$,
- Exactly one of the players could have a winning strategy,
  i.e. $x \in W_{\bigcirc}, x \notin W_{\square}$ or $x \notin W_{\bigcirc}, x \in W_{\square}$, or
- Both could have a winning strategy,
  i.e. $x \in W_{\bigcirc}, x \in W_{\square}$.

For most games, for each of the positions, one of the players has a winning strategy and the other does not. There are games in which none of the players have a winning strategy for some positions, we will see an example much later in the lecture. The following lemma states that the last case can never occur.

3.9 Lemma
For each position x, at most one of the players has a winning strategy.

In particular, $W_{\square} \cap W_{\bigcirc} = \emptyset$.

Proof:
Towards a contradiction, assume that for some position x, both players have a winning strategy $s_{\square}$ resp. $s_{\bigcirc}$. Consider a maximal play p that is conform to both $s_{\square}$ and $s_{\bigcirc}$. In fact, there exists a unique play satisfying this condition that we can inductively construct by

$$p_0 = x\,, \qquad\qquad p_{i+1} = \begin{cases} s_{\square}(p_0 \dots p_i) & \text{if } p_i \in V_{\square}\,, \\ s_{\bigcirc}(p_0 \dots p_i) & \text{if } p_i \in V_{\bigcirc}\,. \end{cases}$$

Since $s_{\square}$ is winning from x, we have $win(p) = \square$. Similarly, we obtain $win(p) = \bigcirc$, a contradiction. ■

After we have checked that there is a winning strategy, we are also interested in finding a *simple* winning strategy. According to the definition, the strategy can make its return value dependent on the whole history of the play, which is finite but unboundedly long.

We are interested in strategies that only take the current position into account and do not look at the history at all.

3.10 Definition
A strategy $s_{☆}$ is called **positional** if for each two plays $p, p' \in Plays_{☆}$ with $p_{last} = p'_{last}$, we have $s_{☆}(p) = s_{☆}(p')$.

Positional strategies are also called *memoryless* in the literature, because they cannot store any information on the history of the play at all. For the same reason, a positional winning strategy $s_{☆}$ is usually given as a function with the signature

$$\left\{x \in V_{☆} \,\middle|\, x \text{ is alive}\right\} \to V\,.$$

As we will see later, there are games in which a position is winning, but no positional strategies exists.

3.11 Remark
Furthermore, we are interested in strategies that are:

- **Uniform**: Instead of having one winning strategy for each position in $W_{☆}$, we want to have one single strategy that is winning from all positions in $W_{☆}$.

  If we allow arbitrary strategies, then in fact uniform strategies do always exist. If we only consider positional strategies, then there are games that have positional winning strategies, but no uniform positional winning strategies.

- Easy to implement & computationally inexpensive: Instead of just allowing positional strategies, one can consider strategies that are allowed to store some information on the history of the play. To do so, we see a strategy $s_{☆}$ as a transducer, an automaton with input and output. It reads moves made by the opponent, i.e. arcs $(x, y) \in R$ with $x \in V_{\overline{☆}}$, and whenever the play has reached a position $x' \in V_{☆}$ (that is alive), it outputs a move $(x', y) \in R$.

  A strategy that can be realized by a deterministic transducer with finite memory and no additional storage mechanism (the transducer equivalent of DFAs) is called **finite memory strategy**. A strategy that can be realized by a deterministic transducer that uses a stack as storage is called **pushdown strategy**. A strategy that can be realized by a deterministic transducer that uses a tape as storage (similar to a Turing machine) is called **computable strategy**.

Before advancing the theoretical development, we take the Nim game from the previous section and formalize it as a game according to the definitions of this section. Furthermore, we consider several other examples.

3.12 Example
The game **Nim** can be defined as follows:

- The set of positions is
$$V = \mathbb{N}^* \times \{\bigcirc, \square\} .$$
The first component of a position $(\vec{c}, \star)$ is a finite sequence $\vec{c}$ of natural numbers, each entry c_i denoting the number of coins on pile i. The second component $\star$ is denoting the active player, i.e. $V = V_\square \uplus V_\bigcirc = \left(\mathbb{N}^* \times \{\square\}\right) \uplus \left(\mathbb{N}^* \times \{\bigcirc\}\right)$.

- The moves are defined as follows:
$$R = \left\{ \left((\vec{c}, \star), (\vec{d}, \overline{\star})\right) \;\middle|\; \begin{array}{l} \vec{c}, \vec{d} \in \mathbb{N}^k \text{ for some } k \in \mathbb{N}, \\ \exists i_0 \in \{0, \ldots, k-1\} \colon d_{i_0} < c_{i_0} \text{ and } c_i = d_i \text{ for all } i \neq i_0 \end{array} \right\} .$$

- The winning condition is given by $win(\ldots(\vec{0}, \star)) = \overline{\star}$, i.e. if we reach position $\vec{0}$, the active player that would have to move next loses the game. Note that every maximal play necessarily ends in a position of the shape $(\vec{0}, \star)$.

- The winning regions can be characterized using the Nim sum,
$$W_\star = \{(\vec{c}, \star) \mid \mathrm{Nim}\Sigma(\vec{c}) \text{ is unbalanced }\} \cup \left\{(\vec{c}, \overline{\star}) \;\middle|\; \mathrm{Nim}\Sigma(\vec{c}) \text{ is balanced }\right\} .$$
Note that $V = W_\square \uplus W_\bigcirc$.

- The strategy presented in the proof of Theorem 2.6 is positional and uniform:
$$s_\star(\vec{c}, \star) = (\vec{d}, \overline{\star}) ,$$
where $\vec{d}$ is an arbitrary balanced successor if such a successor exists and an arbitrary successor otherwise.

Note that for Nim, the set of positions V is infinite, but from each given initial position $x \in V$, only finitely many positions are reachable.

3.13 Example
Chess is maybe the best known game with perfect information. In this exercise, we want to study it, in particular, we want to prove the following result:

In chess, it is the case that

1. either white has a winning strategy,

2. or black has a winning strategy,

3. or both have a drawing strategy.

Here, a winning strategy is a strategy ensuring that the player wins (in particular, the games conform to it do not end in a draw), while a drawing strategy is a strategy that only ensures that player does not lose, i.e. the game is won by her or ends in a draw. Furthermore, we are only interested in the typical initial board configuration of chess, so writing e.g. "white has a winning strategy" should mean that white has a winning strategy from this position.

While chess is intuitively a game with perfect information, it is not conforming to our definition, since "draw" is a possible outcome. To circumvent this issue, we use the following trick: We define two variants of chess, namely **white chess** and **black chess**. In white chess, the white players wins in the case of a draw, analogously for black chess. These variants are games that we can study with the methods presented in this lecture.

In the following, we will use the fact that white and black chess are determined, since they are games played on a finite graph in which each play has a bounded length. (As soon as a board configuration repeats three times, the game ends with a draw in "real" chess.) This means that for each position exactly one of the players has a winning strategy. We have not proven this result yet, but we will do so in the next section.

Using the result, we know that there are four possibilities:

- White has a winning strategy for white and for black chess. In this case, she has a winning strategy for "real" chess: The winning strategy for black chess ensures that the game does not end in what would be a draw in real chess, since draws are won by black in black chess. This is case 1. of the result that we want to prove.

- The analogous case for black gives us case 2. of the result.

- If both players have a winning strategy for the opposite variant of chess (white for black chess, black for white chess), we obtain a contradiction, similar to Lemma 3.9: Consider the play of real chess in which each player conform to her winning strategy. The strategies were winning strategies for the opposite variant of chess, meaning they are winning (and not drawing) in real chess. This means that the play is won by both players, a contradiction, so this case can never occur.

- Assume that each player has a winning strategy for her variant of chess (white for white chess, black for black chess). Since each strategy was winning in the variant of the game in which draws counts as wins, these strategies are drawing strategies for real chess.

> To see that both strategies cannot be winning strategies for real chess, consider the play in which each players conforms to her strategy. The resulting play is winning for white in white chess, and winning for black in black chess, so it has to be a draw in real chess.

This result is credited to a famous paper of Zermelo from 1903 [Zer13], see [SW01] for a discussion.

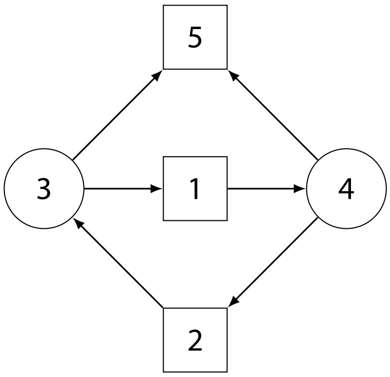
Exercises

3.14 Exercise: Tic-tac-toe
Consider the popular game **tic-tac-toe**,
see e.g. `https://en.wikipedia.org/wiki/Tic-tac-toe`.

Formalize the game, i.e. formally define a game $\mathcal{G} = (G, win)$ consisting of a game arena and a winning condition that imitates the behavior of tic-tac-toe.

Assume that player ○ makes the first mark, and the other player wins in the case of a draw.

3.15 Exercise: Positional and uniform strategies
If a game arena has finitely many positions, we can explicitly give it as a graph. For this exercise, we consider a game on the following game arena $G = (V, R)$. Positions owned by the universal player □ are drawn as boxes, positions owned by the existantial player ○ as circles. The numbers should denote the names of the vertices, i.e. $V = \{1, \dots, 5\}$.



We consider the following winning condition: A maximal play is won by the existential player if and only if the positions 3, 4 and 5 are each visited exactly once.

a) What is the winning region for each of the players?

Present a single strategy $s_{\bigcirc}\colon Plays_{\bigcirc} \to V$ that is winning from all positions x in the winning region $W_{\bigcirc}$ of the existential player. Argue shortly why your strategy is indeed winning from these positions.

Note: Such a strategy is called a *uniform* winning strategy.

b) For each vertex $x \in W_{\bigcirc}$ in the winning region of the existential player, present a positional strategy for existential player $s_{\bigcirc,x}\colon \{3,4\} \to R$ such that $s_{\bigcirc,x}$ is winning from x.

c) Prove that there is no uniform positional winning strategy for the existential player, i.e. no single positional strategy that wins from all $x \in W_{\bigcirc}$.

d) Consider the modified graph that is obtained by adding a vertex 6 owned by □ and the arcs $(6,3)$ and $(6,4)$.
Prove that position 6 is winning for the existential player, but there is no positional winning strategy from 6.

3.16 Exercise: Multiplayer games

Assume that three-player games are defined analogously to two-player games, i.e. they are played on a directed graph with an ownership function $owner\colon V \to \{1,2,3\}$, and their winning condition is a function $win\colon Plays_{max} \to \{1,2,3\}$. (Winning) strategies are defined similar to two-player games.

For every three-player game $\mathcal{G}_{3p} = (G_{3p}, win_{3p})$, where $G_{3p} = (V_1 \uplus V_2 \uplus V_3, R)$ and each player $i \in \{1,\ldots,3\}$, show how to construct a two-player game $\mathcal{G}_i = (G_i, win_i)$ with $G_i = (V_{\Box} \uplus V_{\bigcirc}, R)$ such that:

- The underlying directed graph is the same, i.e. $V_1 \uplus V_2 \uplus V_3 = V_{\Box} \uplus V_{\bigcirc}$.
- Each node $x \in V_1 \uplus V_2 \uplus V_3$ is winning for player i in the game $\mathcal{G}_{3p}$ if and only if it is winning for player ○ in the game $\mathcal{G}_i$.

Prove that your constructed game $\mathcal{G}_i$ has the desired properties.

3.17 Exercise: Deadlocks

Many books in the literature only consider games that are deadlock-free, meaning every position $x \in V$ has at least one outgoing arc $(x,y) \in R$ (where self-loops, i.e. $x = y$, are allowed).

Assume that $\mathcal{G} = (G, win)$ is a game that may contain deadlocks. Furthermore, we assume that the winning condition has the property that any finite play ending in a deadlock is lost by the player owning the last position.

Construct a game $\mathcal{G}' = (G', win')$ that does not contain deadlocks. The new game arena G' should be obtained from G by adding vertices and arcs, in particular each position of the old game is a position of the new game, $V \subseteq V'$.

Your construction should guarantee that each position $x \in V$ of the old game is winning in the new game for the same player for which it was winning in the old game. Argue why it has this property.

3.18 Exercise: Language inclusion as a game
Note: You may need to recall the definitions of finite automata for this exercise.

Consider two non-deterministic finite automata (*NFAs*) $A = (Q_A, q_{0A}, \rightarrow_A, Q_{FA})$ respectively $B = (Q_B, q_{0B}, \rightarrow_B, Q_{FB})$ over the same alphabet of input symbols Σ. We want to construct a game that is won by the universal player □ if and only if the regular language accepted by A is included in the regular language accepted by B, i.e. $\mathcal{L}(A) \subseteq \mathcal{L}(B)$.

Our approach is to let each of the players control one of the automata. The existential player controls automaton A, and her goal is to disprove inclusion. To do so, she step-by-step picks a run of A such that the corresponding word is accepted by A, but not accepted by B. The universal player wants to prove inclusion and controls automaton B. She has to react to the moves made by the existential player to find an accepting run of automaton B for the word chosen by existential player.

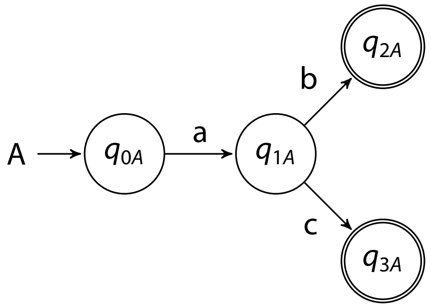
More precisely, the game works as follows:

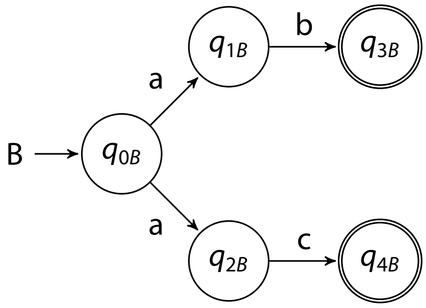
- A configuration of the game consists of a state q_A resp. q_B of each automaton.
- The players alternately takes turns, starting with the existential player ○.
- In each of her turns, ○ selects a transition $q_A \xrightarrow{a}_A q'_A$ of the automaton A.
- In the following turn, the universal player selects a transition $q_B \xrightarrow{a}_B q'_B$ of B. Note that it has to be labeled by the same letter $a \in \Sigma$ that was picked by the existential player in the previous move.
- A maximal play of the game is won by the existential player if it visits a configuration in which the state q_A of A is final, but the state q_B of B is not final (Intuitively, this means that the word chosen step-by-step by refuter is accepted by A, but not accepted by B.) It is also won by ○ if it ends in a position in which □ cannot react

to a move, i.e. there is no transition of B with the required letter. It is won by the universal player □ otherwise.

a) Formalize the game, i.e. formally define a game arena G and a winning condition win such that the game $\mathcal{G} = (G, win)$ has the behavior described above.

b) Let x be the configuration of the game consisting of the initial states q_{0A} and q_{0B} of both automata. We would like to have the following result:
"x is winning for the universal player if and only if the inclusion $\mathcal{L}(A) \subseteq \mathcal{L}(B)$ holds."
Prove that this is **not** true in general by considering the following automata over the alphabet $\{a, b, c\}$.





Part II.
Games on finite graphs

Contents

4. Reachability & safety games

In the last section, we have allowed arbitrary functions as winning conditions without imposing any restriction. In the following sections, we will study specific types of winning conditions. For each of them, we will develop a theory that allows us to conclude that the corresponding games are determined. The theory also leads to algorithms that can be used to compute the winner in case the game arena is finite.

We start with the two most simple conditions, reaching resp. avoiding positions from a given set. More formally, the **reachability condition** is satisfied if the play reaches a position in a given **winning set**. Its analogon is the **safety condition**, for which the play needs to avoid a position in a given **losing set**.

Many games that you know from real life are of this type, e.g. in chess, the winning positions are given by the configurations of the board with checkmate.

Sources
The content of this section is common knowledge in game theory and can be found in most textbooks on the topic. The presentation here does not follow any particular source.

Reachability and safety games

In the following, we assume that the existential player ○ is the player that wins if the set of winning positions is reached, and that the universal player □ wants to avoid this. One can easily adapt the theory for the opposite case by swapping the players everywhere.

4.1 Assumption
In this section, let $G = (V_\square \uplus V_\bigcirc, R)$ be a fixed game arena. We furthermore assume that G has **finite out-degree**, i.e. for each position x, the set of successors $\{y \in V \mid (x, y) \in R\}$ is finite.

Note that in particular, the assumption is satisfied if V is finite.

4.2 Definition: Reachability games
Let $B \subseteq V$ be a set of positions called the **winning set**.

The **reachability game** on G with respect to B is the game whose winning condition is given by

$$\begin{aligned} win \ : \ Plays_{max} &\to \{\bigcirc, \Box\} \\ p &\mapsto \begin{cases} \bigcirc & \text{if } \exists i\colon p_i \in B\,, \\ \Box & \text{else, i.e. } \forall i\colon p_i \notin B\,. \end{cases} \end{aligned}$$

4.3 Remark
The game specified by the above winning condition can also be seen from the perspective of the universal player. It is then called the **safety game** with respect to the **losing set** B.

Our goal is to show that reachability/safety games are uniformly positionally determined, by proving the following theorem.

4.4 Theorem: Reachability/safety games are positionally determined
Reachability/safety games are uniformly positionally determined: The set of positions can be partitioned into the winning regions for each of the players, and each player has a uniform positional winning strategy, a positional strategy that is uniformly winning from all positions in her winning region.

Attractor

In order to solve the reachability game, we need to compute the set of positions from which $\bigcirc$ can enforce that a play visits a position in B. We start by considering the set of positions from which an immediate visit of B (within one move) can be enforced. The definition is parametric in the player $\star \in \{\bigcirc, \Box\}$ of interest, as we will reuse it later.

4.5 Definition: Controlled predecessors
For a set $X \subseteq V$ of positions, the **controlled predecessors** for player $\star \in \{\bigcirc, \Box\}$ are

$$\mathrm{CPre}_{\star}(X) = \left\{x \in V_{\star} \mid \exists (x,y) \in R\colon y \in X\right\} \cup \left\{x \in V_{\overline{\star}} \,\middle|\, x \text{ is live}\,,\ \forall (x,y) \in R\colon y \in X\right\}.$$

The controlled predecessors of X contain all positions of player $\star$ for which there is a move to a position in X. If a play reaches such a position, then $\star$ can enforce a visit of X within one step. Furthermore, a position owned by the opponent $\overline{\star}$ is a controlled predecessor of X if all possible moves lead to X (and there is at least one move). If a play reaches such a position, the opponent cannot prevent a visit of X within one step.

To obtain the set of all positions from which ☆ can enforce a visit of a given set, we apply the controlled predecessors iteratively. We formalize this in the definition of the attractor.

4.6 Definition: Attractor

Let $B \subseteq V$ be a set. The **i-step attractor** $\mathrm{Attr}^i_{☆}(B)$ of B for player $☆ \in \{○, □\}$ is inductively defined as follows:

$$\begin{aligned}\mathrm{Attr}^0_{☆}(B) &= B\\ \mathrm{Attr}^{i+1}_{☆}(B) &= \mathrm{Attr}^i_{☆}(B) \cup \mathrm{CPre}_{☆}\big(\mathrm{Attr}^i_{☆}(B)\big)\end{aligned}$$

The **attractor** $\mathrm{Attr}_{☆}$ is the union of the i-steps attractors for all i,

$$\mathrm{Attr}_{☆}(B) = \bigcup_{i \in \mathbb{N}} \mathrm{Attr}^i_{☆}(B)\,.$$

The i-step attractor is the set of all positions from which ☆ can enforce visiting B within at most i steps. The player can enforce visiting B in zero steps if and only if the current position is already in B, justifying the base case of the definitions. From all positions from which the player can enforce visiting B in at most i steps, she can of course also enforce a visit in at most $i+1$ steps, so $\mathrm{Attr}^i_{☆} \subseteq \mathrm{Attr}^{i+1}_{☆}$. This in particular means that the attractors form a chain

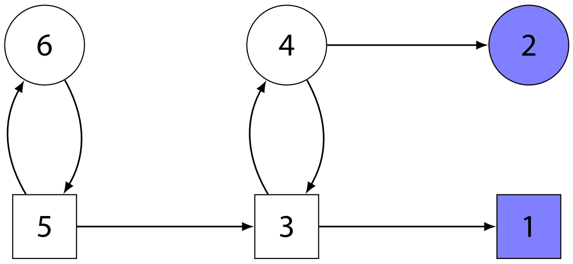
$$\mathrm{Attr}^0_{☆}(B) \subseteq \mathrm{Attr}^1_{☆}(B) \subseteq \mathrm{Attr}^2_{☆}(B) \subseteq \dots\,.$$

To enforce a visit of B in at most $i + 1$ steps, it is sufficient to move to a position from which a visit of B in at most i steps can be enforced.

If we are interested in all positions from which ☆ can enforce visiting B in an arbitrary but finite number of steps, we have to take the union of all i-step attractors.

4.7 Example

Consider the game arena given by the following picture.



We want to compute the attractor of the blue-colored positions for the existential player, i.e. $\mathrm{Attr}_{\bigcirc}(\{1,2\})$.

0. Initially, we have $\mathrm{Attr}_{\bigcirc}^{0}(\{1,2\}) = \{1,2\}$.

1. By the definition, we have $\{1,2\} \subseteq \mathrm{Attr}_{\bigcirc}^{1}(\{1,2\})$. When we check for position 3 whether it should be contained in $\mathrm{Attr}_{\bigcirc}^{1}(\{1,2\})$, we see that there is a successor – namely position 4 – that is not in $\mathrm{Attr}_{\bigcirc}^{0}(\{1,2\})$, so we have $3 \notin \mathrm{Attr}_{\bigcirc}^{1}(\{1,2\})$ since 3 is owned by the universal player. Similarly, positions 5 and 6 are not in $\mathrm{Attr}_{\bigcirc}^{1}(\{1,2\})$.

   Position 4 is owned by the existential player and has a successor in $\mathrm{Attr}_{\bigcirc}^{0}$, so we need to add it. We obtain $\mathrm{Attr}_{\bigcirc}^{1}(\{1,2\}) = \{1,2,3\}$.

2. Now, all successors of position 3 are in $\mathrm{Attr}_{\bigcirc}^{1}(\{1,2\})$, so we need to add 3 to the attractor. Still, for both 5 and 6 no successor is contained in $\mathrm{Attr}_{\bigcirc}^{1}(\{1,2\})$.

   We obtain $\mathrm{Attr}_{\bigcirc}^{2}(\{1,2\}) = \{1,2,3,4\}$.

3. Now position 5 has a successor in the attractor, but it is owned by the opponent, so we do not add it. We obtain

$$\mathrm{Attr}_{\bigcirc}(\{1,2\}) = \mathrm{Attr}_{\bigcirc}^{3}(\{1,2\}) = \mathrm{Attr}_{\bigcirc}^{2}(\{1,2\}) = \{1,2,3,4\}\,.$$

4.8 Lemma
Let $X, Y \subseteq V$ be sets of positions.

a) If $X \subseteq Y$, then $\mathrm{CPre}_{☆}(X) \subseteq \mathrm{CPre}_{☆}(Y)$ and $\mathrm{Attr}_{☆}(X) \subseteq \mathrm{Attr}_{☆}(Y)$

b) $\mathrm{CPre}_{☆}(\mathrm{Attr}_{☆}(X)) \subseteq \mathrm{Attr}_{☆}(X)$.

Proof: Part a) is immediate by the definition. Part b) is essentially Exercise 4.13. ■

The attractor allows us to solve reachability games.

4.9 Theorem
Consider the reachability game with respect to the winning set $B \subseteq V$. The set $\mathrm{Attr}_{\bigcirc}(B)$ is the winning region of the existential player, and its complement is the winning region of the universal player.

The theorem in particular claims that $V = W_{\bigcirc} \uplus W_{\Box} = \mathrm{Attr}_{\bigcirc}(B) \uplus (V \setminus \mathrm{Attr}_{\bigcirc}(B))$. We will construct positional strategies $s_{\Box}, s_{\bigcirc}$ such that $s_{\Box}$ is uniformly winning from all positions in $V \setminus \mathrm{Attr}_{\bigcirc}(B)$ and $s_{\bigcirc}$ is uniformly winning from all positions in $\mathrm{Attr}_{\bigcirc}(B)$. This will prove Theorem 4.4 as well as Theorem 4.9.

Given any live position $x \in V_{\Box}$, $s_{\Box}(x)$ returns a move $(x, y) \in R$ with $y \in W_{\Box} = V \setminus \mathrm{Attr}_{\bigcirc}(B)$ if such a move exists, and an arbitrary move otherwise. (To make the strategy deterministic, we fix one move if several exist.)

$$\begin{aligned} s_{\Box} \quad : \quad \{x \in V_{\Box} \mid x \text{ is live}\} \quad &\to \quad V \\ x \quad &\mapsto \quad \begin{cases} y \text{ with } (x, y) \in R, y \in W_{\Box}\,, & \text{if such a } y \text{ exists}\,, \\ y \text{ arbitrary with } (x, y) \in R\,, & \text{else}\,. \end{cases} \end{aligned}$$

4.10 Lemma
$s_{\Box}$ is a positional strategy that is uniformly winning from all positions in $W_{\Box} = V \setminus \mathrm{Attr}_{\bigcirc}(B)$.

Proof:
Let $p_0 \in W_{\Box}$ be an arbitrary initial position. We show that any play $p = p_0 p_1 p_2 \ldots$ that conforms to $s_{\Box}$ has the property $p_j \in W_{\Box}$ for all j

This already shows that p is won by the universal player: We have $B = \mathrm{Attr}^0_{\bigcirc}(B) \subseteq \mathrm{Attr}_{\bigcirc}(B)$, thus $B \cap W_{\Box} = \emptyset$. Any play with the above property will never visit a position in B.

We show that under the assumption $x \in W_{\Box}$, whenever the universal player is active, there is a move (x, y) leading to a position $y \in W_{\Box}$ that will then be selected by $s_{\Box}$ according to its definition. Under the same assumption, we show that when the existential player has to move, she has no choice but to go to a position in $W_{\Box}$. Those two proofs can be combined into an induction showing the desired property. In the base case, we have $p_0 \in W_{\Box}$ by assumption.

Let $p = p_0 \ldots p_j$ be a play conform to $s_{\Box}$ of length j such that p_j is not dead. By induction, we know that $p_j \in W_{\Box}$. (If p_j is dead, we are done, since the play is maximal and won by the universal player.)

Assume that it is the turn of the universal player, $p_j \in V_{\Box}$. Assume there is no move (x, y) to a position in $W_{\Box}$, meaning that all moves go to positions in $V \setminus W_{\Box} = W_{\bigcirc} = \mathrm{Attr}_{\bigcirc}(B)$. By the definition of the attractor, for each such y, there is i_y such that $y \in \mathrm{Attr}^{i_y}_{\bigcirc}$. Let $i_{max} = \max_y i_y$ be the maximum of the i_y, and note that this is a well-defined natural number since we assumed the game arena to have finite out-degree. By the definition of the attractor resp. the controlled predecessors, we have $x \in \mathrm{Attr}^{i_{max}+1}_{\bigcirc}(B) \subseteq W_{\bigcirc}$, a contradiction to the assumption $x \in W_{\square}$. Thus, there is a move to some $y \in W_{\square}$ as required, and the strategy will pick such a move by its definition.

Assume that it is the turn of the existential player, $p_j \in V_{\bigcirc}$. We need to argue that all moves (x, y) that she can pick go to a position in $y \in W_{\square}$. Assume there is a move to a position $y \in W_{\bigcirc}$. Then there is a number i such that $y \in \mathrm{Attr}^i_{\bigcirc}(B)$, and by the definition of the attractor resp. the controlled predecessors, we have $x \in \mathrm{Attr}^{i+1}_{\bigcirc}(B) \subseteq W_{\bigcirc}$, a contradiction. ■

The strategy for the existential player is a little bit more involved. The universal player wins a play by preventing it from visiting B forever, while the existential player has to ensure that the play visits B within a finite number of steps. If $s_{\bigcirc}$ would just work similar to $s_{\square}$ and pick an arbitrary move to $W_{\bigcirc}$, the strategy would ensure that all positions occurring in a play are inside the winning region, but it would not guarantee that there is some index $i \in \mathbb{N}$ such that B is visited after i steps.

To get rid of this problem, the strategy does not pick an arbitrary move (x, y) such that $y \in W_{\bigcirc} = \mathrm{Attr}_{\bigcirc}(B)$, but a move to $y \in \mathrm{Attr}^i_{\bigcirc}(B)$ such that i is minimal. If any move to $\mathrm{Attr}_{\bigcirc}(B)$ exists, we fix an arbitrary one that minimized i as described before. If no move to $\mathrm{Attr}_{\bigcirc}(B)$ exists, the strategy should return an arbitrary move.

$$\begin{aligned} s_{\bigcirc} \quad : \quad \{x \in V_{\bigcirc} \mid x \text{ is live}\} &\rightarrow V \\ x &\mapsto \begin{cases} y \text{ with } (x,y) \in R, y \in \mathrm{Attr}^i_{\bigcirc}(B) & \\ \quad \text{s.t. } i \text{ is minimal,} & \text{if } y \in \mathrm{Attr}_{\bigcirc}(B) \text{ exists,} \\ y \text{ arbitrary with } (x,y) \in R, & \text{else.} \end{cases} \end{aligned}$$

4.11 Lemma
$s_{\bigcirc}$ is a positional strategy that is uniformly winning from all positions in $W_{\bigcirc} = \mathrm{Attr}_{\bigcirc}(B)$.

Proof:
We have to prove that for all $p_0 \in W_{\bigcirc} = \mathrm{Attr}_{\bigcirc}(B)$, all plays that conform to $s_{\bigcirc}$ from p_0 are won by the existential player. For each such $p_0 \in W_{\bigcirc} = \mathrm{Attr}_{\bigcirc}(B)$, there is an $i \in \mathbb{N}$ such that $p_0 \in \mathrm{Attr}^i_{\bigcirc}(B)$ by the definition of the attractor. Let $i_0 \in \mathbb{N}$ be the minimal i with this property.

We prove the required statement by induction on i_0. In the base case, we have $i_0 = 0$. This means $p_0 \in \mathrm{Attr}^0_{\bigcirc}(B) = B$. Any play from p_0 visits B and is won by $\bigcirc$.

In the inductive step, assume that $i_0 > 0$ is some number such that the statement holds for all $i < i_0$. Consider an arbitrary $p_0 \in \mathrm{Attr}^{i_0}_{\bigcirc}(B)$. We know that

$$\mathrm{Attr}^{i_0}_{\bigcirc}(B) = \mathrm{Attr}^{i_0-1}_{\bigcirc}(B) \cup \mathrm{CPre}_{\bigcirc}\left(\mathrm{Attr}^{i_0-1}_{\bigcirc}(B)\right).$$

Since we assumed that i_0 is minimal, the case $p_0 \in \mathrm{Attr}_{\bigcirc}^{i_0-1}(B)$ cannot occur. We thus know $p_0 \in \mathrm{CPre}_{\bigcirc}\big(\mathrm{Attr}_{\bigcirc}^{i_0-1}(B)\big)$.

In the case that $p_0 \in V_{\bigcirc}$ is owned by the existential player, there is a move (p_0, y) with $y \in \mathrm{Attr}_{\bigcirc}^{i_0-1}(B)$, and the strategy picks one such successor. Thus, any play from p_0 is of the shape $p = p_0p_1p_2\ldots$ where $p_1 \in \mathrm{Attr}_{\bigcirc}^{i_0-1}(B)$. If p conforms to $s_{\bigcirc}$, then also the suffix $p' = p_1p_2\ldots$ is a play that conforms to $s_{\bigcirc}$. By induction, we obtain that p' is won by $\bigcirc$, i.e. it visits B. The play p, obtained from p' by prepending p_0, thus also visits B and is also won by $\bigcirc$.

If $p_0 \in V_{\square}$, we argue similarly. Since $p_0 \in \mathrm{CPre}_{\bigcirc}\big(\mathrm{Attr}_{\bigcirc}^{i_0-1}(B)\big)$, any successor y of x picked by the universal player satisfies $y \in \mathrm{Attr}_{\bigcirc}^{i_0-1}(B)$. Thus, no matter which successor p_1 is picked, we may apply induction to obtain that any play from p_1 that conforms to $s_{\bigcirc}$ is winning. Prepending p_0 does not change this fact. ■

Together, Lemma 4.10 and Lemma 4.11 prove Theorem 4.4 and Theorem 4.9.

If the game arena is finite, Theorem 4.4 gives directly rise to an algorithm that determines the winning region by computing the attractor of the winning set. In fact, one can set up the algorithm in a clever way such that

a) it also computes the winning strategies $s_{\bigcirc}$ and $s_{\square}$,

b) its running time is linear in $|V| + |R|$.

For a), we tweak the computation of the attractor as follows:

- Whenever a position $x \in V_{\bigcirc}$ is added to the attractor for the first time, say to $\mathrm{Attr}_{\bigcirc}^{i+1}(B)$, we set $s_{\bigcirc}(x) = (x, y)$, where (x, y) is the move that caused x to be added to the attractor.
- Whenever a position $x \in V_{\square}$ is found to not belong to the attractor, we set $s_{\square}(x) = (x, y)$, where (x, y) is a move with $y \notin \mathrm{Attr}_{\bigcirc}(B)$ (yet).

Note that while $s_{\bigcirc}(x)$ is fixed after it is set once, $s_{\square}(x)$ might need to be updated if the previously selected move later turns our to lead to the attractor.

For b), notice that naively, the algorithm is linear in $|V|^2 \cdot |E|$: In each iteration of the attractor computation (i.e. whenever we compute the $i+1$-step attractor), we need to consider all vertices that are not yet in the attractor, and check their successors. To obtain $\mathrm{Attr}_{\bigcirc}(B)$, we have to compute at most $|V|$ steps, as we will see in Exercise 4.13.

To get the running time down to $\mathcal{O}(|V| + |R|)$ we need to assign a counter $c(x)$ to each vertex $x \in V$. This counter is initially 1 for positions owned by the existential player, and $|\{y \mid (x, y) \in R\}|$, i.e. the number of successors, for positions owned by the universal player.

Whenever we add a position y to the attractor, we decrease $c(x)$ by one for all positions x that are a predecessor of y, i.e. $(x, y) \in R$. Whenever the counter $c(x)$ of a position that is not yet in the attractor drops to 0, we add it to the attractor.

To start the algorithm, we add all positions in B to the attractor.

Exercises

4.12 Exercise: 2×2 tic tac toe
Consider a 2×2-variant of tic tac toe, i.e. tic tac toe played on a 2×2 matrix. We assume that ○ starts. The player that is first able to put 2 of her marks into one row, column or diagonal wins, and the game then stops.

Formalize this game as a reachability game, draw the game arena as a graph, and solve it explicitly using the attractor algorithm.

4.13 Exercise: Attractors have attractive algorithmics

a) Prove that if $\mathrm{Attr}^{i}_{☆}(B) = \mathrm{Attr}^{i+1}_{☆}(B)$, then we have $\mathrm{Attr}^{i}_{☆}(B) = \mathrm{Attr}_{☆}(B)$.

   Conclude that if the set of positions V is finite, we have $\mathrm{Attr}_{☆}(B) = \mathrm{Attr}^{|V|}_{☆}(B)$.

b) Let $G = (V, E)$ be a finite game arena, and let $B \subseteq V$ be a set. We consider the reachability game on G with respect to B.

   Write down pseudo-code for an algorithm that computes the winning region $W_{\bigcirc}$ of the existential player, and at the same time computes uniform positional winning strategies $s_{\bigcirc}, s_{\Box}$ for both players.

4.14 Exercise: Double-reachability games
Consider a finite game arena $G = (V_{\Box} \uplus V_{\bigcirc}, R)$ without deadlocks and sets $B_1, B_2 \subseteq V$. In the **double-reachability game** $\mathcal{G}$, ○ wins by enforcing that the play visits first B_1 and later B_2. More formally, the winning condition is given by

$$\begin{aligned} win \;:\; Plays_{inf} &\to \{\bigcirc, \Box\} \\ p &\mapsto \begin{cases} \bigcirc, & \text{if } \exists i \in \mathbb{N}\colon p_i \in B_1 \text{ and } \exists j \in \mathbb{N}, j > i\colon p_j \in B_2 \\ \Box, & \text{else.} \end{cases} \end{aligned}$$

a) Present an algorithm that takes a double-reachability game and computes a reachability game $\mathcal{G}'$ that contains the positions in V, i.e. a game arena $G' = (V', R')$ and a winning set $B' \subseteq V'$, with $V \subseteq V'$. For all $x \in V$, x should be winning for ○ in the double-reachability game $\mathcal{G}$ if and only if it is winning for ○ in the reachability game $\mathcal{G}'$. Argue formally that your algorithm is correct.

b) Present an algorithm that directly computes the winning regions of the double-reachability game. Argue that you algorithm is correct.

4.15 Exercise: Reach-and-stay games

Consider a finite game arena $G = (V_{\square} \uplus V_{\bigcirc}, R)$ without deadlocks and a winning set $B \subseteq V$. In a reachability game, any play that visits B is winning for ○, no matter how it continues after the visit.

In this exercise, we consider **reach-and-stay games**, in which the goal of player ○ is to enforce that the play visits B and stays there forever. More formally, the winning condition is given by

$$\begin{aligned} win \;:\; Plays_{inf} &\to \{\bigcirc, \square\} \\ p &\mapsto \begin{cases} \bigcirc, & \text{if } \exists i \in \mathbb{N}\colon \forall k \geq i\colon p_k \in B, \\ \square, & \text{else.} \end{cases} \end{aligned}$$

Present an algorithm that takes a finite game arena without deadlocks and the winning set and computes the winning regions of the reach-and-stay game. Argue that it is correct.

Do uniform positional strategies exist?

Hint: First identify the position form which one stays inside B forever.

4.16 Exercise: Determinacy of games of finite length

Let $\mathcal{G} = (G, win)$ be a game such that each maximal play of $\mathcal{G}$ has finite length. Then $\mathcal{G}$ is determined, i.e. every position is winning for exactly one of the players, $V = W_{\bigcirc} \uplus W_{\square}$.

Hint: Construct a reachability game whose set of positions is $Plays^{\mathcal{G}}$.

Note: When considering chess in Example 3.13, we have already used this result.

4.17 Exercise: Graphs with infinite out-degree

In this section, we made the assumption that the out-degree of the game arena is finite. In this exercise, we want to understand this restriction.

Let $\mathbb{N}^+ = \{1, 2, 3, \ldots\}$ denote the positive natural numbers. We consider the infinite graph $G = (V, R)$ given by

$$V = \{\mathit{start}, \mathit{goal}\} \uplus \bigcup_{i\in\mathbb{N}^+} \mathit{Path}_i\,,\ \text{ where for each } i \in \mathbb{N}^+,\ \text{ we have } \mathit{Path}_i = \left\{p^i_1, p^i_2, \ldots, p^i_i\right\},$$

$$R = \bigcup_{i\in\mathbb{N}^+}\left\{\left(\mathit{start}, p^i_1\right)\right\} \uplus \bigcup_{i\in\mathbb{N}^+}\left\{\left(p^i_i, \mathit{goal}\right)\right\} \uplus \bigcup_{i\in\mathbb{N}^+}\ \bigcup_{j=1}^{i-1}\left\{\left(p^i_j, p^i_{j+1}\right)\right\}.$$

We want to consider a reachability game on G with respect to the winning set $\{\mathit{goal}\}$, i.e. ○ needs to reach the position *goal*, □ wants to prevent this.

a) Draw a schematic representation of the graph G, e.g. involving the vertices $\{\mathit{start}, \mathit{goal}\}$ and the positions in Path_i for $i \leqslant 4$.

b) Assume that all positions are owned by the existential player. For each position $x \in V$, give the minimal $i_x \in \mathbb{N}$ such that $x \in \mathrm{Attr}^{i_x}_{\bigcirc}(\{\mathit{goal}\})$, respectively $i_x = \infty$ if no such i_x exists.

   Present a winning strategy for the reachability game from the position *start*.

c) Assume that all positions are owned by the universal player. For each position $x \in V$, give the minimal i_x such that $x \in \mathrm{Attr}^{i_x}_{\bigcirc}(\{\mathit{goal}\})$, respectively $i_x = \infty$ if no such i_x exists.

   Which player wins the reachability game from *start*?

4.18 Remark

In Part c) of the above exercise, we see that our attractor construction is not able to deal with game arenas that have infinite out-degree.

To fix the problem, we can use a non-constructive definition of the attractor: The **attractor** $\mathrm{Attr}_{\text{☆}}(B)$ is the smallest subset of positions that satisfies the following three properties:

(1) $B \subseteq \mathrm{Attr}_{\text{☆}}(B)$,

(2) if some successor of a position $x \in V_{\text{☆}}$ is contained in $\mathrm{Attr}_{\text{☆}}(B)$, then so is x, and

(3) if all successor of a spotion $x \in V_{\overline{\text{☆}}}$ is contained in $\mathrm{Attr}_{\text{☆}}(B)$, then so is x.

"Smallest set" means that we intersect over all subsets of V that satisfy the properties, i.e. more formally, $\mathrm{Attr}_{\text{☆}}(B)$ is definition to be the intersection over all $V' \subseteq V$ that satisfy the Properties (1) – (3).

If the game arena satisfies the conditions that we have imposed at the beginning of this section, the new definitions of the attractor coincides with the one from Definition 4.6. With the new definition of the attractor, the positional determinacy of reachability/safety games can be proven to hold even if the game arena has infinite outdegree. In this case, the attractor can be "computed" by continuing the iteration beyond all natural numbers, essentially using a concept called transfinite induction.

In the example, we could fix the problem by considering the union of all *i*-step attractors, that we will call $\mathrm{Attr}^{\omega}_{\star}$ in the following, and then doing one more step of the attractor computation.

$$
\begin{array}{rcl}
\mathrm{Attr}^{\omega}_{\star}(B) & = & \bigcup_{i \in \mathbb{N}} \mathrm{Attr}^{i}_{\star}(B) \\
\mathrm{Attr}^{\omega+1}_{\star}(B) & = & \mathrm{Attr}^{\omega}_{\star}(B) \\
 & \cup & \{x \in V_{\star} \mid \exists (x, y) \in R\colon y \in \mathrm{Attr}^{\omega}_{\star}(B)\} \\
 & \cup & \{x' \in V_{\overline{\star}} \mid \forall (x', y') \in R\colon y' \in \mathrm{Attr}^{\omega}_{\star}(B), x' \text{ is live}\}
\end{array}
$$

In general, even this could not be sufficient. We could not only need more steps of the attractor computation, but even more limits steps, steps in which we take the union over all smaller attractors.

Application: Multiprocessor online scheduling

We have now gathered the prerequisites to study a practical application of reachability / safety games. Online scheduling problems can be seen as a game where one player generates the tasks that have to be scheduled and the other player is the scheduler. The existence of a winning strategy for the scheduling player corresponds to the existence of a safe scheduler, a scheduler that guarantees that no job ever misses its deadline.

The content can be found in Section 14.

5. Büchi & coBüchi games

A play that satisfies the reachability winning condition may be infinite, but it essentially can be cut off after the position in which the winning set is reached. An infinite maximal play is winning for the existential player with respect to the winning condition if and only if it has a finite prefix in which a position in the winning set occurs.

Now we want to look at a winning condition that can not be checked by looking at prefixes. To satisfy the winning condition of **Büchi games**, positions in a winning set have to be visited infinitely often. For this reason, they are also called **recurrence games**.

The dual concept are **coBüchi games** or **persistence games**, in which a set of losing positions may be visited finitely many times, but not infinitely often.

Similar to the chapter on reachability and safety games, we assume that the existential player wants to satisfy the Büchi condition, while the universal player wants to prevent it.

Sources
The content of this section is based on Martin Zimmermann's notes [ZKW].

Other available resources for the topic include [CHP08; Kum; Jobb; Joba].

Büchi & coBüchi games

5.1 Definition
Let X be a set (finite or infinite). Let X^ω be the set of infinite sequences of elements in X, i.e.

$$X^\omega = \{f \mid f\colon \mathbb{N} \to X\} .$$

For such a sequence $p \in X^\omega$, we denote by Inf(p) the set of elements of X that occur in s infinitely often,

$$\begin{aligned}\mathrm{Inf}(p) &= \{x \in X \mid p_i = x \text{ for infinitely many } i \in \mathbb{N}\} \\ &= \{x \in X \mid \{i \mid p_i = x\} \text{ is infinite}\} \\ &= \{x \in X \mid \nexists k \in \mathbb{N}\colon |\{i \mid p_i = x\}| = k\}\end{aligned}$$

For finite sequences, we can set Inf(p) = ∅.

Note that we can see the set of infinite plays $Plays_{inf}$ of a game as a subset of V^ω, i.e. writing Inf(p) for an infinite play makes sense.

For the rest of this section, let $G = (V_\Box \uplus V_\bigcirc, R)$. We make two assumptions:

- V is finite (and thus R is finite, too).
- G contains no deadlock, i.e. all positions are live.

We have already discussed in Exercise 3.17 that the second assumption can usually be enforced easily by a minor tweaking of the game arena. This is in particular true for Büchi games. This assumption guarantees that all maximal plays are infinite, $Plays_{max} = Plays_{inf}$.

We comment on the first assumption after the crucial definition.

5.2 Definition: Büchi games
Let $B \subseteq V$ be a set of positions. The **Büchi game** or **recurrence game** on G with respect to the **winning set** B is the game with the winning condition

$$\begin{aligned} win \ : \ Plays_{max} &\to \{\bigcirc, \Box\} \\ p &\mapsto \begin{cases} \bigcirc & \text{, if } \mathrm{Inf}(p) \cap B \neq \emptyset \text{ ,} \\ \Box & \text{, else, i.e. } \mathrm{Inf}(p) \cap B = \emptyset \text{ .} \end{cases} \end{aligned}$$

As in the case of reachability and safety games, we can also see the above definition as the definition of the **coBüchi game** or **persistence game** with respect to the **losing set** B from the perspective of the universal player.

Let us now comment on the assumption that V is finite. If we allow infinitely many positions, then the set B can also be infinite. If B is finite, and a play p visits positions in B infinitely often, then there has to be a position $x \in B$ that is visited infinitely often, and we have $x \in \mathrm{Inf}(p) \cap B$. This is by a variant of the pigeonhole principle: We distribute infinitely many pigeons into finitely many holes.

If we would allow B to be infinite, we could have that infinitely many positions in p are in B, but no single position is visited infinitely often, $\mathrm{Inf}(p) \cap B = \emptyset$.

5.3 Remark
Büchi games are named after the Swiss mathematician Julius Richard Büchi. He introduced Büchi automata in 1962, automata that read infinite words. Their acceptance condition is that infinitely many final control states have to occur during a run.

Although he did not consider games, at least to my knowledge, this type of games is named after him due to the similarity of the winning condition to his acceptance condition.

Recurrence construction

The attractor construction again plays an important role in the solution of such games. Note that if a position is not in $\mathrm{Attr}_{\bigcirc}(B)$, it cannot be winning for the existential player, since the universal player can prevent each play from visiting B even once. There might be positions from which the existential player can enforce a first visit of B, but not a second one. Imagine for example that we reach a position of B that has a single successor not in B, in which the play then loops. This means that the winning region may be smaller than $\mathrm{Attr}_{\bigcirc}(B)$.

Our goal is to restrict the winning set of B to the positions from which a revisit of B can be enforced by the existential player. For each $i \in \mathbb{N}$, we define B^i as ***i*-revisits recurrence set**, the set of positions in B such that the existential player can enforce i revisits of B. We obtain that the **recurrence set** $\bigcap_{i \in \mathbb{N}} B^i$ is the set of positions in B from which arbitrarily many revisits to B can be enforced by the existential player.

Intuitively, the winning region of the Büchi game for the existential player should be the set of positions from which she can enforce reaching the recurrence set, i.e. $\mathrm{Attr}_{\bigcirc}(\bigcap_{i \in \mathbb{N}} B^i)$.

We formalize this **recurrence construction** in the following definition.

5.4 Definition: Recurrence construction
For all natural numbers i, the sets B^i and P^i of vertices are mutually inductive defined as.

$$\begin{aligned} B^0 &= B, \\ P^i &= V \setminus \mathrm{Attr}_{\bigcirc}(B^i), \\ B^{i+1} &= B \setminus \mathrm{CPre}_{\Box}(P^i). \end{aligned}$$

The set B^i is called the i^{th} **recurrence set**, the set P^i is called the i^{th} **persistence set**.

It might not be clear that the sets B^i are indeed the i-revisits recurrence sets that were mentioned before. It is possible to define the sets in a way that corresponds better to the intuition explained above. The definitions here were chosen because they simplify the proof. We will see that both definitions are equivalent in Exercise 5.13.

5.5 Lemma

The sets B^i form a descending, the sets P^i form an ascending chain. There is an index $m \in \mathbb{N}$ such that the chains simultaneously become stationary.

$$\begin{array}{ccccccccccccc} B & = & B^0 & \supseteq & B^1 & \supseteq & \ldots & \supseteq & B^m & = & B^{m+1} & = & \bigcap_{i\in\mathbb{N}} B^i \\ & & P^0 & \subseteq & P^1 & \subseteq & \ldots & \subseteq & P^m & = & P^{m+1} & = & \bigcup_{i\in\mathbb{N}} P^i \end{array}$$

Proof:

Proving $B^i \supseteq B^{i+1}$ and $P^i \subseteq P^{i+1}$ boils down to Lemma 4.8, Part a). In a finite arena, the chains have to become stationary, similar to Exercise 4.13. We leave the details to the reader as an exercise, Exercise 5.12. ■

5.6 Theorem: Solving Büchi games

For the Büchi game with respect to B, we have $W_\square = \bigcup_{i\in\mathbb{N}} P^i$, and $W_\bigcirc = V \setminus W_\square$.

Towards a proof of the theorem, let $X = V \setminus \bigcup_{i\in\mathbb{N}} P^i$ be the set of positions that we claim is the winning region of the existential player.

We prove that all vertices in X are winning for the existential player, $X \subseteq W_\bigcirc$, and we prove that the vertices not in X are winning for the universal player, $V \setminus X \subseteq W_\square$. Because no position can be winning for both players, Lemma 3.9, this proves the claim.

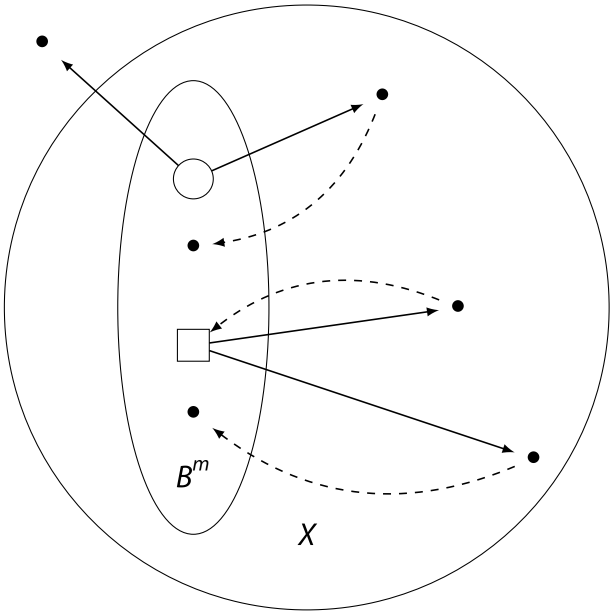
By Lemma 5.5, there is an index $m \in \mathbb{N}$ such that $B^m = B^{m+1}$ and $P^m = P^{m+1}$. This means $\bigcup_{i\in\mathbb{N}} P^i = P^m$.

Furthermore, as previously claimed, we have $X = \mathrm{Attr}_\bigcirc(B^m)$, because

$$\begin{aligned} X &= V \setminus \bigcup_{i\in\mathbb{N}} P^i \\ &= V \setminus P^m \\ &= V \setminus \big(V \setminus \mathrm{Attr}_\bigcirc(B^m)\big) \\ &= \mathrm{Attr}_\bigcirc(B^m). \end{aligned}$$

Let us first show that X is indeed winning for the existential player. Before we state the strategy, we prove the following Lemma. It states that from B^m, the existential player can enforce reaching X within one step.

After proving it, we have collected all ingredients that we need to state the winning strategy and show that it is indeed winning. The concept of the winning strategy is shown in the following picture.



Whenever the current position is in the set X that we claim to be the winning region, the existential player can enforce a visit of B^m within finitely many steps. This is because $X = \mathrm{Attr}_{\bigcirc}(B^m)$. In the picture, this is symbolized by the dashed, bend lines. Whenever the play reaches B^m, the existential player can enforce that we stay in X in the next step. This is the statement of the next lemma, and symbolized by the straight solid lines in the picture.

Note that $B^m \subseteq B$ is a subset of winning positions. Any play that follows the strategy outlined above visits $B^m \subseteq B$ infinitely often, and thus is winning.

5.7 Lemma
$B^m \subseteq \mathrm{CPre}_{\bigcirc}(X)$.

Proof:
Let $x \in B^m$ be arbitrary. We have $x \in B^m = B^{m+1} = \big(B \setminus \mathrm{CPre}_{\Box}(P^m)\big)$ by definition.

If x is owned by the existential player, $x \in V_{\bigcirc}$, x has a successor y not in P^m. Because $P^m = P^{m+1}$, we have that y is not in any P^j and thus by definition, $y \in X$.

If x is owned by the universal player, $x \in V_{\Box}$, all successors are not in P^m, and thus not in any P^j, and thus in X.

In both cases, we conclude $x \in \mathrm{CPre}_{\bigcirc}(X)$. ■

Recall that $X = \mathrm{Attr}_{\bigcirc}(B^m)$. By Lemma 4.11, there is a positional strategy $s_{\bigcirc}^{Attr}$ that, if played from a position in X, reaches $B^m \subseteq B$ within finitely many steps.

We construct a winning strategy $s_{\bigcirc}$ for the existential player as follows.

$$\begin{array}{rcll} s_{\bigcirc} & : & V_{\bigcirc} & \rightarrow V \\ & & x & \mapsto \begin{cases} s_{\bigcirc}^{Attr}(x), & \text{if } x \in \mathrm{Attr}_{\bigcirc}(B^m) \setminus B^m, \\ y \text{ with } (x,y) \in R \text{ and } y \in X, & \text{if } x \in B^m, \\ y \text{ with } (x,y) \in R \text{ arbitrary }, & \text{else.} \end{cases} \end{array}$$

In the first case, note that we apply $s_{\bigcirc}^{Attr}$ to a position in $\mathrm{Attr}_{\bigcirc}(B^m) \setminus B^m \subseteq \mathrm{Attr}_{\bigcirc}(B^m) = X$, i.e. to a position for which it guarantees to reach B^m after finitely many steps.

Furthermore note that in the second case, i.e. $x \in B^m$, we know that there is a successor in X, because we have shown $x \in B^m \subseteq \mathrm{CPre}_{\bigcirc}(X)$ in Lemma 5.7.

5.8 Lemma
$s_{\bigcirc}$ is a positional strategy for the existential player that is uniformly winning from all positions in X.

Proof:
Let $p = p_0p_1p_2\ldots$ be an arbitrary maximal play from some position $p_0 \in X$ that is conform to $s_{\bigcirc}$.

We first show that p never leaves X, i.e. $\forall i \in \mathbb{N}$: $p_i \in X$. We proceed by induction, where the base case is by the assumption $p_0 \in X$.

Assume that p_j is in X. We distinguish the two cases that are also distinguished by our strategy.

If $p_j \in X \setminus B^m$ and p_j is owned by the existential player, the existential player uses the strategy $s_{\bigcirc}^{Attr}$. Note that the strategy from Lemma 4.11 in particular guarantees that all moves stay within the attractor, i.e. the successor y picked by the strategy satisfies $y \in X$. If p_j is owned by the opponent the universal player, she cannot leave $\mathrm{Attr}_{\bigcirc}(B^m)$ as all successors are in X by the definition of the attractor. Note that for both cases, it is important that we are not in the 0-step attractor $\mathrm{Attr}_{\bigcirc}^0(B^m) = B^m$.

Now assume that $p_j \in B^m$. If p_j is owned by the existential player, the strategy picks a successor in X, and we have already argued that this is always possible. If p_j is owned by the universal player, we know that all successors are in X, since we have argued that $B^m \subseteq \mathrm{CPre}_{\bigcirc}(X)$. In both cases, we rely on Lemma 5.7.

To finish the proof, we still need to argue that B is visited infinitely often. Assume that $i \in \mathbb{N}$ is some index such that $p_i \notin B$. Because $B \supseteq B^m$, this implies $p_i \in X \setminus B^m = \mathrm{Attr}_{\bigcirc}(B^m) \setminus B^m$. On such a position, the strategy $s_{\bigcirc}$ behaves as the strategy $s_{\bigcirc}^{Attr}$ that guarantees reaching B^m. In particular, there is some number $k \in \mathbb{N}$ such that $p_{i+k} \in B^m \subseteq B$.

This shows that whenever we are not in B, we reach B again after finitely many moves. Overall, we visit B infinitely often. ■

This finishes our proof of $X \subseteq W_\bigcirc$. We now consider the case of the universal player, showing that

$$V \setminus X = V \setminus \left(V \setminus \bigcup_{i \in \mathbb{N}} P^i \right) = \bigcup_{i \in \mathbb{N}} P^i = P^m$$

is a subset of $W_\square$. To this end, we define a function δ that maps each vertex $x \in P^m$ to a natural number $\delta(x)$ such that any play from x conform to the strategy – which we will present later – visits vertices in B at most $\delta(x)$ many times. In particular, only finitely many visits may occur. The function is defined as follows:

$$\begin{array}{rcccl} \delta & : & P^m & \to & \mathbb{N} \\ & & x & \mapsto & \min\{i \in \mathbb{N} \mid x \in P^{\,i}\} \end{array}$$

Note that we only consider vertices $x \in P^m$, so we have that $\delta(x) \leqslant m$ for all positions x.

Before formally defining the strategy, we prove some properties of δ that will be crucial for the well-definedness of the strategy.

5.9 Lemma

a) For all $x \in P^m \cap B$, we have $\delta(x) > 0$.

b1) For all $x \in P^m \cap V_\square$, there is a successor y such that $\delta(x) \geqslant \delta(y)$.

b2) If additionally $x \in B$, the inequality from b1) is strict, $\delta(x) > \delta(y)$

c1) For all $x \in P^m \cap V_\bigcirc$, and all successors y, $\delta(x) \geqslant \delta(y)$ holds.

c2) If additionally $x \in B$, the inequality from c1) is strict, $\delta(x) > \delta(y)$

Proof:

a) Let $x \in P^m \cap B$. To show $\delta(x) > 0$, we need to argue that $x \notin P^0$. By the definition, we have $P^0 = V \setminus \text{Attr}_\bigcirc(B)$, i.e. no vertex in P^0 is in the attractor of B. Certainly $B \subseteq \text{Attr}_\bigcirc(B)$ holds, so we indeed get $B \cap P^0 = \emptyset$.

b1) Assume that for all successors y, we have $\delta(y) > \delta(x)$. In particular, we have $y \notin P^{\delta(x)}$ for all successors. This proves that all successors $y \in V \setminus P^{\delta(x)} = \text{Attr}_\bigcirc(B^{\delta(x)})$ are in the attractor of $B^{\delta(x)}$. Consequently, $x \in \text{CPre}_\bigcirc\big(\text{Attr}_\bigcirc(B^{\delta(x)})\big) \subseteq \text{Attr}_\bigcirc(B^{\delta(x)})$ since attractors are closed under taking the controlled predecessor by Lemma 4.8, Part b). This is a contradiction, since $x \in P^{\delta(x)} = V \setminus \text{Attr}_\bigcirc(B^{\delta(x)})$.

b2) Assume that additionally, we have $x \in B$. By Part a) of the lemma, we have $\delta(x) > 0$ and thus $x \in P^{\delta(x)}, x \notin P^{\delta(x)-1}$, where $\delta(x) - 1$ is a natural number.

We have $x \in V \setminus \mathrm{Attr}_{\bigcirc}(B^{\delta(x)})$, and therefore $x \notin \mathrm{Attr}_{\bigcirc}(B^{\delta(x)})$. This in particular implies $x \notin B^{\delta(x)} = B \setminus \mathrm{CPre}_{\square}\left(P^{\delta(x)-1}\right)$.

Since we assume $x \in B$, we get $x \in \mathrm{CPre}_{\square}\left(P^{\delta(x)-1}\right)$. By the definition of CPre, there has to be a successor $y \in P^{\delta(x)-1}$, i.e. with $\delta(y) \leqslant \delta(x) - 1 < \delta(x)$.

c) Dual to b1) and b2), see Exercise 5.14.

■

We can now formally define the strategy $s_{\square}$ as follows.

$$\begin{aligned} s_{\square} \;:\; V_{\square} &\;\to\; V \\ x &\;\mapsto\; \begin{cases} y \text{ with } (x,y) \in R \text{ and } \delta(x) > \delta(y), & \text{if } x \in P^m \cap B, \\ y \text{ with } (x,y) \in R \text{ and } \delta(x) \geqslant \delta(y), & \text{if } x \in P^m \setminus B, \\ y \text{ with } (x,y) \in R \text{ arbitrary }, & \text{else.} \end{cases} \end{aligned}$$

The successors picked in the first resp. second case are guaranteed to exist by Lemma 5.9, Part b).

A play that conforms to $s_{\square}$ guarantees that the

- the δ-values do not increase along the play, and
- whenever the play visits B, they strictly decrease.

This is by the definition of the strategy and by Lemma 5.9, Part c).

Since the value is initially at most m, and it stays non-negative, the set B is visited at most m times, in particular only finitely often.

5.10 Lemma
$s_{\square}$ is a positional strategy for the universal player that is uniformly winning from all positions in P^m.

Proof:
Let $p = p_0p_1p_2\ldots$ be an infinite play that is conform to $s_{\square}$.

It is easy to see that for all $i \in \mathbb{N}$, we have $\delta(p_i) \geqslant \delta(p_{i+1})$, i.e. we have an infinite decreasing chain

$$\delta(p_0) \geqslant \delta(p_1) \geqslant \delta(p_2) \geqslant \delta(p_3) \geqslant \ldots$$

For positions owned by the existential player, this is by Lemma 5.9, Part c1), for positions owned by the universal player, this is by the definition of the strategy.

Assume that p is not winning for $\square$, meaning that there are infinitely many i such that $p_i \in B$, say $i_0, i_1, i_2, \ldots$. Again by Part c2) of Lemma 5.9 and by the definition of the strategy, we then obtain an infinite strictly decreasing chain

$$\delta(p_{i_0}) > \delta(p_{i_1}) > \delta(p_{i_2}) > \delta(p_{i_3}) > \ldots$$

Since we have $\delta(p_i) \in \mathbb{N}$ for all i, we get a contradiction. All strictly decreasing chains of natural numbers have to be finite. ■

Together, the Lemmata 5.8 and 5.10 prove Theorem 5.6. Since the strategies are positional and uniformly winning, they even prove the following corollary.

5.11 Corollary: Positional determinacy of Büchi games
Büchi games are positionally determined: The set of positions can be partitioned into the winning regions for each of the players, and each player has a uniform positional winning strategy for her winning region.

Exercises

5.12 Exercise
Formally prove Lemma 5.5.

In the next exercise, we give a more intuitive definition of the recurrence sets B^i, and we prove that it is equivalent to Definition 5.4.

5.13 Exercise: A more intuitive definition of recurrence sets
For the definition, we need a slightly modified attractor construction:

$$\begin{aligned} A^0_{☆}(X) &= \varnothing \\ A^{i+1}_{☆}(X) &= A^i_{☆}(X) \cup \mathrm{CPre}_{☆}\big(A^i_{☆}(X) \cup X\big) \\ \mathrm{Attr}^+_{☆}(X) &= \bigcup_{i \in \mathbb{N}} A^i_{☆}(X) \end{aligned}$$

Now we give an alternative definition for the sets B^i, here called B^i_{ex}:

$$\begin{aligned} B^0_{ex} &= B \\ B^{i+1}_{ex} &= B \cap \mathrm{Attr}^+_{\bigcirc}\big(B^i_{ex}\big) \end{aligned}$$

a) Describe the difference between $\mathrm{Attr}^{+}_{\text{☆}}(B)$ and $\mathrm{Attr}_{\text{☆}}(B)$ in your own words.

b) Formally prove using induction on i that $A^{i}_{\text{☆}}(B) \cup B = \mathrm{Attr}^{i}_{\text{☆}}(B)$ for all $i \in \mathbb{N}$ and conclude $\mathrm{Attr}^{+}_{\text{☆}}(B) \cup B = \mathrm{Attr}_{\text{☆}}(B)$.

c) Formally prove using induction on i that $B^{i} = B^{i}_{ex}$ for all $i \in \mathbb{N}$.

Hint: In the induction step, you essentially need to prove

$$V \setminus \mathrm{Attr}^{+}_{\bigcirc}(B^{i}) = \mathrm{CPre}_{\Box}\Big(V \setminus \mathrm{Attr}_{\bigcirc}(B^{i})\Big)\,.$$

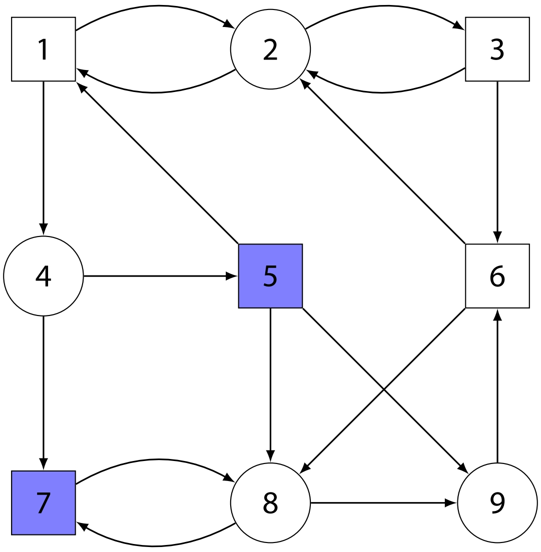
Part b) of this exercise is crucial for proving this statement.

5.14 Exercise

Prove Lemma 5.9, Part c1) and c2).

5.15 Exercise: A Büchi game

Consider the following game arena. As usual, vertices of the universal player are drawn as boxes, those of the existential player as circles.



Consider the Büchi game with respect to the winning set $\{5, 7\}$, i.e. the existential player wants to visit the blue-colored vertices infinitely often.

Solve the Büchi game using the recurrence construction. Give the sets B^{i}, P^{i} for all i, and give all sets $\mathrm{Attr}_{\bigcirc}$ and $\mathrm{CPre}_{\Box}$ that are needed to compute them.

6. Parity games

Our goal in this section is to generalize Büchi games to **parity games**. Similar to Büchi games, the winning condition for Parity games will be a condition on infinite plays that can not be decided by looking at a finite prefix. The winning condition of Parity games allows us to express more involved properties like the following.

- If position x is visited infinitely often, then position y also has to be visited infinitely often.
- If position x is visited infinitely often, then position y should not be visited infinitely often.

Parity games have important applications:

- The model checking problem for certain kinds of logics can be expressed as a parity game. For the modal μ-calculus, parity games are equivalent to the model checking problem.
- **Rabin's tree theorem**, a deep result on the closure properties of a certain class of tree languages, can be proven by using the positional determinacy of parity games. Rabin's tree theorem in turn is used to prove the decidability of MSO logic over infinite trees. We will prove Rabin's tree theorem in the Section 15.
- The emptiness problem for certain types of automata (alternating automata, tree automata) can be solved by solving a parity game.

Furthermore, parity games are an interesting problem in complexity theory; We will discuss this in more detail later.

Sources
The content of this section is loosely based on Roland Meyer's notes on the topic.
They can be found here:
35_parity_tree_automata_part_1.pdf
36_parity_tree_automata_part_2.pdf

Parity games

6.1 Definition: Parity game

A **parity game** is given by a game arena $G = (V_\square \uplus V_\bigcirc, E)$ and a function $\Omega \colon V \to \{0, \ldots, k\}$ for some $k \in \mathbb{N}$ that assigns each position one of finitely many **priorities** (also called **colors**)

6.2 Assumption
We assume that G is deadlock-free and that each position has only finitely many successors. We do not assume that V itself is finite.

Because the game arena is deadlock-free, again each maximal play p has to be infinite. The parity winning condition is satisfied depending on the highest priority that occurs infinitely often in B.

We formalize this as follows: A maximal play $p = p_0p_1p_2\dots$ defines a sequence

$$\Omega(p) = \Omega(p_0)\Omega(p_1)\Omega(p_2)\dots \in \{0,\dots,k\}^{\omega}\,.$$

By the pigeon hole principle, $\{0,\dots,k\}$ being finite implies that $\mathrm{Inf}(\Omega(p))$ is non-empty. We are interested in $\max \mathrm{Inf}(\Omega(p))$, the highest priority occurring infinitely often. By convention, even numbers are good for the existential player, odd numbers are good for the universal player.

6.3 Definition: Parity winning condition
The **parity winning condition** for the parity game given by the game arena $G = (V_{\square} \uplus V_{\bigcirc}, E)$ and the priority function Ω is given by

$$\begin{aligned} win \;:\; Plays_{max} \;&\to\; \{\bigcirc,\square\} \\ p \;&\mapsto\; \begin{cases} \bigcirc, & \text{if } \max \mathrm{Inf}(\Omega(p)) \text{ is even}, \\ \square, & \text{else, i.e. if } \max \mathrm{Inf}(\Omega(p)) \text{ is odd}. \end{cases} \end{aligned}$$

6.4 Example
Consider the Büchi game on some game arena G with respect to the winning set B. We can see it as the parity game on G with the priority function

$$\Omega(x) = \begin{cases} 2, & x \in B, \\ 1, & x \notin B. \end{cases}$$

This example shows that parity games indeed generalize Büchi games. No matter how large the graph is, we just need maximal priority 2 to encode the Büchi winning condition.

Just like the Büchi winning condition (but unlike the reachability condition), the parity winning condition is not depending on finite prefixes of the play.

6.5 Lemma: Winning is prefix-independent
Let $p = p'.p'' \in V^\omega$ be an infinite play that decomposes into a finite prefix p' and an infinite play p''. We have $win(p) = win(p'')$.

Proof: $\mathrm{Inf}(\Omega(p)) = \mathrm{Inf}(\Omega(p''))$. ■

We can see this lemma in two ways: On the one hand, we can cut off a finite prefix of a play without changing its winner. On the other hand, we can prepend a finite prefix to a play without changing its winner.

The consequence of this is that positional winning strategies from different positions can be combined to a single uniform positional winning strategy. We have seen in Exercise 3.15 that this is not true for arbitrary winning conditions.

6.6 Lemma

a) Let $x, x' \in V$ be positions such that player $\star \in \{\bigcirc, \square\}$ has positional winning strategies $s_{\star,x}$ resp. $s_{\star,x'}$ winning from x resp. x'. Then there is a positional strategy $s_\star$ that is winning from both x and x'.

b) Let X be a set of positions such that for each $x \in X$, $\star \in \{\bigcirc, \square\}$ has a positional strategy $s_{\star,x}$ that is winning from x. Then there is a positional strategy $s_\star$ that is uniformly winning from all positions $x \in X$.

Proof:
We prove a), Part b) is Exercise 6.25.

Let $P \subseteq V^\omega$ be the set of all plays from x that conform to $s_{\star,x}$. We define $Y \subseteq V$ to be the set of all positions that occur in such plays,

$$Y = \{y \in V \mid \exists p \in P \exists i \in \mathbb{N} \colon p_i = y\} .$$

By the previous Lemma 6.5, we know that $s_{\star,x}$ is not only winning from x, but also from any position in Y: Any play from a position $y \in Y$ can be seen as the suffix of a play from x.

We define a strategy $s_\star$ as follows:

$$\begin{aligned} s_\star \ : \ V_\star &\to V \\ z &\mapsto \begin{cases} s_{\star,x}(z) & \text{if } z \in Y, \\ s_{\star,x'}(z) & \text{else.} \end{cases} \end{aligned}$$

Intuitively, $s_{☆}$ imitates $s_{☆,x'}$ until the play visits Y. Afterwards, it behaves like $s_{☆,x'}$.

It remains to prove that $s_{☆}$ is indeed winning from both x and x'.

To this end, one should first prove that if a play visits a position in Y, from then on all positions in the play will be contained in Y, and thus $s_{☆}$ will behave like $s_{☆,x}$. This can be easily done by induction, and we leave it to the reader as an additional exercise.

Since $x \in Y$, consequently each play from x that conforms to $s_{☆}$ also conforms to $s_{☆,x}$. Since $s_{☆,x}$ was a winning strategy, the play is then won by player ☆.

Any play p from x' that conforms to $s_{☆}$ will either never visit Y, or there is a smallest index $i \in \mathbb{N}$ such that $p_i \in Y$. In the first case, the play is also conform to $s_{☆,x'}$, and thus winning. In the second case, the play can be decomposed into $p = p_0 \dots p_{i-1}p_ip_{i+1} \dots$, where $p_ip_{i+1} \dots$ is an infinite play from $p_i \in Y$ that is conform to $s_{☆,x}$ and thus winning. By the Lemma 6.5, prepending the prefix $p_0 \dots p_{i-1}$ does not influence the winner of the play. ■

As a result of the Lemma, it is sufficient to show that for each position, exactly one of the players has a positional winning strategy. The lemma then gives us that there are uniform positional winning strategies for both players on their respective winning region.

Zielonka's proof of positional determinacy

The goal of this section is to prove the following result.

6.7 Theorem: Positional determinacy of parity games [Mos91; EJ91; Zie98]
Parity games are positionally determined: There is a decomposition of the positions into the winning regions of the players, $V = W_{\square} \uplus W_{\bigcirc}$, and each player has a uniform positional winning strategy on her winning region.

6.8 Remark: History of parity games
The determinacy of parity games can be proven using the Borel determinacy theorem (Martin 1975 [Mar75]). However, this proof approach is non-constructive and neither gives an algorithm to compute a winner nor information on the type of strategies needed.

In 1982, Gurevich and Harrington [GH82] have proven that it is possible for the winner to win with a strategy that uses only finite memory. However, their proof is non-constructive and does not immediately result in an algorithm to compute the winner of a game. In 1993, McNaughton [McN93] presented an algorithm that can compute the winner for parity games on a finite graph.

The positional determinacy of parity games was independently proven in 1991 by Mostowski [Mos91] and Emerson and Jutla [EJ91]. The proof and the recursive algorithm that we will present here are due to Zielonka [Zie98].

In the proof, we will restrict the game arena to obtain a so-called subgame. To formalize this, we will need the following definition.

6.9 Definition: Trap

We call a set $X \subseteq V$ a **trap** for player $☆ \in \{○, □\}$ if

- for all positions $x \in X$ owned by player ☆, all successors are in X, and
- all positions $x \in X$ owned by the opponent $\overline{☆}$ have at least one successor in X.

The intuition behind this definition is that whenever the play visits a trap X for player ☆, the opponent $\overline{☆}$ can trap the play inside X. This means that player ☆ cannot enforce that the play will ever leave X if the opponent does not cooperate.

The conditions should sound awfully familiar to the definition of the controlled predecessors. Therefore, the first part of the next lemma should not be surprising.

6.10 Lemma

a) Let $Y \subseteq V$ and $☆ \in \{○, □\}$. The complement of the attractor $V \setminus \mathrm{Attr}_{☆}(Y)$ is a trap for player ☆.

b) Let $P_{☆} \subseteq V$ be the set of positions such that for each $x \in P_{☆}$, ☆ has a positional winning strategy from x. Then its complement $V \setminus P_{☆}$ is a trap for player ☆.

Proof:

a) Obvious by the definition of the attractor; See Exercise 6.26.

b) By Lemma 6.6, there is a uniform positional winning strategy on $P_{☆}$ for player ☆.

Assume that $x \in V \setminus P_{☆}$.

If x is owned by the player ☆, and x would have a successor in $P_{☆}$, then the positional winning strategy on $P_{☆}$ could be extended to a positional winning strategy on $P_{☆} \cup \{x\}$ by picking this successor. This means $x \in P_{☆}$, a contradiction. Consequently, for positions owned by ☆, all successors are in $V \setminus P_{☆}$.

If x is owned by the opponent $\overline{☆}$, but all its successors are in $P_{☆}$, then the positional winning strategy on $P_{☆}$ is also winning from x. Consequently, at least one successor is not in $P_{☆}$.

■

Traps are important because one can restrict a deadlock-free game arena to a trap and again obtain a deadlock-free game arena.

6.11 Definition
Let $\mathcal{G}$ be the parity game given by the game arena $G = (V_{\Box} \uplus V_{\bigcirc}, R)$ and the priority function Ω, and let $X \subseteq V$ be a trap for player ☆.

We define $\mathcal{G}_{\restriction X}$ to be the parity game on the game arena

$$G_{\restriction X} = ((V_{\Box} \cap X) \uplus (V_{\bigcirc} \cap X), \{(x, y) \in R \mid x, y \in X\})$$

with respect to the restricted priority function $\Omega_{\restriction X}$.

6.12 Lemma
The subgame $\mathcal{G}_{\restriction X}$ with respect to a trap is deadlock-free.

Proof: Immediate by the definition of trap and the assumption that the original game was deadlock-free. ■

In the definition of the subgame, it did not matter for which player the set X is a trap. This is important for the following lemma.

6.13 Lemma
Let $X \subseteq V$ be a trap for player ☆ in $\mathcal{G}$ and let $s_{\overline{☆}}$ be a strategy for the opponent $\overline{☆}$ that is winning from some vertex $x \in X$ in the subgame $\mathcal{G}_{\restriction X}$. Then $s_{\overline{☆}}$ is also winning from x in the original game $\mathcal{G}$.

Proof: Exercise 6.27. ■

We have now gathered all prerequisites, and turn to proving the Theorem.

Proof of Theorem 6.7:
We proceed by induction on the highest occurring number n that occurs as the priority of one of the positions.

Base case, $n = 0$**:**
In the base case, we assume that $n = 0$. Any play of such a game will by won be the existential player. Thus, any positional strategy for the existential player will be winning.

Induction step:
Now we assume that the statement already holds for games where the highest occurring priority is $n - 1$. Let $\mathcal{G}$ be the given parity game where n is the highest occurring priority.

Without loss of generality, we assume that n is even. If this is not the case, one has to swap the roles of the players in the following proof.

Let $P_\Box \subseteq V$ be the set of positions from which the universal player has a positional winning strategy. Obviously, $P_\Box$ is a subset of the universal player's winning region $W_\Box$.

We show that for each position in the complement $V \setminus P_\Box$, the existential player has a positional winning strategy. By Lemma 6.6, we then get the existence of uniform positional winning strategies for both sets, which proves the theorem.

Consider the subgame $\mathcal{G}' = \mathcal{G}_{\restriction V \setminus P_\Box}$. By Lemma 6.10, Part b), $V \setminus P_\Box$ is a trap for the universal player, and thus $\mathcal{G}'$ is a deadlock-free parity game.

We distinguish two cases:

Case 1: Highest priority n **does not occur in subgame.**
This means there is no position $x \in V \setminus P_\Box$ with $\Omega(x) = n$.

In this case, we can apply the induction hypothesis to $\mathcal{G}'$ and we get that its set of vertices decomposes into the winning region for the two players,

$$V \setminus P_\Box = W'_\Box \uplus W'_\bigcirc \,,$$

and the two players have positional winning strategies on their respective winning region in the subgame.

If $W'_\Box$ is not empty, then there is a vertex $x \in W'_\Box$ such that the universal player has a positional strategy $s_\Box$ from x in the subgame. This strategy can be extended to a strategy for the original game $\mathcal{G}$ by using the strategy on $P_\Box$ that we have by the definition of $P_\Box$. This combined strategy is winning on $P_\Box \uplus \{x\}$ since the parity winning condition is prefix-independent, Lemma 6.5. We conclude $x \in P_\Box$, a contradiction.

Consequently, we have $W'_\Box = \emptyset$ and thus $W'_\bigcirc = V \setminus P_\Box$. By Lemma 6.13, the winning strategy for the existential player in the subgame is also a winning strategy for the existential player in the original game, since $V \setminus P_\Box$ is a trap for the universal player.

Case 2: Highest priority occurs in subgame.
We define N to be the non-empty set of positions of the subgame with priority n,

$$N = \{x \in V \setminus P_\Box \mid \Omega(x) = n\}\,.$$

Now consider the attractor $\mathrm{Attr}_\bigcirc^{\mathcal{G}'}(N)$ in $V \setminus P_\Box$. We again construct a subgame $\mathcal{G}'' = \mathcal{G}'_{\restriction V \setminus P_\Box \setminus \mathrm{Attr}_\bigcirc(N)}$. Here, $V \setminus P_\Box \setminus \mathrm{Attr}_\bigcirc(N)$ should be read as $(V \setminus P_\Box) \setminus \mathrm{Attr}_\bigcirc(N)$.

By Lemma 6.10, Part a), this is indeed a deadlock-free parity game, since $V \setminus P_\Box \setminus \mathrm{Attr}_\bigcirc^{\mathcal{G}'}(N)$ is a trap for the existential player.

Because $N \subseteq \mathrm{Attr}_\bigcirc^{\mathcal{G}'}(N)$, $\mathcal{G}''$ does certainly not contain the highest priority n. We can apply induction to get that its set of positions decomposes into the winning regions of the two players

$$(V \setminus P_\Box) \setminus \mathrm{Attr}_\bigcirc^{\mathcal{G}'}(N)) = W''_\Box \uplus W''_\bigcirc\,,$$

and each player has a positional winning strategy on her respective winning region.

Similar to before, a winning strategy for the universal player on $W''_\Box$ could be extended to a winning strategy for the original game: Since $(V \setminus P_\Box) \setminus \mathrm{Attr}_\bigcirc^{\mathcal{G}'}(N)$ is a trap for $\bigcirc$ (inside $V \setminus P_\Box$), a winning strategy for $\Box$ in $\mathcal{G}''$ is also a winning strategy for $\Box$ in $\mathcal{G}'$ by Lemma 6.13. It could be combined with the positional winning strategy on $P_\Box$. Altogether, we obtain that if $x \in W''_\Box$, then $x \in P_\Box$, a contradiction to $W'_\Box \subseteq (V \setminus P_\Box) \setminus \mathrm{Attr}_\bigcirc^{\mathcal{G}'}(N) \subseteq V \setminus P_\Box$. Hence $W''_\Box = \emptyset$.

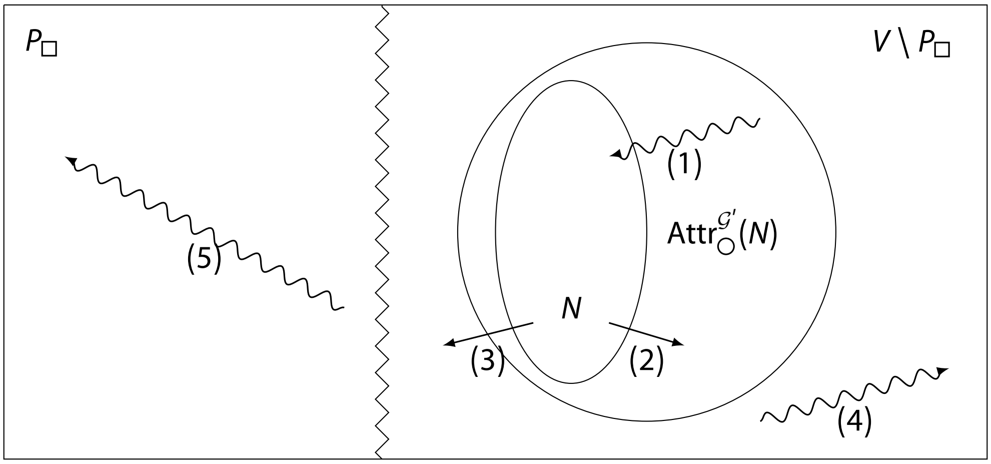
It remains to argue that the existential player has a positional winning strategy on $W''_\bigcirc \cup \mathrm{Attr}_\bigcirc^{\mathcal{G}'}(N) = V \setminus P_\Box$. To this end, we define a positional strategy.

Let s_{Attr} be the attractor strategy for the existential player that is winning for the reachability game in $V \setminus P_\Box$ with respect to the winning set N on $\mathrm{Attr}_\bigcirc^{\mathcal{G}'}(N)$. Note that it will also enforce reaching $\mathrm{Attr}_\bigcirc^{\mathcal{G}'}(N)$ in V itself, since $V \setminus P_\Box$ was a trap for the universal player.

Let $s_{\mathcal{G}''}$ be the winning strategy for the existential player for the parity game $\mathcal{G}''$ on the set $V \setminus P_\Box \setminus \mathrm{Attr}_\bigcirc^{\mathcal{G}'}(N)$. We combine the two strategies and define a positional winning strategy for the existential player for the original game $\mathcal{G}$ as follows.

$$\begin{aligned} s_\bigcirc \;:\; V_\bigcirc &\to V \\ x &\mapsto \begin{cases} s_{\mathrm{Attr}}(x)\,, & \text{if } x \in \mathrm{Attr}_\bigcirc^{\mathcal{G}'}(N) \setminus N, \qquad (1)\\ y \text{ with } y \in \mathrm{Attr}_\bigcirc^{\mathcal{G}'}(N)\,, & \text{if } x \in N \text{ and such a successor } y \text{ exists}\,, \qquad (2)\\ y \text{ with } y \in V \setminus P_\Box\,, & \text{if } x \in N \text{ and no succ } y \text{ as in (2) exists}, \qquad (3)\\ s_{\mathcal{G}''}(x)\,, & \text{if } x \in V \setminus P_\Box \setminus \mathrm{Attr}_\bigcirc^{\mathcal{G}'}(N), \qquad (4)\\ y \text{ arbitrary successor}\,, & \text{else.} \qquad (5) \end{cases} \end{aligned}$$

The strategy is illustrated by the following figure.



For the well-definedness of $s_\bigcirc$, we need to argue that each vertex in $x \in N$ has a successor in $V \setminus P_\Box$. This is because if all successors of a position $x \in N$ would be in $P_\Box$, then x would be winning for the universal player and thus be contained in $P_\Box$, but $N \subseteq V \setminus P_\Box$ by definition.

It remains to argue that $s_\bigcirc$ is winning on $V \setminus P_\Box$.

Let $p = p_0 p_1 p_2 \ldots$ be a play that conforms to $s_\bigcirc$ with $p_0 \in V \setminus P_\Box$. By the definition of $s_\bigcirc$, p never visits $P_\Box$.

If p visits $\mathrm{Attr}^{\mathcal{G}'}_{\bigcirc}(N)$ infinitely often, then p also visits N infinitely often and is indeed won by the existential player.

Let us assume that p visits N only finitely often. This also means there is a last visit of $\mathrm{Attr}^{\mathcal{G}'}_{\bigcirc}(N)$ in p, because after each visit of the attractor, a visit of N follows after finitely many steps. We can split the play $p = p'.p''$ such that p'' does not visit $\mathrm{Attr}^{\mathcal{G}'}_{\bigcirc}(N)$. By the prefix independence, Lemma 6.5, it is sufficient to show that p'' is winning for the existential player.

In p'' the existential player behaves as given by the strategy $s_{\mathcal{G}''}$. As argued before, the universal player cannot force the play to visit $P_\Box$. If the universal player forces the play to visit $\mathrm{Attr}^{\mathcal{G}'}_{\bigcirc}(N)$, this is a contradiction to p'' not visiting this set. Therefore, the play p'' stays inside $V \setminus P_\Box \setminus \mathrm{Attr}^{\mathcal{G}'}_{\bigcirc}(N)$. Thus, it can be seen as a play of the subgame $\mathcal{G}''$ that is conform to the winning strategy $s_{\mathcal{G}''}$, proving that it is won by the existential player.

■

6.14 Remark

We could actually define the winning strategy by

$$\begin{aligned} s_{\bigcirc} \; : \; V_{\bigcirc} & \to V \\ x & \mapsto \begin{cases} s_{\mathrm{Attr}}(x)\,, & \text{if } x \in \mathrm{Attr}_{\bigcirc}^{\mathcal{G}'}(N) \\ s_{\mathcal{G}''}(x) & \text{if } x \in V \setminus P_{\square} \setminus \mathrm{Attr}_{\bigcirc}^{\mathcal{G}'}(N), \\ y \text{ arbitrary successor}\,, & \text{else.} \end{cases} \end{aligned}$$

in the proof. Since $s_{\mathrm{Attr}}(x)$ is a strategy for the game $\mathcal{G}'$ whose set of positions is $V \setminus P_{\square}$, this strategy will also ensure that $P_{\square}$ is never visited. The proof of correctness works without modification. We chose to make the additional case distinction in the proof for didactic reasons.

Zielonka's algorithm

If the game arena is finite, the winning regions can be computed by the following recursive algorithm due to Zielonka [Zie98]. It is a modified version of the McNaughton's algorithm for solving Muller games [McN93].

6.15 Algorithm: Zielonka's recursive algorithm
Input: parity game $\mathcal{G}$ given by $G = (V_\Box, V_\bigcirc, R)$ and Ω.
Output: winning regions $W_\Box$ and $W_\bigcirc$.

```
Procedure solve($\mathcal{G}$)
$n = \max_{x \in V} \Omega(x)$
if $n = 0$ then
    return $W_\bigcirc = V, W_\Box = \varnothing$
else
    $N = \{x \in V \mid \Omega(x) = n\}$
    if $n$ even then
        $\star = \bigcirc, \overline{\star} = \Box$
    else
        $\star = \Box, \overline{\star} = \bigcirc$
    end if
    $A = \mathrm{Attr}^{\mathcal{G}}_{\star}(N)$
    $W'_\bigcirc, W'_\Box = solve(\mathcal{G}{\restriction}_{V \setminus A})$
    if $W'_\star = V \setminus A$ then
        return $W_\star = V, W_{\overline{\star}} = \varnothing$
    else
        $B = \mathrm{Attr}^{\mathcal{G}}_{\overline{\star}}(W'_{\overline{\star}})$
        $W''_\Box, W''_\bigcirc = solve(\mathcal{G}{\restriction}_{V \setminus B})$
        return $W_\star = W''_\star, W_{\overline{\star}} = W''_{\overline{\star}} \cup B$
    end if
end if
```

6.16 Remark: Another proof of positiona determinacy
The algorithm differs from the above proof in a key aspect: In the proof, we assumed the set $P_\Box = W_\Box$ to be fixed, but in the algorithm, we need to compute it.

To understand why the algorithm is correct, it is helpful to consider an alternative proof of positional determinacy that proceeds by induction on the number of positions. The drawback is that it proves positional determinacy only for finite game arenas.

We give a sketch of the proof in the following. In the base case, the game arena is empty and so are the winning regions.

Consider a non-empty game arena, and let n be the highest priority assigned to any node. We again assume that n is even (otherwise, the roles of the players have to be swapped). We define N to be the positions with priority n, and $A = \mathrm{Attr}_{\bigcirc}(N)$ as its attractor. Consider the game $\mathcal{G}' = \mathcal{G}_{\restriction V\setminus A}$. Since $V\setminus A$ is a trap, Lemma 6.10, $\mathcal{G}'$ is a deadlock-free parity game. Since $N \neq \varnothing$ and $N \subseteq A$, its number of positions is strictly smaller than the number of positions of $\mathcal{G}$. Hence, we may apply induction to obtain that $V\setminus A = W'_{\square} \uplus W'_{\bigcirc}$ is partitioned into the winning regions of the players and each player has a uniform positional winning strategy from her winning region.

Consider the case that $W'_{\square} = \varnothing$. We claim that in this case, the existential player wins the whole game $\mathcal{G}$ using a positional winning strategy. We define the strategy $s_{\bigcirc}$ to combine the strategy $s'_{\bigcirc}$ for $\mathcal{G}'$ on $W'_{\bigcirc} = V\setminus A$ and the attractor strategy $s_{\mathrm{Attr},\bigcirc}$ on $A = \mathrm{Attr}_{\bigcirc}(N)$. Any play conform to $s_{\bigcirc}$ either visits N infinitely often (in which case $\bigcirc$ wins since n is the highest priority and even), or after some finite prefix, it stays inside $V \setminus A$. Hence, the play has an infinite suffix that is a play of $\mathcal{G}'$ conforming to the winning strategy $s'_{\bigcirc}$. By prefix independence, the existential player wins the whole play.

Consider $W'_{\square} \neq \varnothing$. Define $B = \mathrm{Attr}_{\square}(W'_{\square})$, and note that $B \neq \varnothing$. Hence, $\mathcal{G}'' = \mathcal{G}_{\restriction V\setminus B}$ is a deadlock-free parity game to which we can apply induction, obtaining $V\setminus B = W''_{\square} \uplus W''_{\bigcirc}$ and corresponding positional winning strategies. The winning strategy for the existential player $\bigcirc$ from $W''_{\bigcirc}$ in $\mathcal{G}''$ is also a winning strategy from $W''_{\bigcirc}$ in $\mathcal{G}$ by Lemma 6.13 since $V \setminus B$ is a trap for $\square$. Hence, $W''_{\bigcirc} \subseteq W_{\bigcirc}$ and a positional winning strategies exist.

To see that $W''_{\square} \cup B \subseteq W_{\square}$, we construct a positional winning strategy that combines (1) the strategy $s'_{\square}$ (for $\mathcal{G}'$) on $W'_{\square}$, (2) the strategy $s_{\mathrm{Attr},\square}$ on $\mathrm{Attr}_{\square}(W'_{\square}) \setminus W'_{\square}$, (3) the strategy $s''_{\square}$ (for $\mathcal{G}''$) on $W''_{\square}$. Any play conforming to the combined strategy from $W''_{\square} \cup B$ either completely occurs in $W''_{\square}$ (and conforms to the winning strategy $s''_{\square}$), or it enters $W'_{\square}$ after finitely many steps and then stays there, conforming to the winning strategy $s_{\square}'$. The techniques needed to formally prove this are similar to the ones used in the proof of Theorem 6.7 above.

Computational complexity

To finish this section, we want to study the computational complexity of solving parity games. To this end, we see parity games as decision problems.

Solving parity games (PARITY)
Given: $G = (V_\Box \uplus V_\bigcirc, R)$, Ω, $x \in V$
Question: Is x winning for the existential player?

The algorithm above can be used to solve PARITY by solving the game and checking whether the given vertex x is in the winning region of the existential player.

6.17 Lemma
Algorithm 6.15 solves PARITY in time $|G|^n \cdot \text{poly}(|G|)$, where n is the highest occurring priority.

Currently, PARITY is not proven to be in P, but it is in NP ∩ *co*NP. To show this, we first consider the following lemma.

6.18 Lemma
Let $s_{\star}$ be a positional strategy and let $x \in V$ be a vertex. One can check in polynomial time whether $s_{\star}$ is winning from x.

Proof: Exercise 6.28, Part a). ■

By Theorem 6.7, exactly one player has a positional winning strategy for the given initial position. This means that positional strategies can be used as a polynomial certificate.

6.19 Proposition
PARITY ∈ NP ∩ coNP.

Proof:
To show PARITY ∈ NP, we give an algorithm that uses existential non-determinism. The algorithm guesses a positional winning strategy for the existential player, checks whether it is winning from x, and returns yes if this is the case. The strategy can be stored using polynomial space, and checking whether it is winning can be done in polynomial time by Lemma 6.18.

To show PARITY ∈ coNP, there are two possible approaches:

- Use universal non-determinism to check that no strategy for the existential player is winning from x.
- Solve the complement problem using existential non-determinism by guessing a strategy for the opponent the universal player and checking whether it is winning from x.

■

6.20 Remark
Assume we could show that PARITY is NP-complete. Then its complement problem is coNP-complete, and since parity games are self-dual, PARITY itself is also coNP-complete.

We would obtain NP = coNP, a statement that is assumed to be wrong, since it would mean that the polynomial hierarchy would collapse to the first level.

In 2017, a new algorithm was presented that achieves a much better running time.

6.21 Theorem: Parity games in polynomial time, Calude et al. 2017 [Cal+17]

- PARITY can be solved in **quasi-polynomial time**

  $$\mathcal{O}\left(|G|^{\log n+6}\right) \subseteq \mathcal{O}\left(2^{(\log |G|)^2}\right),$$

  where n is the highest priority.

- PARITY is **fixed-parameter tractable in the highest priority** n as there is an algorithm solving it in time

  $$\mathcal{O}\left(|G|^5\right) + g(n),$$

  where g is some function whose value only depends on n.

An important consequence is that parity games can be solved quickly even for large game arenas if the highest priority is small. Whether PARITY is in P is still open. On the one hand, there are some problems for which quasi-polynomial algorithms could be improved to obtain a polynomial algorithm. On the other hand, there are problems for which quasi-polynomial lower and upper bounds have been proven. (The lower bounds assume the exponential time hypothesis to hold, a strengthened version of NP ≠ P.)

6.22 Remark
An easier proof ot the result by Calude et al. [Cal+17] was later found by Jurdzinski and Lazic [JL17].

Exercises

6.23 Exercise: Encoding winning conditions

Let $G = (V_\square \uplus V_\bigcirc, R)$ be a deadlock-free, finite game arena. Let $x, y \in V$ be two positions, $x \neq y$.

a) Present a reachability/safety game whose winning condition encodes the following property:
A play is won by the existential player if it visits first x, then y.

Note: You are allowed to modify the game arena G.

b) Present a reachability/safety game whose winning condition encodes the following property:
A play is won by the universal player if it does not visit both x and y.

c) Present a Büchi/coBüchi game whose winning condition encodes the following property:
A play is won by the existential player if it visits x at least once, and later visits y infinitely often.

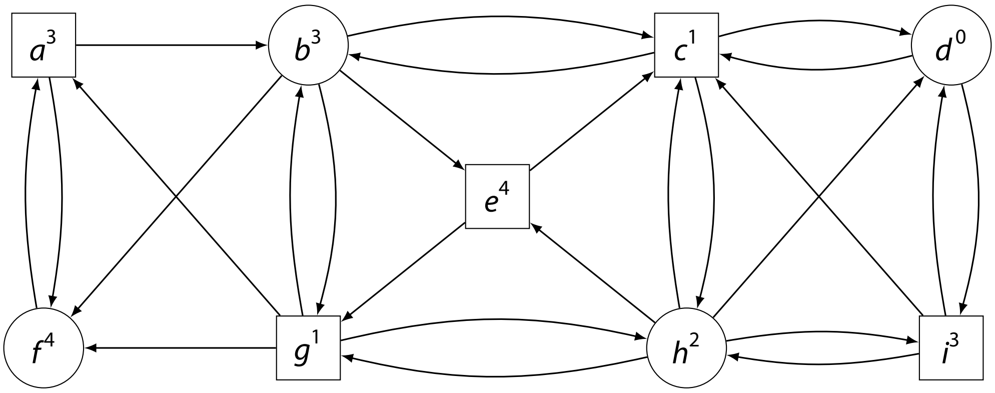
d) Present a parity game whose winning condition encodes the following property:
A play is won by the existential player if it either does not visit x infinitely often, or it visits both x and y infinitely often.

e) Present a parity game whose winning condition encodes the following property:
A play is won by the existential player if it either does not visit x infinitely often, or it visits x, but not y infinitely often.

For each part, reason briefly why your construction is correct.

6.24 Exercise

Consider the parity game given by the following graph. For each vertex labeled with x^i, the letter x denotes the name of the vertex, the superscript denotes its priority $\Omega(x) = i$.



For each player, identify her winning region and present a uniform positional winning strategy. Reason briefly why the strategies are indeed winning.

6.25 Exercise

Prove Part b) of Lemma 6.6.

6.26 Exercise: Is it a trap?

a) Formally prove Part a) of Lemma 6.10.

b) Construct a game arena and a set Y such that $\mathrm{Attr}_{\star}(Y)$ is not a trap for any of the players. Prove that these properties hold.

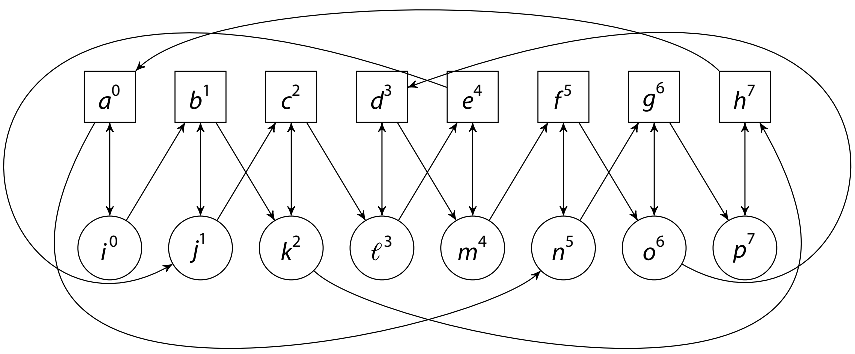
6.27 Exercise: It's a trap!

Formally prove Lemma 6.13.

6.28 Exercise: Algorithmics of parity games

a) Prove Lemma 6.18.

b) Use Zielonka's recursive algorithm to solve the following parity game. The notation is as in Exercise 6.24.



6.29 Exercise: Weak parity games

Let us consider **weak parity games**. Just like a parity game, a weak parity game is given by a game arena $G = (V_\square \uplus V_\bigcirc, R)$ and a priority function Ω. Instead of considering the highest priority that *occurs infinitely often* to determine the winner of a play, we consider the highest priority that *occurs at all*.

Formally, the winner of the weak parity game given by G and Ω is determined by the **weak parity winning condition**:

$$\begin{aligned} win \ : \ Plays_{max} &\to \{\bigcirc, \square\} \\ p &\mapsto \begin{cases} \bigcirc, & \text{if } \max\{\Omega(p_i) \mid i \in \mathbb{N}\} \text{ is even,} \\ \square, & \text{else, i.e. if } \max\{\Omega(p_i) \mid i \in \mathbb{N}\} \text{ is odd.} \end{cases} \end{aligned}$$

a) Present an algorithm that, given a weak parity game on a finite, deadlock-free game arena, computes the winning regions of both players. Briefly argue that your algorithm is correct.

b) Is the winning condition of weak parity games prefix-independent, i.e. does Lemma 6.5 hold?

Do uniform positional winning strategies exist?

Application: Rabin's tree theorem

We have now gathered the prerequisites to study a theoretical application of game theory. Rabin's tree theorem is a deep result from automata theory, stating that a certain class of languages of infinite trees is closed under complementation. Its easiest proof relies on the positional determinacy of Parity games, Theorem 6.7.

The content can be found in Section 15.

7. Muller games

The goal of this section is to generalize the parity winning condition by getting rid of the dependency on the priority assignment. We obtain a type of games called **Muller games**. We will show that these games are determined. However, the winning strategies are not positional, but use finite memory.

Muller games

7.1 Assumption

We assume throughout the section that $G = (V_\square \uplus V_\bigcirc, R)$ is a fixed deadlock-free and finite game arena. We will use $n = |V|$ to denote the number of positions.

Intuitively, the winning condition of a Muller game specifies for each set of position that can occur infinitely often which player wins.

7.2 Definition: Muller game, Muller winning condition

A **Muller game** $\mathcal{G}^{\text{Muller}}$ on the game arena G is given by a **judgment**

$$judgment\colon \mathcal{P}(V) \to \{\bigcirc, \square\}$$

that assigns to each set of positions a winner.

The **Muller winning condition** is given by

$$\begin{array}{rcrcl} win & : & Plays_{max} & \to & \{\bigcirc, \square\} \\ & & p & \mapsto & judgment(\text{Inf}(p)) \end{array}$$

7.3 Remark

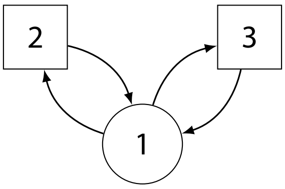
Muller games are named after David E. Muller (1924 - 2008), an American computer scientist. In 1963, he invented Muller automata, automata that accept an infinite word if and only if the set of states that occurs infinitely often is inside a specified collection. The acceptance condition of these automata is very similar to the winning condition of Muller games, hence the name.

Sources

The section does not follow any particular source. The book [HL11] (in German) discusses the translation from Muller automata to parity automata, which is very similar to the construction that we consider here. A discussion of Muller games in English can be found in [KN01]. Note that [KN01] only considers the simple case in which $\mathit{judgment}(V) = \bigcirc$ and $\mathit{judgment}(X) = \Box$ for all $X \subsetneq V$.

See [ZKW] for another presentation of Muller games.

7.4 Example
Consider the following example.



For the above game arena, we use the winning function defined by $\mathit{judgment}(\{1,2,3\}) = \bigcirc$ and $\mathit{judgment}(X) = \Box$ for all other $X \subsetneq V$.

Every position is winning for the existential player. Namely, the strategy that for position 1 alternates between choosing 2 and 3 is winning, as it generates a play in which all positions occur infinitely often.

It is easy to see that the existential player has no positional winning strategy. A positional strategy will either only generate plays p with $\mathrm{Inf}(p) = \{1,2\}$ or only plays with $\mathrm{Inf}(p) = \{2,3\}$. In both cases, the universal player wins. Consequently, there is no priority assignment on the game arena such that the resulting parity game is equivalent to the Muller game.

Latest appearance records

A strategy for a Muller game will need to track information about the past of the game. However, instead of tracking the history of the play (which has unbounded length), it is sufficient to track the order of last appearances of the positions. In the following, we define a data structure that does precisely that.

7.5 Definition: Latest appearance record
A **latest appearance record (LAR)** *lar* for the game arena G is a tuple

$$lar = (x_0 \dots x_{n-1}, i)$$

such that

- $x_0 \dots x_{n-1}$ is a permutation of the set of positions, i.e. a sequence of positions in which each positions occurs exactly once, and
- $i \in \{0, \dots, n-1\}$ is a number.

Let us denote by LAR the set of all LARs, and note that $|LAR| = n! \cdot n$.

LARs support an **update** operation that takes an LAR *lar* and a position $x \in V$ and produces a new LAR lar' defined as follows

$$\begin{array}{rcrcl} \text{update} & : & LAR \times V & \rightarrow & LAR \\ & & ((x_0 \dots x_{n-1}, i), x) & \mapsto & (xx_0 \dots x_{j-1}x_{j+1} \dots x_{n-1}, j) \end{array}$$

where j is the index of x in $x_0 \dots x_{n-1}$ (i.e. $x_j = x$).

This means position x is moved to the front and its old index of the sequence is exhibited as the second component.

7.6 Example
Consider the LAR $lar = (123, 1)$ for the game from Example 7.4. If we now see position 3, we obtain

$$\text{update}(lar, 3) = (312, 2)\,,$$

i.e. 3 is now the most recent positions, and it was moved from index 2.

As the name suggest, the latest appearance records indeed track the latest appearances of the positions in a play. If *lar* is an arbitrary LAR and $p = p_0 \dots p_k \in V^*$ is a finite sequence of positions in which each position occurs at least once, then

$$\text{update}(\dots\text{update}(\text{update}(\text{update}(lar, p_0), p_1)\dots)p_k)$$

will be an LAR of the shape (p', i), where p' is obtained from p by (1) reversing the order (such that the most recent position is leftmost) and (2) removing from each position all occurrences but the last one. The result shows the positions ordered by their latest appearance.

The number i in the second component of an LAR shows the old index of the position that was moved to the front by the update. We will comment on why this number is needed in a second.

Note that we can split the first component $x_0x_1 \dots x_{n-1}$ of an LAR into the **most recent position** x_0 and the **history** $x_1 \dots x_{n-1}$. For convenience, we define a function returning the most recent position,

$$\begin{array}{rrcl} \mathrm{mr} : & LAR & \to & V \\ & (x_0 \dots x_{n-1}, i) & \mapsto & x_0 \, . \end{array}$$

In the following, we will translate the given Muller game into a parity game with LAR as the set of positions and moves induced by the update function.

7.7 Definition
We define the **LAR parity game** $\mathcal{G}^{\text{parity}}$ to be the parity game on the game arena

$$G' = (LAR, R')$$

with

$$\begin{aligned} & owner'(lar) = owner(\mathrm{mr}(lar)) \\ \text{and} \quad & R' = \big\{(lar, lar') \,\big|\, \mathrm{mr}(lar) = x, (x, y) \in R, lar' = \mathrm{update}(lar, y)\big\} \, . \end{aligned}$$

Its priority assignment is given by the function

$$\begin{array}{rrcl} \Omega : & LAR & \to & \{0, \dots, 2n-1\} \\ & (x_0 \dots x_{n-1}, i) & \mapsto & \begin{cases} 2i \, , & \text{if } judgment(\{x_0, \dots, x_i\}) = \bigcirc \, , \\ 2i+1 \, , & \text{if } judgment(\{x_0, \dots, x_i\}) = \square \, . \end{cases} \end{array}$$

Note that if one projects the new graph G' to the most recent positions, one obtains the original game arena G. Therefore, one may see G' as a version of G that keeps track of (a part of) the history of the play.

The intuition behind the priority assignment is more complicated. We argue why it depends on $\{x_0, \dots, x_i\}$, where i is the second component of the LAR, i.e. the old index of the position that was just moved to the front.

Consider a play of the Muller game containing a simple cycle from x to x, say $p = p'.xx^{(1)} \dots x^{(m)}x$. Then during the cycle from x to x, we have seen the set of positions $\{x, x^{(1)}, \dots, x^{(m)}\}$. This cycle is good for the existential player if and only if *judgment*$(\{x, x^{(1)}, \ldots, x^{(m)}\}) = ○$. Let *lar* be the LAR associated to the first occurrence of x, and let *lar*′ be the LAR associated to the second occurrence. Note that we have

$$lar' = \text{update}\Big(\text{update}\Big(\ldots \text{update}\Big(\text{update}\Big(\text{update}\Big(lar, x^{(1)}\Big), x^{(2)}\Big) \ldots\Big)x^{(m)}\Big), x\Big)$$

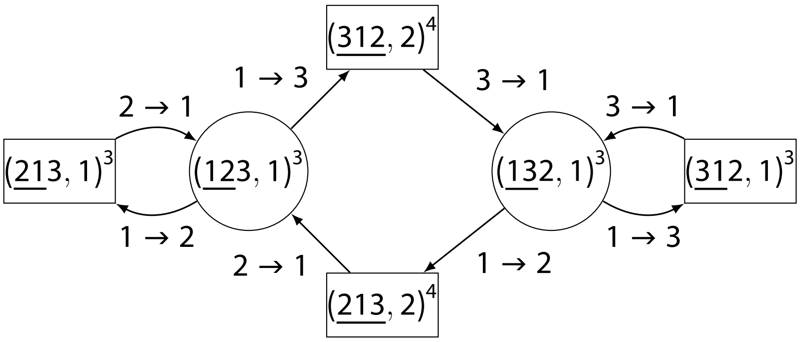
and *lar*′ is of the shape

$$lar' = (xx_1 \ldots x_{n-1}, i)$$

such that indeed $\{x, x_1, \ldots, x_i\} = \{x, x^{(1)}, \ldots, x^{(m)}\}$. Note that the priority of *lar*′ is even (which is good for the existential player with respect to the parity winning condition) if and only if this set is good for ○ with respect to the Muller winning condition.

Now consider an infinite play p of the Muller game and let $X = \text{Inf}(p) \subseteq V$. This means that we can decompose $p = p^{fin}.p^{inf}$ such that that in p^{inf}, only positions in X occur: All positions not in X occur only finitely often, so for each position, there is a finite index at which it occurs for the last time. Take the maximum of these indices over all positions not in X to determine the location of the cut. Consider the sequence of LARs in the parity game associated to p^{inf}. In it, only the first $|X|$ entries of the (first component of the) LAR will be modified anymore, since the other positions do not occur and are never moved to the front. However, when we consider the set $\{x_0, \ldots, x_i\}$ on which the priority assignment depends, we will not always have $\{x_0, \ldots, x_i\} = X$: It might happen that some position occurs twice without all other positions occurring in between. In this case, we have $\{x_0, \ldots, x_i\} \subsetneq X$ and $i < |X| - 1$. Nevertheless, as we are interested in the largest priority infinitely often, we are sure that the priority that is exhibited by the positions with $\{x_0, \ldots, x_i\} = X$ will be the dominating priority in the run.

7.8 Example
We consider the construction of $\mathcal{G}^{\text{parity}}$ for the Muller game $\mathcal{G}^{\text{Muller}}$ from Example 7.4.



Here, we have only drawn the part of the graph G' reachable from the LAR (123, 1). The names of the positions are of the shape $lar^{\Omega(lar)}$, i.e. the superscript denote the priority. The part $\underline{x_0 \ldots x_i}$ of an LAR $lar = (x_0 \ldots x_{n-1}, i)$ that is underlined is the part on which the priority assignment depends. The moves are labeled by the moves of the original game that induce them.

Note that

- no LAR has 0 as second component because the original game contains no self loops,
- there is no LAR in which the first component contains 2 and 3 in succession because there are no corresponding moves in the original game.

From parity to Muller with finite-memory strategies

To show the determinacy of Muller games, we will prove the following correspondence between the original Muller game $\mathcal{G}^{\text{Muller}}$ and the parity game $\mathcal{G}^{\text{parity}}$.

7.9 Theorem: Correspondence
A position $x \in V$ is winning in the Muller game $\mathcal{G}^{\text{Muller}}$ for some player ☆ if and only if any/all positions *lar* with $\text{mr}(lar) = x$ are winning for ☆ in the parity game $\mathcal{G}^{\text{parity}}$.

To prove Theorem 7.9, we show how to transform a positional winning strategy for $\mathcal{G}^{\text{parity}}$ into a winning strategy for $\mathcal{G}^{\text{Muller}}$. Since parity games are determined, Theorem 6.7, we then obtain the determinacy of Muller games.

Assume that $s_{☆}^{\text{parity}}$ is a positional strategy that is uniformly winning from all LARs in the winning region of ☆. For simplicity, we will fix one LAR lar_1 from the winning region that we will consider as the initial position.

Unfortunately, we cannot translate $s_{☆}^{\text{parity}}$ into a positional winning strategy for the Muller game. Instead of translating it into an arbitrary strategy (i.e. one that has unrestricted behavior on plays), we translate it into a *simple* strategy. In the following, we define such simple, although non-positional, strategies in general.

7.10 Definition: Finite-memory strategy
A (deterministic) **finite-memory strategy** (or forgetful strategy) for player ☆ is defined by a (deterministic) **finite-state transducer** T that reads the moves of the game and outputs the moves of ☆. More formally, the transducer is a tuple $T = (Q, V, V, q_0, \delta, \text{o})$ where

- Q is a finite set of **internal control states**, the **finite memory**,
- $q_0 \in Q$ is the **initial state**,

- the set of positions V is the **input** as well as the **output alphabet**,
- $\delta\colon Q \times V \to Q$ is the deterministic **transition function** that, given the old state q and the new position x of the game, determines the new state $\delta(q, x)$, and
- $\mathrm{o}\colon Q \to V$ is the **output function** that determines a the successor $\mathrm{o}(q)$ that is put out depending on the current internal state q.

(The transducer should guarantee that whenever it outputs a position, this is actually a valid successor, but we leave this assumption implicit.)

For such a transducer, we define its state state(p) after reading some finite sequence of positions $p \in V^*$ inductively by

$$\begin{aligned}\mathrm{state}(\varepsilon) &= q_0\ ,\\ \mathrm{state}(p'.x) &= \delta(\mathrm{state}(p'), x)\ .\end{aligned}$$

The strategy induced by the transducer can then by defined by

$$\begin{array}{rcrcl} s_{\star}^{T} & : & \mathit{Plays}_{\star} & \to & V \\ & & p & \mapsto & \mathrm{o}(\mathrm{state}(p))\ . \end{array}$$

7.11 Remark
Note that the transducer has only one initial state q_0, but in the very first step, it can update its state depending on the initial position of the play: The base case of the definition of state is the empty sequence ε. Hence, transducers can be used to define uniform strategies.

Instead of considering strategy with irregular behavior (i.e. strategies that can output different successors for plays that are very similar), a finite-memory strategy will base its decision on the state in which the transducer is after reading the play. This allows us to restrict ourselves to storing the state of the transducer (which can be done with space $\log |Q|$) instead of storing the unbounded history of the play.

Recall that $\mathcal{G}^{\text{parity}}$ was constructed by amending the Muller game with finite information about the history of the play, namely by the latest appearance records. It is therefore natural that we use exactly this information to define a finite-state transducer, which will then allow us to simulate the strategy for $\mathcal{G}^{\text{parity}}$ in the Muller game.

7.12 Definition
We define $s_{☆}^{\text{Muller}}$ to be the finite-memory strategy induced by the transducer

$$T = (LAR, V, V, lar_0, \text{update}, \text{o})$$

where LAR and update are defined as before, the output function is defined by

$$\text{o}(lar) = \text{mr}\Big(s_{☆}^{\text{parity}}(lar)\Big)\,,$$

and the initial state lar_0 is a LAR such that $\text{update}(lar_0, \text{mr}(lar_1)) = lar_1$.

Note that indeed update: $LAR \times V \to LAR$ has the required signature. The reason for picking lar_0 as the initial state is that after we update it with respect to the most recent position from lar_1, we obtain precisely the LAR lar_1 for which we assume that $s_{☆}^{\text{parity}}$ is winning.

7.13 Proposition
Assume that $s_{☆}^{\text{parity}}$ is winning from $lar_1 \in LAR$, then the finite-memory strategy $s_{☆}^{\text{Muller}}$ is winning from $\text{mr}(lar_1)$.

Proof:
Let us denote $x = \text{mr}(lar_1)$. Consider a play $p = p_0p_1p_2$ of the Muller game from x (i.e. $p_0 = x$) that is conform to $s_{☆}^{\text{Muller}}$. We associate to it the sequence of states that T has while reading p,

$$lar_0 \xrightarrow{p_0} lar_1 = \text{state}(p_0) \xrightarrow{p_1} lar_2 = \text{state}(p_0p_1) \xrightarrow{p_2} \ldots$$

i.e. for each number i, let lar_i be the state in which T is after reading $p_0 \ldots p_{i-1}$. (Recall that lar_0 is chosen such that we are indeed in state lar_1 after the first move)

Note that the states are latest appearance records and the transition relation of T coincides with the update operation on LAR. Hence, the sequence of LARs $p^{lar} = lar_1 lar_2 \ldots$ (without the initial state) is a valid play of $\mathcal{G}^{\text{parity}}$ from lar_1. Because the output function of T that is used to determined the moves by $s_{☆}^{\text{Muller}}$, is defined using $s_{☆}^{\text{parity}}$, we have that p^{lar} is conform to $s_{☆}^{\text{parity}}$, and hence winning.

Consequently, the highest priority ℓ that occurs infinitely often in p^{lar} is good for player ☆. We may decompose $p^{lar} = p^{larfin}.p^{larinf}$ such that all priorities that occur in p^{larinf} are smaller or equal to ℓ. By the prefix independence of the parity winning condition, Lemma 6.5, we know that also p^{larinf} is won by ☆. Since the Muller winning condition is also prefix independent, it will be sufficient to argue that the corresponding suffix p^{inf} is winning with respect to the Muller game.

Note that since there is some index i that corresponds to priority ℓ (which is, depending on which player ☆ is, either $\frac{\ell}{2}$ or $\frac{\ell-1}{2}$) such that all LARs $(x_0 \ldots x_{n-1}, j)$ have $j \leqslant i$, and infinitely many LARs with $j = i$ occur. This in particular means that all but the first i entries are not moved inside p^{larinf}. Define $X = \{x_0, \ldots, x_i\}$ as the entries that are swapped for some LAR from p^{larinf}. (By the previous argumentation, it does not matter which one we pick.)

We obtain that $\mathrm{Inf}(p) = \mathrm{Inf}\left(p^{inf}\right) \subseteq X$, as the positions not in X are never swapped to the front in p^{larinf}, which means that they do not occur in p^{inf}. To see that $\mathrm{Inf}(p) = X$, note that every position from X has to occur infinitely often, as it is infinitely often swapped to the front in p^{larinf}: Swapping another position to the front will make it wander towards the end of the sequence, until it appears at index i. Since i occurs infinitely often as the second component, it is then swapped to the front after finitely many steps.

To conclude the proof, note that since ℓ was a good priority for ☆, $X = \{x_0, \ldots, x_i\}$ is a set of positions that is good for player ☆ with respect to the Muller judgment. ■

7.14 Example

Consider the positional strategy $s_{\bigcirc}^{\text{parity}}$ for ○ in the parity game $\mathcal{G}^{\text{parity}}$ from Example 7.8 that is defined by $(123, 1) \mapsto (312, 2)$ and $(132, 1) \mapsto (213, 2)$. The plays of $\mathcal{G}^{\text{parity}}$ that are conform to it use the cycle in the middle of the game arena infinitely often (and thus are won by ○ since the highest occurring priority is 4). This strategy induces the finite state strategy $s_{\bigcirc}^{\text{Muller}}$ for $\mathcal{G}^{\text{Muller}}$ that, whenever the game is in position 1, alternates between outputting successor 2 and outputting successor 3 (because the internal state of the transducer alternates between $(123, 1)$ and $(132, 1)$). As discussed in Example 7.4, this strategy for the Muller game is indeed winning.

7.15 Remark

For the proof of Proposition Proposition 7.13, we have constructed a non-uniform winning strategy. In Exercise 7.20, you will see that finite-memory strategies can always be made uniform. In the special case of Muller games, it is actually possible to prove that, assuming $s_{☆}^{\text{parity}}$ is a uniform winning strategy, the LAR strategy $s_{☆}^{\text{Muller}}$ is also uniformly winning.

Using Proposition 7.13, it is easy to show Theorem 7.9.

Proof of Theorem 7.9:
If a LAR *lar* is winning in $\mathcal{G}^{\text{parity}}$ for $\star$, then $\text{mr}(lar)$ is winning in $\mathcal{G}^{\text{Muller}}$ for $\star$ by Proposition 7.13.

If a position x is winning in the Muller game for $\star$, then all LARs *lar* with $\text{mr}(lar) = x$ need to be winning for $\star$ in the parity game. If one such LAR is not winning for $\star$, it needs to be winning for $\overline{\star}$ because parity games are determined. Consequently, also x would be winning for $\overline{\star}$ by Proposition 7.13, a contradiction. ■

From Theorem 7.9, we obtain easily the determinacy of Muller games.

7.16 Theorem: Determinacy of Muller games I
Muller games are determined, $V = W_{\square} \uplus W_{\bigcirc}$.

We have even shown a stronger result: If a position is winning for some player $\star$, then this player has a finite-memory strategy with memory bounded by $n! \cdot n$ (because this is the number of LARs). As mentioned in Remark 7.15, one can in fact show that both players have a uniform finite-memory winning strategy with memory bounded by $n! \cdot n$.

To improve the result, one can observe that the second component i of an LAR $(x_0 \dots x_{n-1}, i)$ is only needed to determine the priority assignment. One could show that there is a uniform positional strategy $s_{\star}^{\text{parity}}$ for $\mathcal{G}^{\text{parity}}$ that does not depend on this second component, i.e. it has

$$s_{\star}^{\text{parity}}(x_0 \dots x_{n-1}, i) = s_{\star}^{\text{parity}}(x_0 \dots x_{n-1}, j)$$

for all i, j. Using this, one can build a transducer that only uses the first components of the LARs as memory.

Altogether, we can strengthen the statement of Theorem 7.16 to obtain the following result.

7.17 Theorem: Determinacy of Muller games II
Muller games are determined, $V = W_{\square} \uplus W_{\bigcirc}$, and each player has a uniform finite-memory strategy for her winning region with memory bounded by $n!$, where $n = |V|$.

One may criticize that the memory requirement of $n!$ bits is very large. However, note that already the encoding of a Muller game is quite large: To encode the judgment, we need to store for each subset of nodes for which player it is winning. If the graph has

$|V| = n$ nodes and we just need one bit per subset, we will need $2^n = |\mathcal{P}(V)|$ many bits to encode the judgment. Now observe that

$$n! \leqslant n^n = (2^{\log n})^n = 2^{\log n \cdot n}$$

which is not polynomial in 2^n, but only *slightly* super-polynomial.

Furthermore, one can show that the memory consumption of $n!$ is essentially optimal.

7.18 Theorem: Optimality of LARs, Theorem 15 in [DJW97]
For each $n \in \mathbb{N}$, there is a Muller game on a game arena with $2n$ many positions such that any winning finite state strategy needs to have memory at least $n!$.

The game that is used to prove Theorem 7.18 is presented in Example 7.22. For the proof, we refer the reader to [DJW97].

7.19 Remark
Deciding which player wins a Muller game from a fixed position is a PSPACE-complete problem: There is a (deterministic) algorithm solving the problem that uses polynomial space, but exponential time. Unless PSPACE = P, there is no algorithm solving the problem just using polynomial time.

Exercises

7.20 Exercise: Making finite-memory strategies uniform
Let $\mathcal{G}$ be a game on some finite graph $G = (V_\square \uplus V_\bigcirc, R)$ with some arbitrary fixed winning condition *win*. Assume for each of the two positions $x, y \in V$, player $\bigcirc$ has some finite-memory strategy, say induced by the transducers T_x and T_y, respectively.

Show how to construct a transducer T such that the finite-memory strategy induced by T is winning from both x and y.

7.21 Exercise: Constructing a transducer

Consider the game $\mathcal{G}(n)$ (for some $n \in \mathbb{N}, n > 0$) on the following graph:

$$\begin{aligned}
V &= \{guess, go\} \uplus N \uplus X \quad \text{with} \\
N &= \{1, \ldots, n\}\,, \\
X &= \{x_1 \ldots x_n\}\,, \\
R &= \quad \{(guess, i), (i, go) \mid i \in \{1, \ldots, n\}\} \\
&\quad \cup \{(go, x_i) \mid i \in \{1, \ldots, n\}\} \cup \big\{(x_i, x_j) \mid i, j \in \{1, \ldots, n\}\big\}\,, \\
owner(guess) &= owner(i) = \square \text{ for all } i \in N\,, \\
owner(go) &= owner(x_i) = \bigcirc \text{ for all } x_i \in X\,.
\end{aligned}$$

Let us focus on plays starting in position *guess*. Note that all maximal plays from this position are infinite, and they visit exactly one position from the set N, and they visit this position exactly once.

Such a play p is won by $\bigcirc$ if and only $|\mathrm{Inf}(p) \cap X| = m$ holds, where m is the unique position from N that occurs in p.

a) Draw G for $n = 4$. Assume that the universal player picks the move $(guess, 3)$. Draw in a positional strategy for $\bigcirc$ that wins under this assumption.

b) Let $n \in \mathbb{N}, n > 0$ be an arbitrary fixed number, and consider $\mathcal{G}(n)$. Show how to construct a transducer T such that the finite-memory strategy for $\bigcirc$ induced by T is winning from *guess*.

7.22 Example: An expensive game

Let $n \in \mathbb{N}, n > 0$ be a fixed positive number. We define a Muller game $\mathcal{G}^{\text{Muller}}$ on the game arena $G = (V, R)$ with $V = \{1, \ldots, n\} \times \{\bigcirc, \square\}$ (where the second component indicates the active player) and the moves defined by

$$R = \Big\{(i, \star) \to (j, \overline{\star}) \;\Big|\; i, j \in \{1, \ldots, n\}, \{\bigcirc, \square\} = \big\{\star, \overline{\star}\big\}\Big\}\,.$$

The Muller judgment is defined as follows: $judgment(X) = \bigcirc$ if and only if

$$|X \cap V_{\bigcirc}| = \max\{i \mid (i, \square) \in X\}\,.$$

a) Draw G for $n = 2$.

b) Explain the winning condition in your own words.

c) Construct the parity game $\mathcal{G}^{\text{parity}}$ obtained form $\mathcal{G}^{\text{Muller}}$ by the LAR construction. You can fix some initial LAR lar_0 and just draw all LARs reachable from lar_0. Similar to the example in the lecture, mark all positions with their priorities.

Draw in a positional winning strategy for the existential player ○.

7.23 Exercise: From parity to Muller

a) Let $G = (V_{\Box} \uplus V_{\bigcirc})$ be a finite, deadlock-free graph. Consider the parity game $\mathcal{G}^{\text{parity}}$ defined on G by some priority assignment $\Omega \colon V \to \{0, \ldots, n\}$.

Present a Muller judgment $\mathit{judgment}\colon \mathcal{P}(V) \to \{\bigcirc, \Box\}$ such that the corresponding Muller game $\mathcal{G}^{\text{Muller}}$ is equivalent to $\mathcal{G}^{\text{parity}}$: Any position $x \in V$ is winning for some player ☆ in $\mathcal{G}^{\text{Muller}}$ if and only if it is winning for this player in $\mathcal{G}^{\text{parity}}$.

b) We call a Muller game $\mathcal{G}^{\text{Muller}}$ **union-closed** if its defining judgment has the following property: If $\mathit{judgment}(X) = ☆$ and $\mathit{judgment}(Y) = ☆$ for some sets $X, Y \subseteq V$, then $\mathit{judgment}(X \cup Y) = ☆$.

Check that the game you have constructed in Part a) is union-closed.

Note: One can show that if a Muller game is union-closed, and $x \in V$ is winning for some player ☆, then ☆ has a positional winning strategy from x.

8. Mean payoff games

In this section, we want to study a different kind of games with perfect information: The goal of the players it not to satisfy a winning condition, but to optimize their **payoff**, a numeric value associated to each play.

Our goal is to study mean payoff games. In these games, both players alternately pick moves in a finite graph. Each position has an associated **weight**, and the payoff of an infinite play is determined by the mean (average) of the weights.

Similar to the games that we previously considered, we want to show a theorem stating that positional winning strategies for such games exist.

In the proof of this theorem, we will associate to a mean payoff game a game whose plays are of bounded length. Therefore, we will start by studying such games.

Sources
The content of the first subsection is common knowledge in game theory and can be found in most textbooks on the topic.

The content of the rest of this section is based on the papers [EM79] and [ZP96].

Zero-sum games

8.1 Definition

A **zero-sum** game of length $k \in \mathbb{N}$ is a game $\mathcal{G}$ given by a game arena G, a fixed initial position x_0 and a **payoff** function φ.

It is played as follows: Both players play for in total at most k moves or until the play deadlocks. Let $Plays^{\leq k}_{max}$ denote the set of such plays.

The **payoff** function φ maps

$$\varphi\colon Plays^{\leq k}_{max} \to \mathbb{R}\ ,$$

such plays to a real number, yielding the **payoff** $\varphi(p)$ of the play.

We think of it as if after play p, the existential player has to pay the universal player the value $\varphi(p)$ (respectively the universal player pays the value $|\varphi(p)|$ to the existential player if $\varphi(p)$ is negative). The goal of the existential player is to minimize her loss $\varphi(p)$, the goal of the universal player is to maximize her income $\varphi(p)$.

8.2 Remark
These games are called zero-sum games because the income of the universal player equals the loss of the existential player. Wealth is neither created nor destroyed.

The goal of each player is not to satisfy a winning condition, but to optimize her payoff. To formalize this, we define strategies that guarantee a certain payoff.

8.3 Definition
A strategy $s_{☆}$ for either player **guarantees value** v if any play from the initial position x_0 that conforms to $s_{☆}$ has

- $\varphi(p) \leqslant v$ if $☆ = \bigcirc$, resp.
- $\varphi(p) \geqslant v$ if $☆ = \square$.

We are interested in the smallest value $v_{\bigcirc}$ that can be guaranteed by a strategy for the existential player, and in the largest value $v_{\square}$ that can be guaranteed by a strategy for the universal player. We are in particular interested in whether these values coincide.

For (not necessarily positional) strategies $s_{\square}, s_{\bigcirc}$ for each of the players, let $p@s_{\square}s_{\bigcirc}$ denote the unique play from the initial position x_0 that is conform to the strategies. Note that each play occurs as $p@s_{\square}s_{\bigcirc}$ for suitable strategies.

The best value that a strategy for the existential player can guarantee is

$$v_{\bigcirc} = \min_{s_{\bigcirc}} \max_{s_{\square}} \varphi(p@s_{\square}s_{\bigcirc}) \,.$$

Similarly, the best value that a strategy for the universal player can guarantee is

$$v_{\square} = \max_{s_{\square}} \min_{s_{\bigcirc}} \varphi(p@s_{\square}s_{\bigcirc}) \,.$$

The next lemma states that the minimal loss of the existential player is in general larger or equal to the maximal income of the universal player.

8.4 Lemma

$$v_{\bigcirc} = \min_{s_{\bigcirc}} \max_{s_{\square}} \varphi(p@s_{\square}s_{\bigcirc}) \geqslant \max_{s_{\square}} \min_{s_{\bigcirc}} \varphi(p@s_{\square}s_{\bigcirc}) = v_{\square} \,.$$

Proof:
Let $s'_{\bigcirc}$ be the strategy that minimizes $\max_{s_\square} \varphi(p@s_\square s_\bigcirc)$. Similarly, let $s'_\square$ be the strategy maximizing $\min_{s_\bigcirc} \varphi(p@s_\square s_\bigcirc)$.

We have

$$\begin{aligned}
\min_{s_\bigcirc}\max_{s_\square} \varphi(p@s_\square s_\bigcirc) = \quad & \max_{s_\square} \varphi\big(p@s_\square s'_\bigcirc\big) \\
\geqslant \quad & \varphi\big(p@s'_\square s'_\bigcirc\big) \\
\geqslant \quad & \min_{s_\bigcirc} \varphi\big(p@s'_\square s_\bigcirc\big) \\
= \max_{s_\square}\min_{s_\bigcirc} & \varphi(p@s_\square s_\bigcirc)\ .
\end{aligned}$$

■

This lemma even holds in much more general settings than the one considered here, e.g. when we drop the condition that each maximal play has bounded length. Note that we can write $\min_{s_\bigcirc}$ and $\max_{s_\square}$ because there are only finitely many strategies. In a more general setting, infinitely many strategies may exist, so the minimum and maximum might not be well-defined. In this case, we have to replace the minimum by the infimum $\inf_{s_\bigcirc}$ over all strategies, and maximum by the supremum $\sup_{s_\square}$.

8.5 Definition
A length-k zero-sum game **has value** v if there are strategies for each of the players that guarantee value v, i.e.

$$v = v_\bigcirc = v_\square\ .$$

In particular, a game either has no value, or it has a unique value.

There are games that do not have a value, i.e. games for which $v_\bigcirc > v_\square$ holds.

8.6 Remark
As already briefly mentioned, the concepts in this section correspond to concepts for the types of games that we already studied: The payoff corresponds to the winning condition, and strategies that guarantee a value correspond to winning strategies.

Likewise, having a value corresponds to the game being determined.

The simple games under consideration here always have a value.

8.7 Theorem: Minmax theorem for zero-sum games of bounded length
Each length-k zero-sum game has a value.

Proof:
We proceed by induction on the maximal length k.

For $k = 0$, the value is $\varphi(\varepsilon)$ and there is nothing to show.

Assume the statement holds for games of length at most $k - 1$. Let $\mathcal{G}$ be the game of length k under consideration. A play of length k can be seen as the first position x_0, followed by a play of length $k - 1$ from y.

$$x_0 \longrightarrow y \rightsquigarrow$$

For each possible successor y, we consider a new zero-sum game $\mathcal{G}^y$:

- Its plays have length at most $k - 1$.
- Its initial position is y.
- Its payoff function φ' is defined by

  $$\varphi'(p) = \varphi(x_0.p) ,$$

  i.e. we prepend the position x_0 that we assume has already been visited in $\mathcal{G}$.

By induction, each such game has a value v^y.

We assume wlog. that the player making the first move is the existential player. If this is not true, one has to swap the roles of the players in the following and to maximize instead of minimizing.

We claim that the value of the original game is $\min_y v^y$, where we minimize over all y such that there is an arc $(x_0, y) \in R$.

Let y' be a node y such that v^y is minimal. It remains to prove that both players can guarantee $v^{y'}$ in $\mathcal{G}$.

The existential player can pick the move (x, y') and then use her strategy for $\mathcal{G}^{y'}$ guaranteeing value $v^{y'}$. More formally, let $s_{\bigcirc}^{y'}$ be a strategy for the existential player for the game $\mathcal{G}^{y'}$ that guarantees $v^{y'}$. We define a strategy $s_{\bigcirc}$ for $\mathcal{G}$ as follows:

$$s_{\bigcirc}(x_0) = y' ,$$
$$s_{\bigcirc}(x_0.p) = s_{\bigcirc}^{y'}(p) .$$

By the definition of the payoff function φ' on $\mathcal{G}^{y'}$, any play of the original game conforming to $s_{\bigcirc}$ has value at most $v^{y'}$

The universal player has no influence on the first move (x_0, y) that is made by the existential player. For each such y, let $s^y_\square$ be her strategy guaranteeing v^y in $\mathcal{G}^y$. We combine these strategies to obtain $s_\square$ as follows:

$$s_\square(x_0.y.p) = s^y_\square(y.p) .$$

Assuming we fix the first move (x_0, y) made by the existential player, then $s_\square$ guarantees value v^y by the definition of the payoff function φ'. Therefore, for an arbitrary first move, $s_\square$ guarantees $\min_y v^y$. ■

8.8 Remark

This theorem has also been established for more general payoff games, but as mentioned above, it does not hold in all settings.

Mean payoff games

In this subsection, we want to consider payoff games whose maximal plays are infinite.

In the previous subsection, we have allowed an arbitrary payoff function that can assign each play an arbitrary value. In a sense, this allows the function to exhibit irregular behavior: Very similar plays can have vastly different payoffs.

Here, we restrict ourself to a very regular setting: We assume that each position x of the graph has an associated **weight** $w(x) \in \mathbb{R}$. The payoff of a play is the mean (average) over the weights of the positions visited in the play. Since the play is infinite, we have to express this mean as a limit.

For simplicity, we impose some more restrictions on the finite game arena $G = (V_\square \uplus V_\bigcirc, R)$:

- It should be deadlock-free.
- We assume the initial position $x_0 \in V_\bigcirc$ is owned by the existential player.
- We assume that $V = V_\square \uplus V_\bigcirc$ is a **bipartite** decomposition of the graph:

$$R \subseteq (V_\square \times V_\bigcirc) \uplus (V_\bigcirc \times V_\square) .$$

The last condition enforces that the players alternately take turns.

We furthermore assume that a **weight function**

$$w\colon V \to \mathbb{R}$$

is given, assigning each position x its **weight** $w(x)$.

8.9 Definition

A **mean payoff game** $\mathcal{G}^{inf}$ is given by a game arena G and initial position x_0 as above together with a weight function w.

To an infinite play $p = p_0 p_1 p_2 \ldots$ of $\mathcal{G}^{inf}$, written as a sequence of moves, we associate two values

$$\varphi_{\bigcirc}(p) = \limsup_{n\to\infty} \frac{1}{n+1} \sum_{i=0}^{n} w(p_i)\,,$$
$$\varphi_{\Box}(p) = \liminf_{n\to\infty} \frac{1}{n+1} \sum_{i=0}^{n} w(p_i)\,.$$

We think of $\varphi_{\bigcirc}(p)$ as the loss of the existential player, and of $\varphi_{\Box}(p)$ as the income of the universal player. The goal of the existential player is to minimize $\varphi_{\bigcirc}$, the goal of the universal player is to maximize $\varphi_{\Box}$.

8.10 Remark

For each $n \in \mathbb{N}$, the expression

$$\frac{1}{n+1} \sum_{i=0}^{n} w(p_i)$$

is the mean (arithmetic average) over the weights of the first $n+1$ positions of the play. We can consider the sequence formed by these values for all $n \in \mathbb{N}$. We would like to define the mean over the infinite play as the limit of this sequence, i.e.

$$\varphi(p) = \lim_{n\to\infty} \frac{1}{n+1} \sum_{i=0}^{n} w(p_i)\,.$$

Unfortunately, it is not clear whether this limit exists.

To solve this problem, we consider the limit superior respectively the limit inferior. Recall that they are defined to be the supremum resp. infimum of the set of limit points of a sequence.

In contrast to the limit, they are well-defined for any sequence. The limit exists if and only if their values coincide.

In principle, these values could be (minus) infinity. This will never occur in the setting considered here, because we only have finitely many arcs and thus the range of the weight function is bounded.

Note that by definition, this type of game is not necessarily a zero-sum game: $\varphi_{\bigcirc}(p)$ could be strictly larger than $\varphi_{\square}(p)$.

The key theorem that we want to prove expresses that, firstly, mean payoff games always have a value, and secondly, this value can be guaranteed for both players by positional strategies.

8.11 Theorem: Ehrenfeucht & Mycielski 1979 [EM79]

There is a value v such that both players have positional strategies $s_{\square}, s_{\bigcirc}$ such that:

- Any play p from x_0 conforming to $s_{\bigcirc}$ has $\varphi_{\bigcirc}(p) \leqslant v$.
- Any play p from x_0 conforming to $s_{\square}$ has $\varphi_{\square}(p) \geqslant v$.

Using the notions from the previous subsection, one could phrase this as: Mean payoff games have a value, and it can be achieved using positional strategies.

8.12 Remark

- In the literature, one usually considers a weight function in $R \to \mathbb{R}$ that assigns each arc a weight. To fit better the notation used in the rest of this lecture, we have adapted the theory to the case of weighted vertices.
- Without the assumption that the graph is bipartite, i.e. the players are taking turns alternately, the theory becomes substantially more difficult. Nevertheless, positional determinacy can be proven [V A88].

To establish the result, we consider a version of the game in which all maximal plays are of bounded length. The idea is to stop after the first repetition of a position.

8.13 Definition

The game $\mathcal{G}^{fin}$ is defined to be played on the same game arena G and from the same initial position $x_0 \in V_{\bigcirc}$ as $\mathcal{G}^{inf}$.

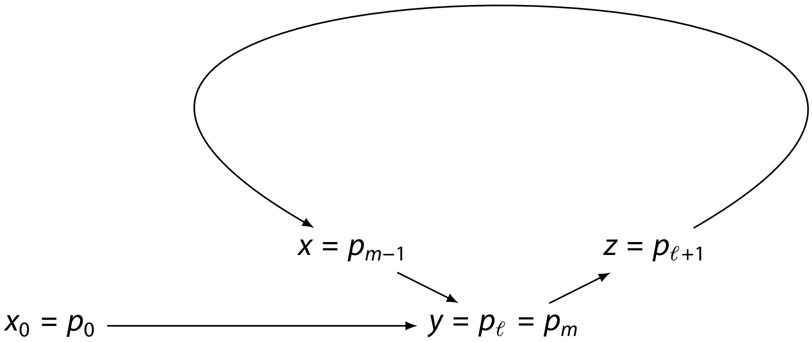
Both players pick moves as usual.

A play p deadlocks as soon as a player $\star \in \{\bigcirc, \square\}$ picks a position $p_m = y \in V_{\overline{\star}}$ such that y already occurred in the game, i.e. there is $\ell < m$ with $p_\ell = y$.

For such a play p of this game, we assume that the existential player pays to the universal player the value

$$\varphi^{fin}(p) = \frac{1}{m-\ell} \sum_{i=\ell+1}^{m} w(p_i) \, .$$

We depict a maximal play of $\mathcal{G}^{fin}$ in the following figure.



The payoff of a play of $\mathcal{G}^{fin}$ is the average of the weights of the positions occurring in the loop from y to y (with y only counted once). The weights in the prefix are not regarded.

8.14 Lemma
$\mathcal{G}^{fin}$ has a value.

Proof:
$\mathcal{G}^{fin}$ can be modeled as zero-sum payoff game of bounded length, see Exercise 8.30.

■

The idea in the following is to relate $\mathcal{G}^{fin}$ to $\mathcal{G}^{inf}$. Firstly, we will prove that strategies for $\mathcal{G}^{fin}$ can be lifted to obtain strategies for $\mathcal{G}^{inf}$ guaranteeing the same value. This will prove that $\mathcal{G}^{inf}$ has a value. Secondly, we need to show that there are positional strategies for $\mathcal{G}^{inf}$. By the first part of the development, it will be enough to prove that $\mathcal{G}^{fin}$ admits positional strategies. To show this, we go from the finite game (more precisely, the game with finite maximal plays) to the infinite game, where we can use that the payoff function is defined as a limit.

Ehrenfeucht and Mycielski state this in their paper as follows: "An amusing feature of our proofs is that we have to use both games to establish our claims about any one of them."

We start by lifting strategies for $\mathcal{G}^{fin}$ to strategies for $\mathcal{G}^{inf}$.

Let $p = p_0 \dots p_k$ be a finite play of $\mathcal{G}^{inf}$.

Assume there are numbers $\ell < m \leqslant k$ such that

$$p_m = p_\ell = y\,.$$

We call such a situation a **repetition**.

We want to consider the first repetition: Let m_0 be the least m such that there is a corresponding ℓ such that $\ell < m_0$ forms a repetition. Note that this ℓ has to be unique is, let us denote it by ℓ_0.

We define purge(p) to be the play in which we delete the segment $p_{\ell_0+1} \dots p_{m_0}$:

$$\text{purge}(p) = p_0 \dots p_{\ell_0} p_{m_0+1} \dots p_k\,.$$

For an arbitrary play p' (that may not necessarily contain a repetition), we define purge$^*(p)$ to be the sequence we get by applying purge as often as possible, i.e. until the resulting sequence contains no repetition any more.

8.15 Lemma
Let p be a finite play of $\mathcal{G}^{inf}$. The sequence purge$^*(p)$ is a valid play of $\mathcal{G}^{fin}$ ending in the same position as p.

Using purge*, we can lift strategies from $\mathcal{G}^{fin}$ to $\mathcal{G}^{inf}$: For a given play of $\mathcal{G}^{inf}$, we apply purge* and then ask the strategy for $\mathcal{G}^{fin}$ for the next move.

8.16 Definition
Let $s^{fin}_{☆}$ be a strategy for a player ☆ for the game $\mathcal{G}^{fin}$. We define a strategy $s^{inf}_{☆}$ for $\mathcal{G}^{inf}$ as follows:

$$s^{inf}_{☆}(p) = s^{fin}_{☆}\big(\text{purge}^*(p)\big)\,.$$

Using the properties of purge*, the following lemma is easy to prove.

8.17 Lemma
If $s^{fin}_{☆}$ is positional, so is $s^{inf}_{☆}$.

The crucial lemma is the following. It shows that lifting the strategies also lifts the value they guarantee.

8.18 Lemma
If $s^{\mathit{fin}}_{\star}$ guarantees value v in $\mathcal{G}^{\mathit{fin}}$, then $s^{\mathit{inf}}_{\star}$ guarantees value v in $\mathcal{G}^{\mathit{inf}}$.

Proof:
We consider the case of the existential player, i.e. $\star = \bigcirc$. The proof for the universal player is similar.

Let $p = p_0p_1p_2\dots$ be an infinite play of $\mathcal{G}^{\mathit{inf}}$ from x_0 that is conform to $s^{\mathit{inf}}_{\bigcirc}$. We need to show $\varphi_{\bigcirc}(p) \leqslant v$.

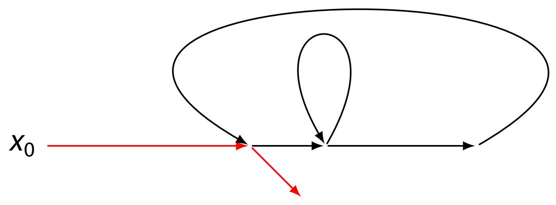
First note that since $\mathcal{G}^{\mathit{inf}}$ is played on a finite graph, after some finite number of steps, each position that will be visited at all has been visited for the first time. From this moment on, p is essentially a sequence of loops. In total, p is a sequence of loops plus a finite prefix.

In each loop, the existential player plays as if all previous loops had not occurred. This is because $s^{\mathit{inf}}_{\star}$ is defined to apply the purge*-operation.

To prove the desired statement, we first provide an estimation for the average of finite prefixes of p. Later, we lift this estimation to the infinite play p.

Let us consider for each $n \in \mathbb{N}$ the finite prefix $p^{(n)} = p_0p_1\dots r_n$ of p. $p^{(n)}$ decomposes into loops and a part of the play that is not contained in any loop.

The following figure depicts a possible decomposition of $p^{(n)}$ into two loops. Only the prefix and the suffix of the play that are marked using red color are not part of any loop.



The idea of the proof is to consider the positions contained in any loop and the ones not contained in any loop separately. The average over the weights occurring in the loops is bounded by v, as the strategy is obtained by lifting a strategy for the finite game $\mathcal{G}^{\mathit{fin}}$ guaranteeing value v. The number of positions not contained in any loop is bounded, thus the corresponding weights do not influence the payoff of the infinite play. This is made precise in the following.

Let us consider the mean over the weights in $p^{(n)}$, i.e. the value

$$\mathrm{avg}^{(n)} = \frac{1}{n+1}\sum_{i=0}^{n} w(p_i)\ .$$

Let $\mathrm{Loop}_1, \ldots, \mathrm{Loop}_k \subseteq \{0, \ldots, n\}$ denote for each j the set of indices $i \in \{1, \ldots, k\}$ such that p_i is part of the j^{th} loop. Let Rest denote the set of indices not contained in any loop.

We can rewrite the expression above by decomposing the sum accordingly, obtaining

$$\mathrm{avg}^{(n)} = \frac{1}{n+1}\left(\sum_{j=1}^{k}\sum_{i\in \mathrm{Loop}_j} w(p_i) + \sum_{i\in\mathrm{Rest}} w(p_i)\right).$$

Observe that each loop together with the part that leads to it (in which we remove all loops that occurred earlier) is a play of $\mathcal{G}^{fin}$. In fact, it is a play that conforms to the strategy $s_{\bigcirc}^{fin}$ that we lifted to obtain $s_{\bigcirc}^{inf}$. Therefore, each such play has payoff at most v in $\mathcal{G}^{fin}$. The payoff function of $\mathcal{G}^{fin}$ was defined to yield the mean over the moves occurring in the loop. Thus, the mean value of the weights of each loop is at most v. Consequently, the total value of each loop is at most its cardinality times v. We obtain the new estimation

$$\begin{aligned}
\mathrm{avg}^{(n)} &\leqslant \frac{1}{n+1}\left(\sum_{j=1}^{k} v\left|\mathrm{Loop}_j\right| + \sum_{i\in\mathrm{Rest}} w(p_i)\right)\\
&= \frac{1}{n+1}\left(v(n+1-|\mathrm{Rest}|) + \sum_{i\in\mathrm{Rest}} w(p_i)\right)\\
&= \frac{1}{n+1}\left(v(n+1) - v\cdot|\mathrm{Rest}| + \sum_{i\in\mathrm{Rest}} w(p_i)\right).
\end{aligned}$$

Let us now consider the expressions involving Rest: The number of moves in Rest is bounded by $|V|$: After going through all positions once, it is not possible to make a move without having a repetition and thus closing a loop. In particular, this bound does not depend on the length n of the play under consideration. This allows us to bound the influence of these nodes on the average by taking their maximum number times the maximal weight assigned to any position of G. Altogether, we may chose a suitable constant c not depending on n such that we have

$$\begin{aligned}\mathrm{avg}^{(n)} &\leqslant \frac{1}{n+1}(v(n+1)+c)\\ &= \frac{v(n+1)}{n+1} + \frac{c}{n+1}\\ &= v + \frac{c}{n+1}\ .\end{aligned}$$

Let us now consider the value $\varphi_{\bigcirc}(p)$ for the infinite play. We have $\varphi_{\bigcirc}(p) = \liminf_{n\to\infty} \mathrm{avg}^{(n)}$ by definition, and thus

$$\varphi_{\bigcirc}(p) \leqslant \liminf_{n\to\infty}\left(v + \frac{c}{n+1}\right) = v\ ,$$

since $\frac{c}{n+1}$ goes to 0 when n becomes large.

This is what we needed to show. ■

Intuitively, we have exploited that an infinite play p consists of infinitely many loops and a negligible (bounded) part not contained in any loop. Since the payoff is a limit, this bounded part does not matter.

The lemma that we have just proven already gives us a part of the desired theorem.

8.19 Corollary
The game $\mathcal{G}^{inf}$ has a value, namely the same value as $\mathcal{G}^{fin}$.

It remains to show that the value of $\mathcal{G}^{inf}$ can be achieved using positional strategies. By the Lemmas 8.17 and 8.18, it is sufficient to show that $\mathcal{G}^{fin}$ has positional strategy guaranteeing the value.

A positional strategy essentially forgets the whole past of the play. To prove this, we will introduce another finite game that has a forgetting-mechanic: as soon as a certain position is visited, the prefix of the play up to this point is forgotten. We will show that even in this "forgetful game", we are able to achieve the same value as in $\mathcal{G}^{fin}$, and then deduce the existence of positional strategies for $\mathcal{G}^{fin}$.

The catch is that to prove this statement, we will need to go back to the infinite game $\mathcal{G}^{inf}$: In an infinite play where the payoff is defined as limit, forgetting a finite prefix of the play certainly will not hurt.

Let us first introduce some notation: For a position $x \in V_\bigcirc$ owned by the existential player, let $\mathcal{G}_x^{fin}$ and $\mathcal{G}_x^{inf}$ be the games that work like $\mathcal{G}^{fin}$ and $\mathcal{G}^{inf}$, but are played from x (instead of x_0)as the initial position.

We can now define the forgetful game.

8.20 Definition
For a node $x \in V_\bigcirc$, the game $\mathcal{G}^x$ is a zero-sum payoff game of bounded length played on the same game arena G and from the same initial position x_0 as $\mathcal{G}^{fin}$ and $\mathcal{G}^{inf}$.

Its plays work as follows:

- As long as position x is not visited in a play, the termination criterion and the payoff function are defined as for $\mathcal{G}^{fin}$.

  This means we stop after the first repetition, and the payoff is the mean of the weights of the positions occurring in the loop.

- If a play p visits x, say $p_k = x$ the game essentially forgets the prefix $r_0 \ldots r_{k-1}$.

  The play continues until a repetition occurs, i.e. until there are $k \leqslant \ell < m$ such that $r_m = r_\ell = y$. The payoff of such a play is

$$\varphi^x(p) = \frac{1}{m-\ell} \sum_{i=\ell+1}^{m} w(p_i) \,.$$

Note that in the second case, we require $k \leqslant \ell < m$, i.e. that the repetition only involves positions that happened after visiting x. We do not consider moves $p_{m'}$ that close a loop started by $p_{\ell'}$ with $\ell' < k$ as repetitions, since we want to forget the prefix $p_0 \ldots p_{k-1}$.

We can rephrase the mechanics of $\mathcal{G}^x$ as follows: As long as the universal player does not move to x, $\mathcal{G}^x$ behaves like $\mathcal{G}^{fin}$. If she does, the game behaves like $\mathcal{G}_x^{fin}$, $\mathcal{G}^{fin}$ started from x, and the prefix leading to x is forgotten.

8.21 Lemma
For each $x \in V_\bigcirc$, $\mathcal{G}^x$ has a value.

This lemma can again be proven by modeling $\mathcal{G}^x$ as a zero-sum payoff game with bounded length.

Crucial is that the value of $\mathcal{G}^x$ is still the same as the value of $\mathcal{G}^{fin}$.

8.22 Lemma
For every $x \in V_\bigcirc$, the value of $\mathcal{G}^x$ equals the value of $\mathcal{G}^{fin}$.

Proof:
Consider a strategy for some player $s_{☆}^{fin}$ for $\mathcal{G}^{fin}$ that guarantees value v. It remains to show that there is a strategy $s_{☆}^{x}$ for $\mathcal{G}^{x}$ also guaranteeing value v.

If in no play of $\mathcal{G}^{fin}$ conforming to $s_{☆}^{fin}$, the universal player ever uses a move leading to x, then $s_{☆}^{fin}$ is also a strategy for $\mathcal{G}^{x}$ that guarantees value v.

Assume that there is a play in which x is visited. Consider the lifted strategy $s_{☆}^{inf}$ for $\mathcal{G}^{inf}$. Then there is a play conforming to $s_{☆}^{inf}$ that visits x in $\mathcal{G}^{inf}$.

We observe that this means that the value of $\mathcal{G}_{x}^{inf}$ is not worse than the value of $\mathcal{G}^{inf}$. (Here, not worse means $\leqslant$ if we consider the existential player, and $\geqslant$ if we consider the universal player.) Each play of $\mathcal{G}_{x}^{inf}$ can be seen as a play of $\mathcal{G}^{inf}$ in which we remove the finite prefix leading to the first visit of x. Since the payoff function in $\mathcal{G}^{inf}$ is a limit, it does not care about removing a finite prefixes.

We may apply Corollary 8.19 to $\mathcal{G}_{x}^{inf}$ and $\mathcal{G}_{x}^{fin}$ to conclude that also the value of $\mathcal{G}_{x}^{fin}$ is not worse than the value of $\mathcal{G}^{fin}$.

Since plays of $\mathcal{G}^{x}$ in which x is visited behave like $\mathcal{G}_{x}^{fin}$, this proves the result: Recall that $s_{☆}^{fin}$ is a strategy for $\mathcal{G}^{fin}$ that guarantees value v. Let $s_{☆}^{finx}$ be a strategy for $\mathcal{G}_{x}^{fin}$ that guarantees value v. (It exists by the previous discussion.)

We define a strategy $s_{☆}^{x}$ for $\mathcal{G}^{x}$ guaranteeing value v as follows: As long as we have not visited x, $s_{☆}^{x}$ behaves like $s_{☆}^{fin}$ would.

Assume the play has already visited x, and let $p = p'p''$ be a decomposition into the prefix leading to x and the rest. Then we define $s_{☆}^{x}(p) = s_{☆}^{finx}(p'')$ to be the strategy for $\mathcal{G}_{x}^{fin}$ applied to the rest. ■

The lemma that we have just proven is now crucial for finally proving the second part of the theorem of Ehrenfeucht and Mycielski. Recall that it is sufficient to prove that the value of $\mathcal{G}^{fin}$ can be achieved using positional strategies.

8.23 Proposition
There are positional strategies for $\mathcal{G}^{fin}$ guaranteeing its value v.

Proof:
We only consider the case of the existential player. The case of the universal player is similar, using a suitable version of $\mathcal{G}^{x}$ for positions $x \in V_{\Box}$.

Let $R_{\bigcirc}$ be the set of arcs originating in positions of the existential player, i.e. $R_{\bigcirc} = R \cap (V_{\bigcirc} \times V_{\Box})$.

We proceed by induction on $m = |R_\bigcirc| - |V_\bigcirc|$. This number m is essentially the number of possible choices for $\bigcirc$: Note that $m \geqslant 0$ has to hold, since the game is deadlock-free.

If $m = 0$, we have $|V_\bigcirc| = |R_\bigcirc|$ and each position has a unique move originating in it. Consequently, each strategy can only pick this move and is necessarily positional.

Assume $m > 0$ holds. Then there is a position $x \in V_\bigcirc$ such that there are several arcs originating in x.

Consider the game $\mathcal{G}^x$. By the previous Lemma, there is a strategy $s^x_\bigcirc$ for $\mathcal{G}^x$ guaranteeing the value v. We can assume that the strategy only uses at most one arc (x, y) originating in x: If x is visited at second time, this closes a loop, and thus the play stops.

We will now consider variants $\mathcal{G}^{fin'}$ and $\mathcal{G}^{x'}$ of the games in which we remove all arcs originating in x but this one arc (x, y). Removing these arcs does not influence the strategy $s^x_\bigcirc$, since it has only used the arc that still exists. Thus it guarantees value v in $\mathcal{G}^{x'}$.

By Lemma 8.22 applied for $\mathcal{G}^{fin'}$ and $\mathcal{G}^{x'}$, this value coincides with the value of $\mathcal{G}^{fin'}$. Since in $\mathcal{G}^{fin'}$, there are strictly less choices for the existential player, so we can apply the induction hypothesis. This yields a positional strategy $s^{pos}_\bigcirc$ guaranteeing value v in $\mathcal{G}^{fin'}$.

We may see this as a strategy for $\mathcal{G}^{fin}$. Since we only restricted the choices of the existential player in $\mathcal{G}^{fin'}$, any play of $\mathcal{G}^{fin}$ conforming to the strategy is also a play of $\mathcal{G}^{fin'}$ conforming to the strategy. We conclude that, since $s^{pos}_\bigcirc$ guarantees value v in $\mathcal{G}^{fin'}$, it also guarantees value v in $\mathcal{G}^{fin}$. ■

Proposition 8.23 together with Corollary 8.19 proves Theorem 8.11.

The complexity of mean payoff games

In the previous subsection, we have followed the development of the original paper by Ehrenfeucht and Mycielski from 1979 [EM79] to prove that mean payoff games have optimal positional strategies. The proof does not yield an efficient algorithm: It is a brute-force approach, as we essentially try out all subgames in which only one choice is possible.

In this subsection, we want to list some complexity results on mean payoff games without giving detailed proofs. Let us assume that all weights are integers, i.e. the weight function has signature $w \colon V \to \mathbb{Z}$. Note that a weight function assigning rational weights can be transformed into an equivalent one assigning integer weights by multiplying all weights with the least common denominator.

The decision problem variant of mean payoff games can be phrased as follows. Note that, as usual for the decision problem variant of optimization problems, we assume that a proposal for the value v is given instead of computing it.

Solving mean payoff games (MEANPAYOFF)

Given: G bipartite deadlock-free game arena, $x_0 \in V$ initial position,
$w\colon V \to \mathbb{Z}$ weight function,
$v \in \mathbb{Q}$,
player ☆ $\in \{\bigcirc, \square\}$

Question: Can ☆ guarantee value at least v?

Here, *at least v* should mean *greater than or equal to v* if the player ☆ is the universal player, and *less than or equal to v* if the player ☆ is the existential player.

First, note that similar to parity games, mean payoff games can be solved in NP as well as in coNP.

8.24 Lemma
MEANPAYOFF $\in$ NP $\cap$ coNP

Proof (sketch):
Similar to the proof of Proposition 6.19 (the analogous result for parity games) the goal is to guess a positional strategy for one of the players and check whether it guarantees the given value. The deterministic polynomial-time algorithm that checks whether a strategy is winning is more involved than the one that proves Lemma 6.18. Assume we have reduced the game arena to a game arena in which only the opponent has choices (using the guessed positional strategy to resolve the choices of the player of interest). One needs to check whether a cycle in this graph exists of which the mean is *worse* than v. To this end, one can use a polynomial time algorithm by Karp [Kar78].

If such a cycle exists, the opponent could force the play to have the *bad* value of the cycle by taking this cycle infinitely often If the mean of all cycles is *better* than v, the guessed strategy is winning.

The algorithm of Karp can be used to determine the maximal resp. minimal mean over a cycle. Therefore, this approach works for strategies for both players, just the meaning of worse has to be adjusted. ■

Even more interesting than the decision problem variant would be to actually compute the precise value of the game (which is the best value each player can guarantee). Furthermore, one would like to obtain positional strategies without using brute force methods. These problems are studied by Zwick and Paterson in their 1996 paper [ZP96].

To determined the value of the game, they consider a variant of the game that is played from an initial position x for a fixed number k of moves. The payoff of a play is the sum of the obtained weights. Let $v_k(x)$ denote the value of this game for initial position x and length k. It can be easily computed using a recursive algorithm, see Exercise 8.29.

The time needed for the recursive computation of the values $v_k(x)$ is in $\mathcal{O}(k \cdot |E|)$. To this end, notice that the recursive computation naively forms a tree, which would be too large, but it can be compacted into a DAG (directed acyclic graph). This is possible by merging multiple occurrences of $v_i(y)$ for the same i and y into one node. The resulting DAG has at most $k \cdot |E|$ many arcs.

To actually implement the algorithm, one should use a dynamic programming approach: We compute the values $v_i(y)$ for all $y \in V$ in a loop for $i = 1, \dots, k$. This allows to evaluate each $v_{i+1}(y)$ by looking up the values for v_i without recursive call.

Intuitively, the value v of the mean payoff game should be obtainable as the limit

$$\lim_{k\to\infty} \frac{v_k(x_0)}{k} .$$

This indeed holds, as stated by the following result.

In the following, we will use $n = |V|$ to refer to the number of positions, and $W = \max_{x \in V} |w(x)|$ for the the greatest absolute value of the weight function.

8.25 Proposition

$$k \cdot v - 2nW \leqslant v_k(x_0) \leqslant k \cdot v + 2nW.$$

Note that $2nW$ does not depend on k. Using the sandwich criterion, we get that $v_k(x_0)$ has the same limit behavior as $k \cdot v$, and thus $\frac{v_k(x_0)}{k}$ converges to v for $k \to \infty$.

8.26 Theorem
The value v of a mean payoff game can be computed in time $\mathcal{O}(n^3 \cdot |E| \cdot W)$.

Proof sketch / Algorithm:
Compute the value $v_k(x_0)$ for $k = 4|V|^3 W$. As stated above, this is possible in $\mathcal{O}(n^3 W \cdot |E|)$. Define $v' = v_k(x_0)/k$. Using Proposition 8.25, we obtain

$$v' - \frac{2nW}{k} \leqslant v \leqslant v' + \frac{2nW}{k} .$$

By plugging in the definition of k, we can obtain the estimation

$$v' - \frac{1}{2n(n-1)} < v < v' + \frac{1}{2n(n-1)} .$$

We use (without proof) that v can be written as a rational number with denominator at most n.

The index k was chosen such that the interval provided by the estimation contains a unique such number, which then has to be v. ■

We now turn to optimal positional strategies.

8.27 Theorem
For both players, optimal positional strategies can be computed in time $\mathcal{O}(n^4 \cdot |E| \cdot \log\left(\frac{|E|}{n}\right) \cdot W)$.

Proof sketch / Algorithm:
The idea is to use the fact that the existence of positional optimal strategies guarantees that for each position, we can select a unique outgoing arc that should be used by the strategies. We select candidate arcs and test whether the value of the game stays the same, if yes, we have chosen the correct arc.

Instead of doing this one by one, we use binary search to detect in a logarithmic number of steps for each position the optimal move. Zwick and Paterson call this approach a group test strategy. We use the fact that we can compute the value of a game using the previous algorithm, even if the game involves choices by both players.

The algorithm is as follows:

- Compute the values v for the game using the algorithm above.
- For all positions with only one outgoing arc, the move made by the strategy is fixed.
- As long as there is a position y with out-degree $d > 1$:

  Let R_y be the set of arcs outgoing from this position. We may partition it into non-empty sets $R_y = R_y^1 \uplus R_y^2$ of size $\lfloor d/2 \rfloor$ resp. $\lceil d/2 \rceil$.

  Consider the game $\mathcal{G}^j$ in which all arcs from R_y but the arcs in R_y^j have been removed, for $j \in \{1, 2\}$. Compute their values v^j using the algorithm above. If $v = v^j$, then there is an optimal positional strategy for the player owning y in which she choses an arc in R_y^j.

Repeat this approach and refine until a single optimal move has been identified from every position.

■

Note the the algorithms presented here a pseudo-polynomial, as they are polynomial in the size of G and W. This means they are polynomial assuming weights are encoded in unary. Note that $W = 2^{\log W}$ is exponential in its binary encoding, i.e. the algorithms are not polynomial using an usual binary encoding of the input.

If the maximum weight is small, the algorithms presented here are efficient. If the maximum weight is very high compared to the size of the graph, the brute-force-approach that is exponential in the size of G, but essentially independent of W, could be better.

Exercises

8.28 Exercise

Let $\mathcal{G}$ be a zero-sum game. Prove that if $\mathcal{G}$ has a value, it has a unique value.

Use the definitions in your proof, do not use $v_{\bigcirc} \geqslant v_{\Box}$ or the minmax theorem.

8.29 Exercise

Let $\mathcal{G}$ be a length-k payoff game on a finite game arena $G = (V_{\Box} \uplus V_{\bigcirc}, R)$ for some initial position $x_0 \in V$. We assume that there is a weight-function $w\colon V \to \mathbb{Z}$ assigning each position its weight as an integer. The value of the payoff function φ of a play $p = p_0 p_1 p_2 \ldots r_n$ of length $n \leqslant k$ is defined as follows:

$$\varphi(p) = \sum_{i=0}^{n} w(p_i)\,.$$

Present a recursive algorithm determining the value v of such a game.

Hint: For each position y and each number $n \leqslant k$, define $v_k(y)$ as the value achieved in the game where we see y as the initial position and n as the bound on the length of plays. Show how to compute these values.

8.30 Exercise

Consider the game $\mathcal{G}^{fin}$ from Definition 8.13.

a) Prove that if the positions corresponding to the indicesod $\ell < m$ form a repetition, then $\ell + m$ is even.

b) Derive a bound on the length of maximal plays of $\mathcal{G}^{fin}$.

c) Prove that $\mathcal{G}^{fin}$ has a value by modeling $\mathcal{G}^{fin}$ as a zero-sum game of bounded length.

8.31 Exercise

Formally prove Lemma 8.15 and conclude the statement of Lemma 8.17.

8.32 Exercise

a) Throughout the whole lecture, we have assumed that the game arena is parallel-free, meaning there is at most one arc from some position to another.

   Assume you are given a game arena that is not parallel free. Show how to construct an equivalent game arena that is parallel-free

   - for reachability/Büchi/parity games,
   - for mean payoff games.

b) In this section, we have assumed that the game arena is bipartite and the players alternately take turns.

   Assume you are given a non-bipartite game arena. Show how to construct an equivalent bipartite game arena for reachability/Büchi/parity games.

   Does this also work for mean-payoff games?

Here, by equivalent game arena, we mean that the old set of positions V is a subset of the set of positions of the new game arena V', i.e. $V \subseteq V'$. Furthermore, we want that a position $x \in V \subseteq V'$ is winning for player ☆ in the old game if and only if x is winning for ☆ in the new game.

8.33 Exercise

Let G be a finite, bipartite, deadlock-free game arena, and let B be the winning set for a Büchi game from a fixed initial position x_0.

Show how this Büchi game can be transformed into a mean payoff game.

Assume the initial position x_0 is winning for some player ☆ in the Büchi game. How is this reflected in the mean payoff game? Make your argumentation formal!

Part III.

Games on infinite graphs

Contents

9. An undetermined Gale-Stewart game

So far, we have only considered games that are determined: For each initial position, exactly one of players has a winning strategy. In this chapter, we will see that one can construct undetermined games.

Sources
The content of this section is mostly based on Yurii Khomskii's notes [Kho].

Gale-Stewart games

The games that we will look at in this section are not played on a graph, they have a much simpler shape.

9.1 Definition: Gale-Stewart games
Let A be a set of **actions**. Let $B \subseteq A^{\omega}$ be a set of infinite sequences over A, called the **winning set** (or winning condition, or payoff set). The **Gale-Stewart game** $\mathcal{G}(A, B)$ with respect to A and B is played as follows:

- The players alternately take turns, starting with the existential player.
- In each turn i, the player whose turn it is picks an action $a_i \in A$.
- A maximal play is an infinite sequence

  $$p = a_0 a_1 a_2 \dots$$

  in which the actions a_i with i even have been picked by the existential player and the actions a_i with i odd have been picked by the universal player.
- Such an infinite play p is won by the existential player if and only if $p \in B$.

In other words, the game is played by both players alternately naming actions of their choice, without any restrictions. Note that the plays in which the existential player is active are exactly the plays $a_0 \dots a_n$ where n is odd (including the empty play ε).

We may see $\mathcal{G}(A, B)$ as a game on the graph $A \times \{\bigcirc, \square\}$ in which arcs $(a, \star) \to (a', \overline{\star})$ exist for all $a, a' \in A$. The winning set B defines the winning condition. We make this precise in Exercise 9.13.

Similarly, any graph game can be seen as Gale-Stewart game with the set of nodes V as the set of actions. To this end, the graph structure has to be encoded into the winning set, i.e. the set B is such that if a player picks an illegal move in the graph, she loses. We consider this construction for the case of reachability games in Exercise 9.14.

(Winning) strategies can be defined as usual. Here, we fix the empty sequence ε as the initial position of interest.

9.2 Definition: Strategies, and winning
A **strategy** $s_{☆}$ for player ☆ is a function

$$s_{☆}\colon \{p \in A^* \mid p = a_0 \ldots a_i,\ i \text{ is odd (if } ☆ = \bigcirc\text{) resp. even (if } ☆ = \Box\text{)}\} \to A$$

that takes a finite play p in which it is player ☆'s turn and selects the next action $s_{☆}(p) \in A$ that ☆ should pick.

We call a strategy **winning** if any play (starting in ε) conforming to it is winning.

Player ☆ wins the Gale-Stewart game $\mathcal{G}(A, B)$ if she has a winning strategy.

Let us consider some easy examples.

9.3 Example
Let $A = \{0, 1\}$.

a) Consider $B = A^*00A^\omega \cup A^*11A^\omega$. The winning plays are exactly the plays in which there are two consecutive occurrences of the same letter.

   Note that the existential player can win in her second move (overall the third move) by repeating the action picked by the universal player in the second move.

b) Consider $B = A^*000A^\omega \cup A^*111A^\omega$. The winning plays are exactly the plays in which there are three consecutive occurrences of the same letter.

   The universal player can ensure her win by avoiding the repetition of any action used by the existential player.

9.4 Remark
Gale-Stewart games are named after the American mathematician David Gale (1921 – 2008) and F. M. Stewart. Together, they founded the research on infinite games with perfect information.

An undetermined game and transfinite induction

Our goal is to show that it is possible to pick A, B such that $\mathcal{G}(A, B)$ has no winner: Each infinite play is won by one of the two players, but none of the players has a systematic way of winning.

9.5 Theorem: Existence of undetermined games
There is an undetermined Gale-Stewart game: There are sets A and $B \subseteq A^{\omega}$ such that none of the players has a winning strategy for $\mathcal{G}(A, B)$.

The rest of this subsection is dedicated to proving the theorem.

We can pick $A = \mathbb{N}$ as the set of natural numbers. Note that this set is not finite, which will be needed in the proof.

Choosing $B \subseteq \mathbb{N}^{\omega}$ is much harder and requires a bit of preparation.

Crucial in our development is the fact that even for a fixed strategy $s_{☆}$, the number of plays conforming to $s_{☆}$ is immense. More precisely, it is not even countable.

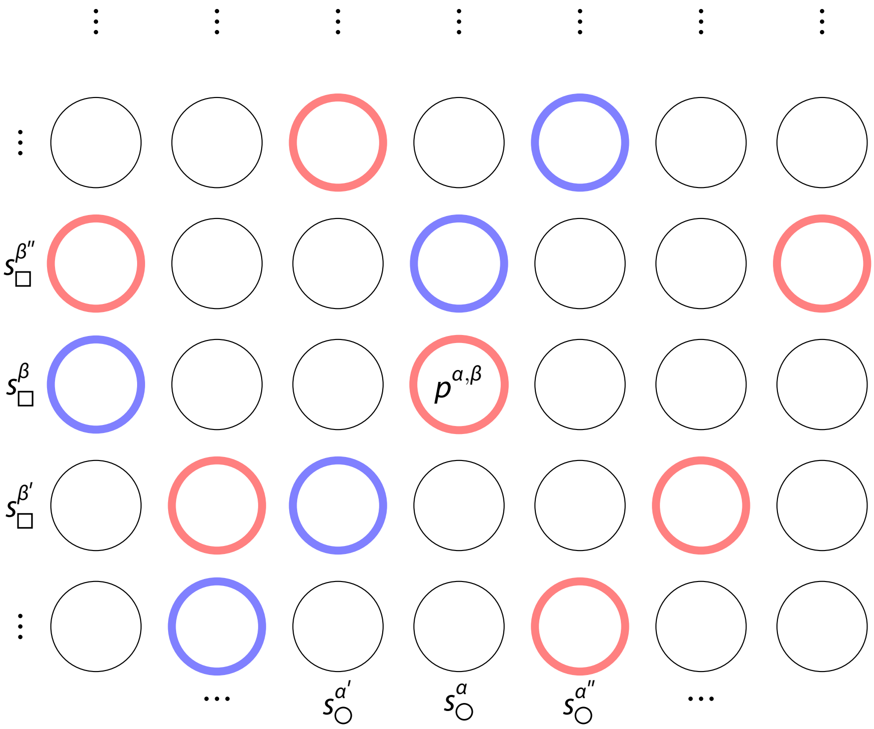
This will allow us to to pick B such that for each strategy s for any of the players

- there is at least one play conforming to the strategy in B, and

- there is at least one play conforming to the strategy not in B.

Since a winning strategy for the existential player has to guarantee that all plays are in B, and a winning strategy for the universal player has to guarantee that no play is in B, this set B results in an undetermined game.

The following graphic depicts this schematically. The columns represent strategies $s_{\bigcirc}^{\alpha}$ for the existential player, the rows represent strategies $s_{\square}^{\beta}$ for the universal player. The node $p^{\alpha,\beta}$ in row $s_{\bigcirc}^{\alpha}$ and column $s_{\square}^{\beta}$ represents the (unique) play in which each player plays conforming to her strategy.

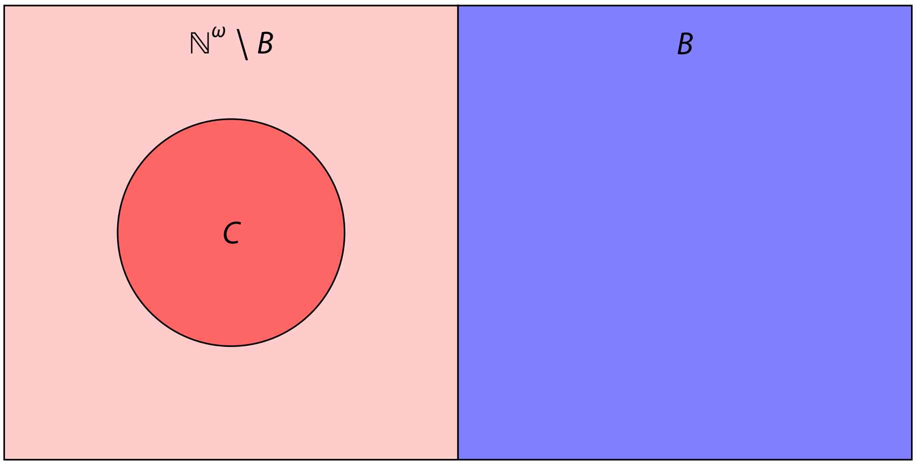


For each row, we guarantee that it contains a play in B (marked with a blue border), so the universal player has no winning strategy. Similarly, we guarantee that at least one play per column is not in B (marked with a red border), meaning that the existential player has no winning strategy.

9.6 Remark

The graphic may wrongfully give the impression that the number of plays and strategies is countable. This is not true, both sets are uncountable as we will see in the proof.

In fact, we will not only construct B, but we will also construct a set $C \subseteq \mathbb{N}^\omega \setminus B$ in the complement of B. We will guarantee that each strategy of $\bigcirc$ has a play in C and that each strategy for $\square$ has a play in B.



It remains to construct the sets B and C by picking two plays for each strategy. To do so, we apply a concept called **transfinite induction** to the set of strategies. Transfinite induction lifts the proof principle of induction from finite sets to arbitrary well-ordered sets.

We recall the definition of a well-ordering.

9.7 Definition
Let $\mathcal{I}$ be a set, and $\leqslant \subseteq \mathcal{I} \times \mathcal{I}$ be a relation on $\mathcal{I}$.

We call the tuple $(\mathcal{I}, \leqslant)$ a **well-order** if $\leqslant$ satisfies the following conditions:

- $\leqslant$ is a **partial order** on $\mathcal{I}$, i.e. reflexive, transitive, and antisymmetric.
- $\leqslant$ is a **total order**, i.e. any two elements from $\mathcal{I}$ are comparable:

$$\forall \alpha, \beta \in \mathcal{I} \colon \alpha \leqslant \beta \text{ or } \beta \leqslant \alpha \ .$$

- $\leqslant$ is **well-founded**: Every non-empty subset $J \subseteq \mathcal{I}$ contains a $\leqslant$-minimal element, i.e. an element $\alpha \in J$ such that there is no $\beta \in J$, $\beta \neq \alpha$ with $\beta \leqslant \alpha$.

9.8 Remark
If we assume that $\leqslant$ is total, the condition of being well-founded can be rephrased as follows: Every non-empty subset $J \subseteq \mathcal{I}$ contains a least element, i.e. an element $\alpha \in J$ with $\alpha \leqslant \beta$ for all $\beta \in J$,

$$\forall J \subseteq \mathcal{I}, J \neq \varnothing \colon \exists \alpha \in J \ \forall \beta \in J \colon \alpha \leqslant \beta \ .$$

This least element is unique by antisymmetry.

An equivalent formulation is that there is no infinite strictly-descending chain in $\mathcal{I}$, i.e. there is no infinite sequence

$$\alpha_0 > \alpha_1 > \alpha_2 > \alpha_3 > \dots$$

of elements $\alpha_i \in \mathcal{I}$.

Here, strictly smaller $<$ is defined as usual, $\alpha < \beta$ iff $\alpha \leqslant \beta$ and $\alpha \neq \beta$. We write $\alpha \geqslant \beta$ resp. $\alpha > \beta$ for $\beta \leqslant \alpha$ resp. $\beta < \alpha$.

Transfinite induction proceeds as follows: Let $A(\alpha)$ be a statement that is parametric in an element $\alpha \in \mathcal{I}$, where $(\mathcal{I}, \leqslant)$ is well-ordered. If we want to show $\forall \alpha \in \mathcal{I} \colon A(\alpha)$ (i.e. A is true for any α), we can proceed as follows:

- Base case: The statement holds for the least element of $\mathcal{I}$, and
- Inductive step: Assuming that the statement holds for all elements $\beta \in \mathcal{I}$ that are strictly smaller than some $\alpha \in \mathcal{I}$, the statement also holds for α.

If the well-ordered set of consideration is $(\mathbb{N}, \leqslant)$, with $\leqslant$ defined as usual, transfinite induction and the usual induction proof principle coincide.

9.9 Remark

We shortly argue that the proof principle of transfinite induction is sound. Consider some statement $A(\alpha)$ that is parametric in $\alpha \in \mathcal{I}$, $\mathcal{I}$ well-ordered. Assume we have shown that, for any fixed α, that $A(\beta)$ holding for all $\beta < \alpha$ implies that $A(\alpha)$ holds.

We claim that this proofs that $\forall \alpha \in \mathcal{I} \colon A(\alpha)$ holds. Towards a contradiction assume that this is not true. Consider the subset ` $= \{\alpha' \in \mathcal{I} \mid A(\alpha')$ does not hold$\}$. By assumption, this set is non-empty, so it contains some least element $\alpha_0 \in$ `. Because α_0 is the least element from `, all strictly smaller elements are not in `. Consequently, $A(\beta)$ holds for all $\beta < \alpha_0$. Using our initial assumption, we obtain that $A(\alpha_0)$ has to hold, a contradiction.

Note that the base case was not needed in the proof. In fact, the case of the least element is the case in which the set of elements that are strictly smaller is empty.

Our goal is to apply transfinite induction to the set of all strategies to pick the sets B and $C \subseteq \mathbb{N}^\omega \setminus B$. However, the set of strategies does not come with a natural well-ordering. In fact, it is not even possible to actually construct such a well-ordering explicitly. Still we can apply the following lemma to equip it with one. The lemma can be seen as a stronger version of the **well-ordering** theorem, on which the proof relies.

9.10 Lemma

For any set X, there is a well-order $(\mathcal{I}, \leqslant)$ such that

(1) $|X| = |\mathcal{I}|$, i.e. X and $\mathcal{I}$ have the same size, and

(2) For any $\alpha \in \mathcal{I}$, the set of elements strictly smaller than α

$$\{\beta \in \mathcal{I} \mid \beta < \alpha\}$$

has cardinality strictly smaller than $|X| = |\mathcal{I}|$.

9.11 Remark

Recall that the cardinality for infinite sets is defined using functions:

- $|X| \leqslant |Y|$ if there is an injection $f: X \to Y$,
- $|X| \geqslant |Y|$ if there is a surjection $f: X \to Y$, and
- $|X| = |Y|$ if there is a bijection $f: X \to Y$.

The cardinality of X is strictly smaller than the cardinality of Y if $|X| \leqslant |Y|$ and $|X| \neq |Y|$, i.e. there is an injection, but no bijection from X to Y.

Note that for finite sets, these definitions coincide with the usual ones, see Exercise 9.15. For example, we write $|X| = 5$ if $|X| = |\{0, \ldots, 4\}|$.

Property (1) of $\mathcal{I}$ of the lemma is essentially just the well-ordering theorem: On any set, there is an order such that the set together with the order is a well-order. Property (2) of $\mathcal{I}$ in the lemma gives us a stronger property that we will need in the proof of Theorem 9.5.

9.12 Example

- The natural numbers are well-ordered by $\leqslant$ (defined as usual). Additionally, $(\mathbb{N}, \leqslant)$ satisfies the second property in the Lemma:

  For each number n, the set $\{m \in \mathbb{N} \mid m < n\}$ has cardinality $n < |\mathbb{N}|$.

- Consider the real numbers $\mathbb{R}$ with $\leqslant$ defined as usual. This is not a well-ordering: The open interval $]0, \infty[$ has no least element. We still may check whether Property (2) holds: For any $a \in \mathbb{R}$, the set of elements smaller than a is the open interval $]-\infty, a[$. It has the same cardinality as $\mathbb{R}$ itself, thus Property (2) does not hold.

For the sake of completeness, we give the proof of the lemma. It uses properties of cardinal and ordinal numbers. If you are not familiar with these concepts, you can skip the proof and treat the lemma as a black box result.

Proof of Lemma 9.10:

By the well-ordering theorem, any set can be well-ordered. This means there is a relation $\leqslant \subseteq X \times X$ such that $(X, \leqslant)$ is a well-order.

For every well-order, there is an ordinal number α that is order-isomorphic to it. (I.e. there is a bijection $f: X \to \alpha$ such that for $x, y \in X$ we have $x \leqslant y$ iff $f(x) \leqslant f(y)$.)

Pick κ as the least ordinal in bijection to α, i.e. $|\alpha| = |\kappa|$. The least ordinal with a certain cardinality is a cardinal number, and we have $|\kappa| = |\alpha| = |X|$ (recall that f was a bijection).

This proves that κ satisfies Property (1).

For any $\gamma \in \kappa$, we need to consider the set

$$\{\beta \in \kappa \mid \beta < \gamma\} .$$

Using the von Neumann definition of ordinals, the definition of the set simplifies to

$$\{\beta < \kappa \mid \beta < \gamma\} = \{\beta \mid \beta < \gamma\} = \gamma .$$

Since $\gamma \in \kappa$, we certainly have $|\gamma| \leqslant |\kappa|$.

If equality would hold, then γ would be a ordinal smaller than κ that is in bijection to α. This would be a contradiction to the choice of κ. Therefore, we obtain $|\gamma| < |\kappa|$. This proves that κ satisfies Property (2) and is thus as required. ■

We are now prepared to prove the existence of undetermined games.

Proof of Theorem 9.5:
The proof will proceed in four steps.

1. We will determine the number of strategies and the number of plays conforming to a fixed strategy.
2. We use transfinite induction to pick the sets of plays B and C.
3. We show $B \cap C = \varnothing$, which proves that C lives inside the complement of B.
4. We prove that $\mathcal{G}(A, B)$ is undetermined as outlined above: Each strategy for the existential player has a play in C (and thus not in B), each strategy for the universal player has a play in B.

1. Preliminaries

A strategy for a player is essentially a function

$$s_{\text{☆}} \colon \mathbb{N}^* \to \mathbb{N}$$

picking for each finite prefix the next action in $\mathbb{N}$. Note that $\mathbb{N}^*$ is countably infinite, and therefore isomorphic to $\mathbb{N}$.

Consequently, the number of strategies for each player is

$$\left|\mathbb{N}^* \to \mathbb{N}\right| = |\mathbb{N} \to \mathbb{N}| \overset{def}{=} 2^{\aleph_0} .$$

Here, you may see $2^{\aleph_0}$ as a name that we use for the size of the set $\mathbb{N} \to \mathbb{N}$.

Remark: $\aleph_0$ is usually used to denote the cardinality of $\mathbb{N}$. One can indeed show that $|\mathbb{N} \to \mathbb{N}| = |\mathcal{P}(\mathbb{N})|$, so the name $2^{\aleph_0}$ for the size of $\mathbb{N} \to \mathbb{N}$ makes sense.

Let $\mathrm{Strat}_{\bigcirc}$ be the set of all possible strategies for the existential player. We apply Lemma 9.10 to it to obtain a well order $(\mathcal{I}, \leqslant)$ that satisfies the Properties (1) and (2). The set $\mathcal{I}$ has the same cardinality, thus there is a bijection $f{:}\,\mathcal{I} \to \mathrm{Strat}_{\bigcirc}$. For each element $a \in \mathcal{I}$, let $s_{\bigcirc}^{a}$ denote the strategy $f(a)$. We may write

$$\mathrm{Strat}_{\bigcirc} = \left\{ s_{\bigcirc}^{a} \mid a \in \mathcal{I} \right\} .$$

We apply the same argumentation for the set $\mathrm{Strat}_{\square}$ of possible strategies for the universal player. Since $|\mathrm{Strat}_{\bigcirc}| = |\mathrm{Strat}_{\square}| = 2^{\aleph_0}$, we may even use the same well-order $(\mathcal{I}, \leqslant)$. Again, there is a bijection (say g with $g(a) = s_{\square}^{a}$) between $\mathcal{I}$ and $\mathrm{Strat}_{\square}$, and we can write

$$\mathrm{Strat}_{\square} = \left\{ s_{\square}^{a} \mid a \in \mathcal{I} \right\} .$$

For a strategy $s_{☆}$, let $Plays_{inf}(s_{☆})$ denote the set of all maximal plays in which player ☆ moves conforming to her strategy $s_{☆}$. The cardinality of each set $Plays_{inf}(s_{☆})$ is

$$\left|Plays_{inf}(s_{☆})\right| = 2^{\aleph_0} ,$$

as there is essentially one play per strategy of the opponent.

More precisely, for each prefix $\mathbb{N}^*$ of suitable length (even or odd), the opponent has one choice per number in $\mathbb{N}$ with which she can react. Therefore, we obtain

$$\left|Plays_{inf}(s_{☆})\right| = |\mathbb{N} \to \mathbb{N}| = 2^{\aleph_0} .$$

2. Defining *B* and *C*

We will now define two sets B and C using transfinite induction on $(\mathcal{I}, \leqslant)$.

They will have the shape

$$B = \{b_\alpha \mid \alpha \in \mathcal{I}\}\,,$$
$$C = \{c_\alpha \mid \alpha \in \mathcal{I}\}\,,$$

where the b_α and c_α are defined by simultaneous induction. In each step of the induction, say for $\alpha \in \mathcal{I}$, we first pick b_α, then c_α.

Our goal is enforce that the sets are disjoint. To this end, we ensure that each b_α is not equal to any previously picked c_β for $\beta < \alpha$, and each c_α is not equal to any previously picked b_β, $\beta \leqslant \alpha$.

Base case:

Let $\alpha_0 \in \mathcal{I}$ denote the least element of $\mathcal{I}$ (with respect to the well-ordering $\leqslant$). It exists because we may see $\mathcal{I} \subseteq \mathcal{I}$ as a non-empty subset of itself.

- *Picking $b_{\alpha_0} \in B$:*
  Consider the strategy $s_\square^{\alpha_0} \in \mathrm{Strat}_\square$, and pick b_{α_0} as an arbitrary play in $Plays_{inf}(s_\square^{\alpha_0})$.
- *Picking $c_{\alpha_0} \in C$:*
  Consider $s_\bigcirc^{\alpha_0}$. The set $Plays_{inf}(s_\bigcirc^{\alpha_0})$ has more than one element, so $Plays_{inf}(s_\bigcirc^{\alpha_0}) \setminus \{b_0\}$ is non-empty. Pick c_{α_0} as an arbitrary element from this set.

Induction hypothesis:

Let $\alpha \in \mathcal{I}$ be fixed and suppose that for all $\beta < \alpha$, the elements b_β and c_β have already been chosen.

Induction step:

We have to chose b_α and c_α.

- *Picking $b_\alpha \in B$:*
  Consider the set of elements
  $$\{c_\beta \mid \beta < \alpha\}$$
  of C that have already been chosen. It has at most the same cardinality as
  $$\{\beta \in \mathcal{I} \mid \beta < \alpha\}\,,$$
  since the function defined by $\beta \mapsto c_\beta$ is a surjection. By Property (2) of Lemma 9.10, we have that this second set has cardinality strictly smaller than $|\mathcal{I}| = |\mathrm{Strat}_\square| = 2^{\aleph_0}$.

As discussed, $Plays_{inf}(s^{\alpha}_{\square})$ has cardinality $2^{\aleph_0}$. Thus, the set

$$Plays_{inf}(s^{\alpha}_{\square}) \setminus \{c_\beta \mid \beta < \alpha\}$$

is nonempty, and we may define b_α to be an arbitrary element in this set.

- *Picking $c_\alpha \in C$:*
  Similarly, the set
  $$\{b_\beta \mid \beta < \alpha\}$$
  of previously picked elements of B has cardinality less than $2^{\aleph_0}$. If we add the element $b_\alpha \in B$ that we have picked above, this still holds true. (Adding a single element to an infinite set does not change its cardinality). Consequently, the cardinality of
  $$\{b_\beta \mid \beta < \alpha\} \cup \{b_\alpha\}$$
  is strictly smaller than the cardinality of $\mathcal{I}$, which is $2^{\aleph_0}$. Therefore, we may pick an arbitrary element c_α from the non-empty set
  $$Plays_{inf}(s^{\alpha}_{\bigcirc}) \setminus \left(\{b_\beta \mid \beta < \alpha\} \cup b_\alpha\right).$$

Note that each element b_α is a play, and thus a sequence of natural numbers. We have $B, C \subseteq \mathbb{N}^\omega$.

The desired undetermined Gale-Stewart game is $\mathcal{G}(\mathbb{N}, B)$, where $B = \{b_\alpha \mid \alpha \in \mathcal{I}\}$.

Towards proving that $\mathcal{G}(\mathbb{N}, B)$ is not determined, let us first prove that C lives inside the complement of B, $B \cap C = \emptyset$.

3. Claim: $B \cap C = \emptyset$

Let $b \in B$ be arbitrary. By the definition of B, there is an $\alpha \in \mathcal{I}$ such that $b = b_\alpha$. Note that when picking b_α in the inductive step, we made sure that b_α is not equal to c_β for any $\beta < \alpha$. When picking any c_γ for $\gamma \geqslant \alpha$, we make sure that c_γ is neither equal to b_γ, nor to any b_β for $\beta < \gamma$. Thus, any such c_γ is not equal to b_α.

Consequently, $b = b_\alpha$ is not contained in C.

We can now conclude the desired result.

4. Claim: $\mathcal{G}(\mathbb{N}, B)$ is undetermined

We consider an arbitrary strategy for each of the players and show that is cannot be winning.

- *For the universal player:*
  Assume the universal player has a winning strategy $s_\square \in \mathrm{Strat}_\square$. This strategy has to ensure that no play conforming to it is in the winning set B, thus $Plays_{inf}(s_\square) \cap B = \varnothing$.

  There is an $\alpha \in \mathcal{I}$ such that $s_\square = s^\alpha_\square$. When picking $b_\alpha \in B$, we have chosen it in $Plays_{inf}(s^\alpha_\square)$.

  Thus, $B \cap Plays_{inf}(s^\alpha_\square)$ contains b_α and is non-empty, a contradiction.

- *For the existential player:*
  Assume the existential player has a winning strategy $s_\bigcirc \in \mathrm{Strat}_\bigcirc$. This strategy has to ensure that all plays conforming to it are in the winning set B, thus $Plays_{inf}(s_\bigcirc) \subseteq B$.

  There is an $\alpha \in \mathcal{I}$ such that $s_\bigcirc = s^\alpha_\bigcirc$. When picking $c_\alpha \in C$, we picked $c_\alpha \in Plays_{inf}(s^\alpha_\bigcirc)$. Since $B \cap C = \varnothing$ by the previous claim, this proves $c \in Plays_{inf}(s_\bigcirc) \setminus B$. This disproves the inclusion $Plays_{inf}(s_\bigcirc) \subseteq B$, a contradiction.

■

Exercises

9.13 Exercise: Gale-Stewart games as graph games

Let $\mathcal{G}(A, B)$ be a Gale-Stewart game. Define an equivalent game over a graph with set of positions

a) $V = A \times \{\bigcirc, \square\}$,

b) $V = A^*$.

In each case, specify the ownership, the arcs, the winning condition and the initial position of interest.

9.14 Exercise: Reachability games as Gale-Stewart games

Let $\mathcal{G}$ be a reachability game, specified as usual by a game arena $G = (V_\square \uplus V_\bigcirc, R)$ and a winning set $V_{reach} \subseteq V$. For simplicity, let us assume that G is deadlock-free and bipartite: Any move from some position $x \in V_\bigcirc$ leads to a position $y \in V_\square$ and vice versa, i.e. the players take turns alternately. Furthermore, we fix the initial position $x_0 \in V_\square$.

Design a Gale-Stewart game $\mathcal{G}(V, B)$ where the actions are nodes of G and B is such that

1. the existential player loses if she does not start in x_0,

2. the existential player loses if she picks an illegal move, i.e. if the play p is of the shape $p = p'.x_i.x_{i+1}.p''$ where i is odd and $(x_i, x_{i+1}) \notin R$,
3. the universal player loses if she picks an illegal move, i.e. if the play p is of the shape $p = p'.x_i.x_{i+1}.p''$ where i is even and $(x_i, x_{i+1}) \notin R$,
4. any play that does not fall into one of the Cases 1. to 3. is won by the existential player if and only if it contains a position from V_{reach}.

Argue briefly that your set B enforces the desired behavior.

Note: If a play falls into several cases, i.e. into 2. and 3. if both players cheat, you may resolve this as you wish.

9.15 Exercise: Cardinality and functions

Recall that a function $f: X \to Y$ is called **injective** (or an **injection**) if for $x \neq x'$ we have $f(x) \neq f(x')$. A function is called **surjective** (or a **surjection**) if for any $y \in Y$, there is some $x \in X$ with $f(x) = y$. It is called **bijective** (or a **bijection**) if it is both injective and surjective.

In the rest of this exercise, assume that $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_m\}$ are finite sets.

a) Prove that if there is an injection $f: X \to Y$ if and only if $|X| \leqslant |Y|$.

b) Prove that if there is a surjection $f: X \to Y$ if and only if $|X| \geqslant |Y|$.

c) Prove that there is an injection $g: Y \to X$ if and only if there is a surjection $f: X \to Y$.

d) Prove that if there is an injection $f_i: X \to Y$ and a surjection $f_s: X \to Y$, then there is a bijection $f_b: X \to Y$.

Note: The properties that you have proven in Part c) and d) also hold for infinite sets. However, their proof is much more complicated in this case and involves the axiom of choice.

10. Infinite games on the configuration graphs of automata

In the previous section, we have seen that games on an infinite game arena may be undetermined. In this section, we will see a general way to obtain games on infinite arenas that are not only determined, but also decidable in some cases. To be able to obtain an algorithm that computes the winner, we consider games that have a **finite syntactic representation**. Here, we use structures known from automata theory.

Sources
The content of this section is common knowledge in automata theory and does not follow any particular source.

Automata and counter machines

Recall that a **transition system** (V, R) consists of a set of **configurations** V (usually infinite) and a **transition relation** $R \subseteq V \times V$.

Conceptually, an automaton is a transition system such that $V = Q \times M$, where Q is a *small* set of **control states** (in particular: a finite set), and M is the **memory** (which is potentially infinite). Furthermore, the transition relation acts on the memory in a **local** way that admits a finite description. More precisely, there should be a **finite set of rules** $\rightarrow$ such that the set of transitions R consists of precisely the transitions that satisfy one of the rules.

One usually calls $A = (Q, \rightarrow)$ the **automaton**, and the transition system $(Q \times M, R)$ its **configuration graph**.

10.1 Example: Automata

a) A **finite-state system** is an automaton with $M = \{1\}$, i.e. there is no memory. In this case, we essentially have $\rightarrow = R \subseteq Q \times Q$.

b) A **pushdown system** is an automaton with $M = S^*$ for some finite stack alphabet S, i.e. it consists of a finite control and an unbounded stack as storage. The rules in $\rightarrow$ are of the shape $\rightarrow \subseteq (Q \times S) \times (Q \times S^*)$: A transition only depends on the tuple (q, s) formed by the control state and the topmost stack symbol.

c) A **Turing machine** is an automaton with $M = S^* \times S \times S^*$, i.e. the storage is a finite tape that decomposes into the part left of the head position, the tape content at the head position, and the part right of the head position. The transitions only depend on the control state and on the symbol at the current head position.

10.2 Remark

All these automata models are versions of the well-known finite automata, pushdown automata and Turing machines that do not read any input (or produce any output).

It is easy to extend the notion of transition systems by initial and final configurations and in- or output.

Let us consider another automaton model for which the set of configurations is infinite.

10.3 Definition: Counter machine

A **counter machine of dimension** $d \in \mathbb{N}$ is an automaton $A = (Q, \rightarrow)$ over memory $\mathbb{N}^d$. More precisely:

- Q is a finite set of control states
- Let $\mathcal{X} = \{x_1, x_2, \ldots, x_d\}$ be a set consisting of d **counters**. The set of **operations** is given by

  $$Ops = \{noop\} \cup \bigcup_{x \in \mathcal{X}} \{x++, x--, x = 0, x \neq 0\} .$$

  $\rightarrow \subseteq Q \times Ops \times Q$ is a finite set of rules. Instead of $(q, op, q') \in \rightarrow$, we usually write $q \xrightarrow{op} q'$, i.e. $q \xrightarrow{x_1 \neq 0} q'$.

A configuration of such a counter machine is of the shape $(q, \vec{c})$, where $q \in Q$ is a control state and $\vec{c} = (c_1, \ldots, c_d) \in \mathbb{N}^d$ is a vector of non-negative integer **counter values**. Consequently, $\Gamma = Q \times \mathbb{N}^d$.

The semantics of counter-machines are as expected: A transition $q \xrightarrow{x_i++} q'$ **increments** counter i, a transition $q \xrightarrow{x_i--} q'$ **decrements** it. A **zero-test** transition $q \xrightarrow{x_i=0} q'$ can only be taken if c_i is indeed zero, a non-zero test can only be taken if the corresponding counter is non-zero. A transition $q \xrightarrow{noop} p$ performs **no operation**.

More formally, there is a transition

$$\big((q, \vec{c}), (p, \vec{d})\big) \in R$$

if and only if

- there is a rule $q \xrightarrow{x_i++} p$, and $d_i = c_i + 1$ and $d_j = c_j$ for all $j \neq i$, or
- there is a rule $q \xrightarrow{x_i--} p$, and $d_i = c_i - 1$ and $d_j = c_j$ for all $j \neq i$, or

- there is a rule $q \xrightarrow{x_i=0} p$, and $c_i = 0$, and $\vec{c} = \vec{d}$, or
- there is a rule $q \xrightarrow{x_i \neq 0} p$, and $c_i > 0$, and $\vec{c} = \vec{d}$, or
- there is a rule $q \xrightarrow{noop} p$, and $\vec{c} = \vec{d}$.

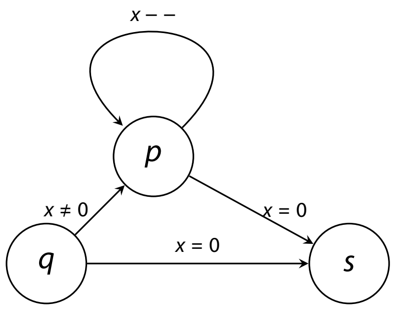
Note that *noop*, zero and non-zero tests do not change the counter assignment, and that increments and decrements only influence the value of one counter. Furthermore, we only consider non-negative counter values. In a configuration $(q, \vec{c})$ with $c_i = 0$, a transition $q \xrightarrow{x_i--} p$ is **not enabled**.

10.4 Example

a) Consider the 1-counter machine $A = (Q, \rightarrow)$ with counter x and

$$Q = \{q, p, s\} ,$$
$$\rightarrow = \left\{q \xrightarrow{x=0} s, q \xrightarrow{x\neq 0} p, p \xrightarrow{x--} p, p \xrightarrow{x=0} s\right\} .$$

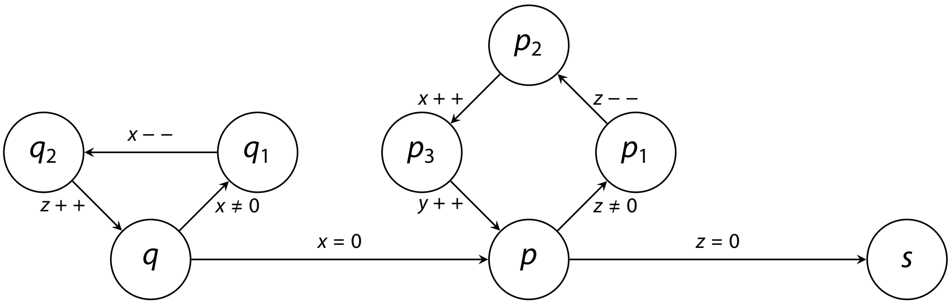
A can be represented graphically as follows:



From a configuration $(q, n) \in Q \times \mathbb{N}$, there is a unique transition sequence ending in $(s, 0)$: If $n = 0$, the transition sequence is just $(q, 0) \rightarrow (s, 0)$. Else, it is

$$(q, n) \rightarrow (p, n) \rightarrow (p, n-1) \rightarrow (p, n-2) \rightarrow \ldots \rightarrow (p, 1) \rightarrow (p, 0) \rightarrow (s, 0) .$$

b) Consider the 3-counter machine $A = (Q, \rightarrow)$ with $Q = \{q, q_1, q_2, p, p_1, p_2, p_3, s\}$ and counters x, y, z, where $\rightarrow$ is given by the following graphical representation.



From a configuration $(q, n, m, 0) \in Q \times \mathbb{N}^3$ (i.e. $c_x = n, c_y = m, c_z = 0$), there is a unique transition sequence reaching control state s. This sequence reaches the configuration $(s, n, m+n, 0)$, i.e. the value of counter x has been added to counter y. Counter z is only used as an intermediary storage.

In the following, when defining counter machines, instead of explicitly stating Q and $\rightarrow$, we will only give the graphical representation. We will not give explicit names to control states for which the names are not important.

Games on configuration graphs

We can now consider games that are played on the transition graphs of automata.

10.5 Definition

Assume that $A = (Q, \rightarrow)$ is some automaton (with memory M) and $Q = Q_\Box \uplus Q_\bigcirc$ is a partitioning of the control states into the control states owned by the universal player $\Box$ and the ones owned by the existential player $\bigcirc$.

Then we obtain an infinite game arena $G_A = (V, R)$ as follows. The configurations and moves are given by the transition system induced by the automaton:

$$V = Q \times M\,,$$

$$R = \left\{ ((q,m),(q',m')) \in V \times V \;\middle|\; \begin{array}{l} ((q,m),(q',m')) \in R, \\ \text{i.e. } ((q,m),(q',m')) \text{ satisfies a rule from } \rightarrow \end{array} \right\}.$$

The ownership is induced by the ownership on the control states:

$$V_\Box = Q_\Box \times M, \quad V_\bigcirc = Q_\bigcirc \times M\,.$$

It is now possible to equip G_A with various winning conditions. Here, we restrict ourselves to a very simple setting.

10.6 Definition

Let A be an automaton and q_f one of its control states. The **control state reachability game** is the reachability game on G_A with respect to the winning set $\{q_f\} \times M$. In words, the goal of the existential player is to reach a configuration (q, m) with $q = q_f$.

10.7 Remark

Analogously, one can define reachability games where the winning condition is a set of control states. One can also define Parity games on G_A. For this, one assumes that a priority assignment $\Omega \colon Q \to \mathbb{N}$ on the control states is given. The priority of a configuration (q, m) is then induced by the priority of q.

The decision problem that we are interested in is as usual to check which player is winning. Here, we fix the initial configuration of interest.

Deciding the winner of a control state reachability game

Given: Automaton $A = (Q, \to)$, state $q_f \in Q$, initial configuration $(q_0, m_0) \in Q \times M$

Question: Does $\bigcirc$ have a winning strategy for the control state reachability game with respect to A and q_f from the initial position (q_0, m_0)?

10.8 Example

a) Control state reachability games on finite-state systems are just a special case of reachability games on finite graphs, where the winning set is a singleton. We have seen how such games can be solved in Section 4.

b) Control state reachability games on pushdown systems are decidable. We will discuss the decision procedure in Section 12.

c) Control state reachability games on Turing machines are undecidable. Recall that the **halting on the empty word** problem is undecidable: Given a Turing machine $A = (Q, \ldots)$ with initial state q_0 and **halting state** q_f, it is not decidable whether there is a transition sequence from q_0 with the empty tape to configuration with control state q_f. Now observe that if let the existential player own all control states, $Q_\bigcirc = Q, Q_\square = \emptyset$, the control state reachability game with respect to A, q_f and q_0 + empty tape is equivalent to the halting on the empty word problem.

In Part c) of the example, we have seen that if a *verification problem* (that has no non-determinism or only one type of non-determinism) is undecidable for some type of automaton, then the associated *game problem* (with two types of non-determinism) is also undecidable.

Games on counter machines

In the rest of this section, we want to look at control state reachability games where the automaton is a d-dimensional counter machine. We have to distinguish two cases: $d = 1$, i.e. one-counter machines (also called **one-counter automata**), and $d > 1$.

10.9 Lemma
One-counter automata are a special case of pushdown automata.

Proof: Exercise 10.21. ■

Together with the decidability of pushdown games that we will prove in Section 12, we obtain that control state reachability games for one-counter automata are decidable.

Surprisingly, adding a second counter makes the problem undecidable. In fact, we can prove that 2-counter machines are Turing-powerful.

10.10 Theorem: see e.g. Minsky 1967 [Min67]
For $d \geqslant 2$, counter machines of dimension d are **Turing powerful**: Given a Turing machine A_{TM} with two designated states q_0, q_f, we can construct in polynomial time a two-counter machine A_{2CM} with two designated states q_0, q_f such that A_{TM} can reach q_f from q_0 with the empty tape if and only if A_{2CM} can reach q_f from q_0 with both counters zero.

In other words: For any Turing machine, we can construct a two-counter machine that simulates it.

10.11 Remark
The backwards direction also works: For any counter-machine (with arbitrary dimension), we can construct a Turing machine that simulates it by storing the counter values on the tape. We say that two-counter machines are **Turing complete**.

As a consequence, any problem that is undecidable for Turing machines is also undecidable for two-counter machines. This in particular applies to the following variant of the **halting problem**.

Control state reachability for two-counter machines

Given: Two-counter machine A, control states q_0, q_f

Question: Is there a transition sequence from $(q_0, 0, 0)$ to (q_f, n, m) for some $n, m \in \mathbb{N}$?

10.12 Corollary
Control state reachability for two-counter machines is undecidable.

Using this result, we immediately obtain the undecidability of control state reachability games on counter machines of dimension $d \geqslant 2$, similar to Part c) of Example 10.8.

10.13 Corollary
Control state reachability games on counter machines of dimension $d > 1$ are undecidable.

It remains to prove Theorem 10.10. We proceed in two steps.

1. We show how to simulate a Turing machine by a counter machine of dimension 3.
2. We show how to simulate a counter machine of arbitrary fixed dimension d by a two-counter machine.

From Turing machines to three-counter machines

10.14 Proposition
For $d \geqslant 3$, counter machines of dimension d are Turing powerful.

Proof:
Let $A_{\mathrm{TM}} = (Q, \delta, \ldots)$ be a Turing machine. It is well known that it is sufficient to consider **deterministic** Turing machines with tape alphabet $\{0, 1\}$. Any other Turing machine can be transformed into such a machine in polynomial time.

We show how to construct a three-counter machine $A_{\mathrm{CM}} = (Q', \rightarrow)$ such that

- $Q \subseteq Q'$: The control states of A_{CM} are the control states of A_{TM} plus some constant number of helper states.
- The counters x, y are used to simulate the tape content of A_{TM}. The third counter h is a helper used as intermediary storage.

Recall that the transition function of a Turing machine consists of mappings of the the shape

$$\delta(q, a) = (p, b, d)\,.$$

If the machine is in state $q \in Q$, and the symbol at the current head position in the tape is $a \in \{0, 1\}$, the machine will

1. replace a by $b \in \{0, 1\}$,
2. move the head to the left or right, depending on $d \in \{L, R\}$, and
3. change the control state to p.

Consider some configuration $v\,qa\,w$, i.e. $v \in \{0, 1\}^*$ is the tape content to the left of the head, $a \in \{0, 1\}$ is the tape content at the head position and $w \in \{0, 1\}^*$ is the remainder to the right of the head. For example, consider

$$\underbrace{001100}_{v}\ q\,\underbrace{1}_{a}\ \underbrace{010}_{w}\ .$$

We may see the tape content as given by the numbers $c_x, c_y \in \mathbb{N}$ such that the binary representation of c_x is v and the binary representation of c_y is reverse($a.w$). In our example, we have

$$c_x = 001100_2, \qquad c_y = \text{reverse}(1.010) = 0101_2\ .$$

We see that the bits that are closest to the head position are the least significant bits in each of the numbers. Here, we follow the convention that the tape content at the head position is a part of c_y, i.e. a is the least significant bit of c_y.

Remark: We have $0101_2 = 101_2$, i.e. if we see the tape content as a number, we cannot detect leading zeros anymore. This corresponds to ignoring leading as well as trailing zeros on the tape of the Turing machine. Initially, we assume that the Turing machine starts with a tape that is empty in the sense that it is filled with infinitely many zeros. This corresponds to the counter values being 0.

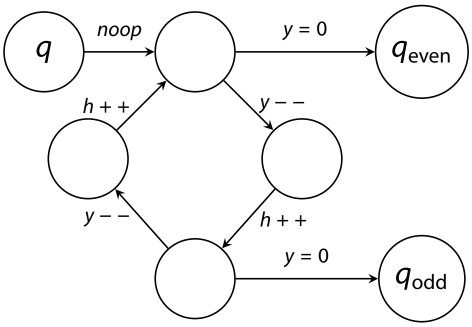
Our goal is to construct the three-counter machine A_{CM} such that counter x stores c_x and counter y stores c_y. Initially, we let both counters be 0, which corresponds to the tape being empty. We now explain how to simulate the Turing machine step by step.

Assume we are in configuration $(q, c_x, c_y, 0)$, which represents the configuration $\text{bin}(c_x)\ q\ \text{reverse}(\text{bin}(c_y))$. We explain how to simulate a transition of the shape

$$\delta(q, a) = (p, b, d)\ .$$

- The first step is to check whether the transition is applicable, i.e. if the symbol at the head position is actually equal to a. To this end, we need to check whether the least significant bit of (the binary representation of) c_y is equal to a. Note that this bit is 1 if and only if c_y is odd.

To test this, we use the following gadget.



From configuration $(q, c_x, c_y, 0)$, we reach configuration $(q_{\text{even}}, c_x, 0, c_y)$ if and only if c_y is even. If c_y is odd, we reach $(q_{\text{odd}}, c_x, 0, c_y)$. Note that we have moved the value c_y from counter y to the helper h.

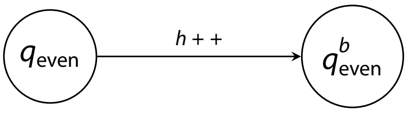
In the states q_{even} and q_{odd}, it is clear which transition of the Turing machine A_{TM} has to be applied: $\delta(q, 0)$ in q_{even} and $\delta(q, 1)$ in q_{odd}.

We restrict ourselves to the case of $\delta(q, 0)$ in q_{even} here, the case of q_{odd} is similar.

It remains to actually apply the transition, i.e. by (i) replacing the content at the head position, (ii) moving the head and (iii) changing the control state.

- For (i), observe that if $a = b$, nothing has to be done. If $a = 0, b = 1$, we have to add a single transition that increments the value. Analogously, if $a = 1, b = 0$, we have to subtract one.

  Let us consider the case $a = 0, b = 1$ here. We obtain the following transition.



  Note that we increment counter h here, because it currently stores the value c_y.

- Next, we need to move the head position. We only discuss the case $d = R$, i.e. the head should be moved to the right. The case $d = L$ is slightly more involved and remains as an exercise for the reader.

  If the current configuration of the Turing machine is

$$v \; qb \; \underbrace{c.w'}_{w},$$

and we move the head to the right, we obtain the new configuration

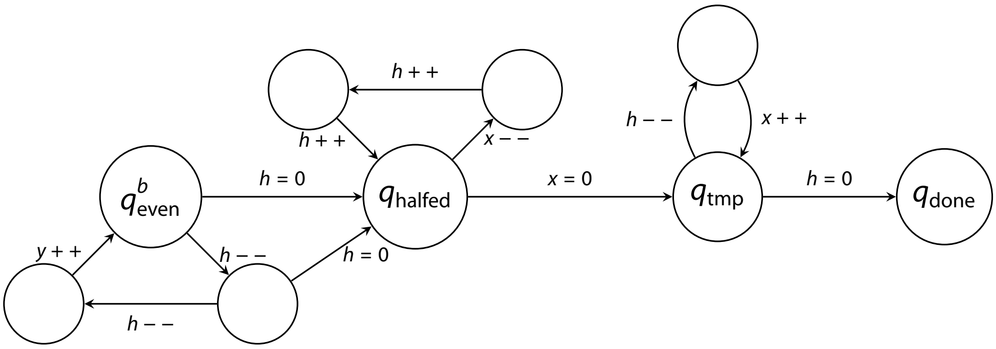
$$v.b \ q \underbrace{c\ w'}_{w} \ .$$

We need to imitate this operation on the numbers representing v and w. Observe that

$$v.b = 2 \cdot v + b \ , \qquad w = \left\lfloor \frac{b.w}{2} \right\rfloor .$$

Shifting a number one bit to the left means multiplying it by two: The bit that was least significant now becomes second-to-least significant. Similarly, shifting a number to the right means dividing it by two (and dropping the remainder).

We design a gadget that performs these operations.



Recall that we are in configuration $(q^b_{\text{even}}, c_x, 0, c_y + b)$ when we enter the gadget. When we reach control state q_{halfed}, we are in configuration

$$(q_{\text{halfed}}, c_x, \left\lfloor \frac{c_y + b}{2} \right\rfloor, 0) \ .$$

The loop from q^b_{even} to q^b_{even} implements the division by decrementing h twice per increment of y. The *shortcut* to q_{halfed} is used to handle the case that $c_y + b$ is odd.

From this configuration, we can in turn reach the configuration

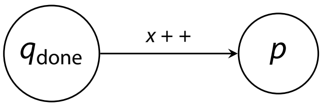
$$(q_{\text{tmp}}, 0, \left\lfloor \frac{c_y + b}{2} \right\rfloor, 2 \cdot c_x) \ .$$

To this end, we take the loop from q_{halfed} to q_{halfed} that decrements counter x while storing the doubled value in counter h. It remains to move the counter value to counter x again, which is implemented by the loop in state q_{tmp}. Finally, we reach

$$(q_{\text{done}}, 2 \cdot c_x, \left\lfloor \frac{c_y + b}{2} \right\rfloor, 0) \ .$$

- It remains to add the former tape content at the head position b to counter x, and to change the control state to p.

  Recall that we have assumed that $b = 1$. The final part of the translation is the following gadget.

- After going through all gadgets, we are in configuration $(p, c'_x, c'_y, 0)$, where c'_x and c'_y represent the new tape content of the Turing machine A_{TM}. The simulation of the next transition can begin.

It is tedious, but conceptually easy to check that any transition sequence of the Turing machine from q_0 + empty tape to q_f induces a transition sequence from $(q_0, 0, 0)$ to $(q_f, n, m, 0)$ in the three-counter machine and vice versa.

Note that we have replaced each transition of the Turing machine by a constant number of transitions and states of the counter machines. Thus, the size of A_{CM} is linear in the size of A_{TM}. ■

From three to two counters

To complete the proof of Theorem 10.10, it remains to show that the three-counter machine A_{CM} that we have constructed in the proof of Proposition 10.14 can be simulated by a two-counter machine. In fact, we show a stronger statement.

10.15 Proposition
For any counter machine of dimension d, we can construct a two-counter machine that simulates it.

The proof of the proposition uses a famous trick due to Minsky [Min67]. Let $\vec{c} \in \mathbb{N}^d$ be a d-dimensional vector of counter values. Let $p_1, p_2, \ldots, p_n$ be the first d prime numbers.

We define the **prime encoding** primenc($\vec{c}$) of $\vec{c}$ to be the number

$$\text{primenc}(\vec{c}) = {p_1}^{c_1} \cdot {p_2}^{c_2} \cdots {p_d}^{c_d} \in \mathbb{N}\,.$$

10.16 Example
Consider $c_x = 10, c_y = 5, c_z = 0$. We have

$$\text{primenc}(\vec{c}) = 2^{10} \cdot 3^5 \cdot 5^0 = 1024 \cdot 243 \cdot 1 = 248832\,.$$

Instead of storing $\vec{c}$, it will be sufficient to store the single number primenc($\vec{c}$). For this to be valid, it is crucial that the value of $\vec{c}$ can be recovered from primenc($\vec{c}$).

10.17 Lemma
The prime encoding is unique: If primenc($\vec{c}$) = primenc($\vec{e}$), then $\vec{c} = \vec{e}$.

Proof:
Assume that primenc($\vec{c}$) = primenc($\vec{e}$). Consequently, we have

$$p_1^{c_1} \cdot p_2^{c_2} \cdots p_d^{c_d} = p_1^{e_1} \cdot p_2^{e_2} \cdots p_d^{e_d}\,.$$

Note that both expressions are prime decompositions of primenc($\vec{c}$). By the **fundamental theorem of arithmetic**, the prime factorization of a number is unique. Thus, we have $c_1 = e_1, \dots, c_d = e_d$ and $\vec{c} = \vec{e}$. ■

It remains to implement the required operations in the form of a two counter machine.

Proof of Proposition 10.14:
Assume that we are given a counter machine A_{CM} of some fixed dimension d. We construct a two-counter machine A_{2CM} that uses two counters. The first counter v will be used to store the prime encoding. The second counter h is used as a helper for intermediary storage.

More formally, to the configuration $(q, \vec{c})$ of A_{CM}, we associate the configuration $(q, \text{primenc}(\vec{c}), 0)$ of A_{2CM}. The two-counter machine will use additional control states and reach intermediary configurations in which the helper is non-zero.

We consider $(q_0, 1, 0)$ as the initial configuration for A_{2CM}, as primenc$(0, 0, \dots, 0) = 1$.

It remains to explain how each type of transition of A_{CM} can be simulated by A_{2CM}. Here, we will consider operations for the second counter y of A_{CM}, which corresponds to the prime number 3. The simulation of the operations for the other counters is similar. However, the later counters will need more control states, as the associated prime number becomes bigger. Here, it is important that d is arbitrary but fixed.

- To simulate a transition of the type $q \xrightarrow{noop} p$, we add a transition $q \xrightarrow{noop} p$ to A_{2CM}.

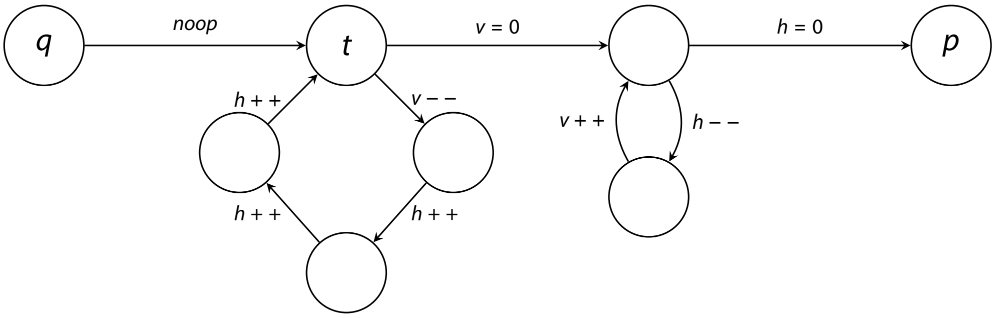
- To simulate a transition of the type $q \xrightarrow{y{+}{+}} p$, observe that this rule induces transitions of the shape

  $$(q, c_x, c_y, \ldots) \to (p, c_x, c_y + 1, \ldots)\,,$$

  and that

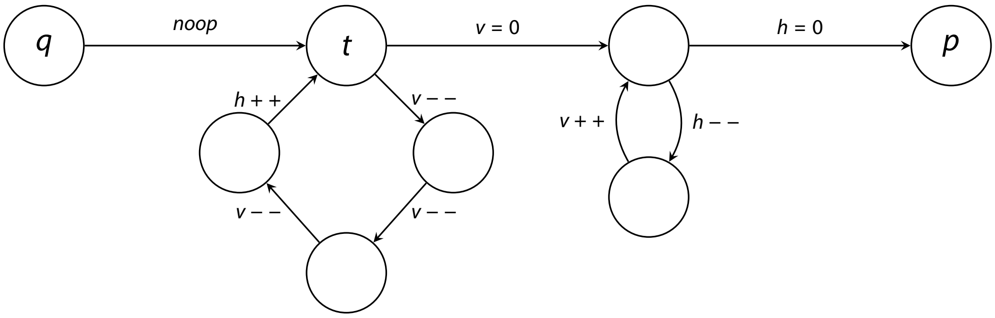
  $$\mathrm{primenc}\big(c_x, c_y + 1, \ldots\big) = 2^{c_x} \cdot 3^{c_y+1} \cdots = 3 \cdot 2^{c_x} \cdot 3^{c_y} \cdots = 3 \cdot \mathrm{primenc}\big(c_x, c_y, \ldots\big)\,.$$

  Consequently, we simulate an increment of c_y by tripling the value of v. This is implemented by the following gadget.



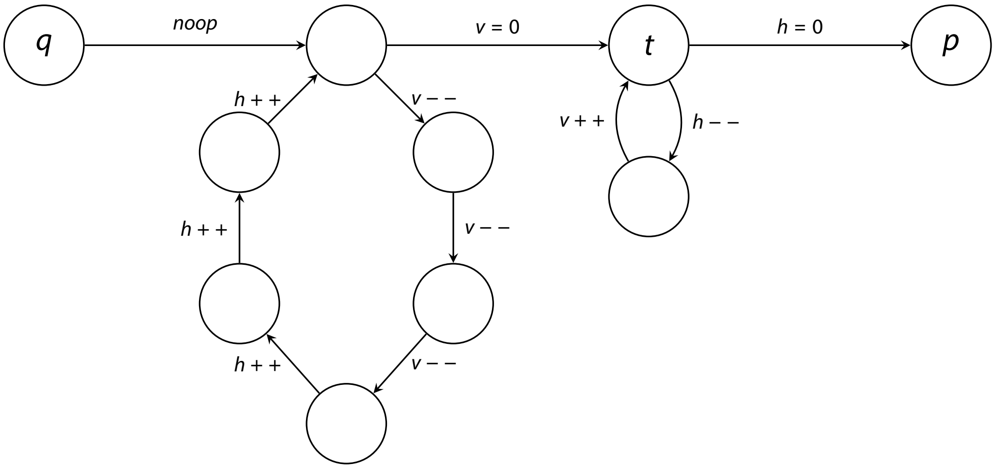
  The gadget takes configuration $(q, c_v, 0)$ first to configuration $(t, 0, 3{\cdot}c_v)$, and then to configuration $(p, 3 \cdot c_v, 0)$ as desired.

- Transitions of type $q \xrightarrow{y{-}{-}} p$ can be implemented very similarly to the case $y + +$. Here, we have to divide the current value of v by 3.



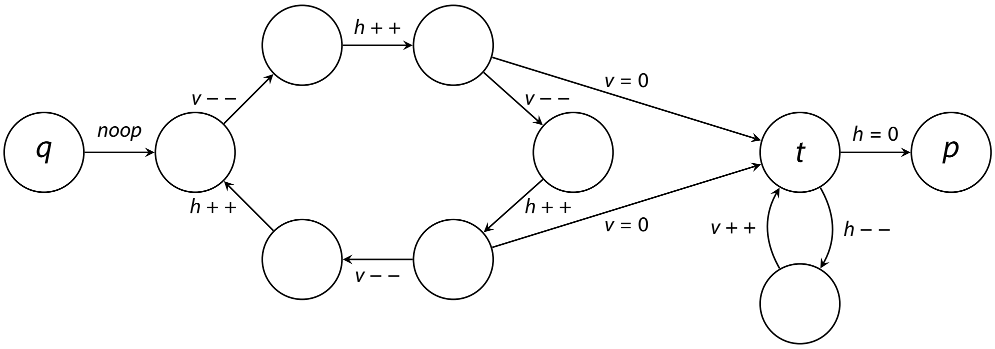
  Note that it may occur that a run of the machine gets stuck in the loop that decrements v. This will happen if and only if the initial value c_v is not divisible by 3, which in turn corresponds to the value of y being 0 in the vector encoded by v. If y is 0, the transition $y - -$ is not enabled, so the fact that our gadget **blocks** is not a problem.

- As mentioned above, testing y for being non-zero corresponds to testing whether c_v is divisible by 3. Transitions of type $q \xrightarrow{y \neq 0} p$ are implemented by the following gadget.



From configuration $(q, c_v, 0)$, we can reach state t if and only if c_v is divisible by 3. In this case, we reach configuration $(t, 0, c_v)$. Finally, we arrive in configuration $(p, c_v, 0)$ as desired.

- To simulate $q \xrightarrow{y=0} p$, we use a gadget similar to the previous one.



Here, we can only reach t if the initial value of v is either 1 or 2 modulo 3, i.e. if it is not divisible by 3. In this case, we restore the original counter value and proceed to state p.

Again, each transition of A_{CM} has been replaced by a constant number of transitions of A_{2CM}. The size of A_{2CM} is linear in the size of A_{CM}, assuming that the dimension d is fixed.

■

10.18 Remark

As we have seen, control state reachability games over counter machines are not decidable. However they are still determined: These are reachability games over a countable graph with finite outdegree, as the number of outgoing transitions in a configuration is bounded by the finitely many rules in →. Thus, these games satisfy the assumptions that we made in Section 4, and (uniform positional) determinacy applies.

A similar argumentation holds for games over Turing machines.

Exercises

10.19 Exercise: Counter machines

Show how to construct a counter machine of dimension $d \geqslant 2$ with two control states q_0, q_f such that there is a transition sequence from $(q_0, n, m, \ldots)$ that reaches q_f if and only if

a) $n \geqslant m$,

b) $n < m$,

c) n is divisible by m .

Hint: You may use an arbitrary constant number of additional counters.

10.20 Exercise: Primality testing

Show how to construct a counter machine of dimension $d \geqslant 1$ with two states q_0, q_f such that **there is** a transition sequence from $(q_0, n, \ldots)$ that reaches state q_f if and only if n is **not** a prime number. Explain your construction.

Hints: You may use an arbitrary constant number of additional counters. You can use non-determinism. You may split your construction into smaller parts (*gadgets*) and explain later how these should be combined.

10.21 Exercise: One-counter automata as pushdowns

Prove that one-counter automata can be simulated by pushdown systems.

Recall that a pushdown system is an automaton $(Q, \to)$ with memory S^*, where S is some finite **stack alphabet**. The transition rules in $\to$ are of the shape

$$q \xrightarrow{\text{push } a} p \quad \text{or} \quad q \xrightarrow{\text{push } a} p$$

for symbols $a \in S$. There is a transition $((q, m) \to (p, m')) \in T$ if

- there is a rule $q \xrightarrow{\text{push } a} p$ and $m' = m.a$, or
- there is a rule $q \xrightarrow{\text{pop } a} p$ and $m = m'.a$.

(Here, we use the convention that the right end of the word m encodes the top of stack.) Note that a pop a transition is only enabled when a is indeed the top of stack.

Assume that some one-counter automaton $A_{\text{OCA}} = (Q', \to')$ with states q_0, q_f is given. Show how to construct a pushdown system $A_{\text{PDS}} = (Q, \to)$ with $Q' \subseteq Q$ over a suitable stack alphabet such that q_f is reachable in A_{OCA} from $(q_0, 0)$ if and only if q_f is reachable in A_{PDS} from some suitable initial configuration. Briefly argue that your construction is correct.

10.22 Exercise: Integer counter machines

An **integer counter machine** of dimension d is defined similarly to a counter machine of dimension d. However, the counters can reach negative values, i.e. the memory is $\mathbb{Z}^d$. A transition of type $q \xrightarrow{x_i--} p$ is enabled even if the value of counter x_i is zero.

a) Let A_{ICM} be an integer counter machine of dimension d, and q_0, q_f control states. Show how to construct a counter machine A_{CM} with states q_0' and q_f' such that q_f is reachable from $(q_0, 0, \ldots, 0)$ in A_{ICM} if and only if q_f' is reachable from $(q_0', 0, \ldots, 0)$ in A_{CM}.

b) Let A_{CM} be a counter machine of dimension d, and q_0', q_f' control states. Show how to construct an integer counter machine A_{ICM} with states q_0 and q_f such that q_f is reachable from $(q_0, 0, \ldots, 0)$ in A_{ICM} if and only if q_f' is reachable from $(q_0', 0, \ldots, 0)$ in A_{CM}.

In both cases, argue briefly that your construction is correct.

11. Undecidable games over counter nets

In the previous section, we have seen that control state reachability games on counter machines are undecidable. In the proof, we have not even used the game aspect; we have relied on the fact that already verification problems for counter machines are undecidable. In this section, we want to weaken the computational model to so-called **counter nets**. In contrast to counter machines, almost all verification problems for counter nets are decidable. However, games on these automata remain undecidable.

Sources
The content of this section is common knowledge in the theory of perfect information games and does not follow any particular source.

Counter nets

We start by introducing counter nets, restricted counter machines that cannot perform zero tests.

11.1 Definition
A **counter net** of dimension $d \in \mathbb{N}$, is defined similar to a counter machine of dimension d, see Definition 10.3. However, the set of operations that is allowed in the definition of the transition rules $\rightarrow \subseteq Q \times NOps \times Q$ is restricted. Let $\mathcal{X} = \{x_1, \ldots, x_d\}$ be the set of counters. We have

$$NOps = \{noop\} \cup \bigcup_{x \in \mathcal{X}} \{x++, x--\} .$$

We call the counters of a counter net **partially blind**. They are *partially* **blind**, because we cannot test them for being zero. However, they are only **partially** *blind*, because we are still able to assert that a counter has a positive value: After a transition of type $q \xrightarrow{x--} p$ has been taken, we know that the previous value of counter x was non-zero; otherwise the transition would not have been enabled.

11.2 Remark
The reader familiar with automata theory might see that counter nets are just a variant of **vector addition systems with states (VASS)** (or, equivalently, Petri nets). To be precise, counter nets are VASS in which the transition multiplicities are encoded in unary. In a VASS, we allow transitions of type $q \xrightarrow{x+m} p$ for arbitrary constants m. This in particular allows having a transition $q \xrightarrow{x+2^n} p$ that adds an exponential value to the counter, but it can be encoded in binary using $\log 2^n = n$ bits. In our definition of counter nets,

such a transition would need to be decomposed into 2^n many increments, which will need at least 2^n bits.

Thus, the binary representation that one usually considers for VASS is more succinct.

Counter nets have decidable verification problems: Intuitively, the computational power of counter machines relies on having zero tests. Removing them limits their capabilities, but makes many problems decidable.

11.3 Theorem
The control state reachability problem for counter nets is:

- in EXPSPACE for arbitrary dimension, i.e. it can be solved using exponential space and doubly exponential time,
- EXPSPACE-hard for arbitrary dimension, i.e. it cannot be solved using less than exponential space, and, unless EXPSPACE = EXP, not in exponential time or less,
- NL-complete for dimension 2.

The result follows from the corresponding results for VASS resp. Petri nets, namely

- an EXPSPACE algorithm for coverability (Rackoff 1978 [Rac78]),
- the EXPSPACE-hardness of coverability and reachability (Lipton 1976 [Lip76])
- the NL-completeness of coverability in the case of two-dimensional unary VASS. The NL-hardness is by the NL-hardness of the PATH problem for directed graphs. The membership in NL is implies by the membership of reachability for two-dimensional unary VASS, proven in [ELT16].

The last result is explicitly for VASS encoded in unary. Rackoff's result talks about VASS encoded in binary, but trivially also works for VASS encoded in unary. Lipton's hardness proof works for both VASS encoded in binary or unary, as his construction never makes use of transitions that decrement or increment a counter by more than one.

11.4 Remark
The proofs and more information on Rackoff's and Lipton's result can be found in our lecture notes on **concurrency theory**, available at `https://tcs.cs.tu-bs.de/documents/lecturenotes/conctheo2017.pdf`.

Games on counter nets

Surprisingly, games over two-counter nets are undecidable. This is in sharp contrast with the decidability of control state reachability.

11.5 Theorem
The problem of deciding the winner of a control state reachability game on a counter net of dimension $d > 1$ is undecidable.

In the proof of the theorem, we will reduce the control state reachability problem for two-counter machines, which is undecidable by Corollary 10.12. Using the game aspect, we can simulate zero tests.

Proof:
Let $A_{\text{2CM}} = (Q, \rightarrow)$ be a given two-counter machine with counters x and y. Let q_0, q_f be a designated initial and halting state, respectively.

We show how to construct a two-counter net $A_{\text{Net}} = (Q_\Box \cup Q_\bigcirc, \rightarrow')$ such that there is a transition sequence from $(q_0, 0, 0)$ to (q_f, n, m) for A_{2CM} if and only if the existential player has a winning strategy for the control state reachability game with respect to A_{Net}, $(q_0, 0, 0)$ and q_f. As the control state reachability problem for two-counter machines is undecidable, Corollary 10.12, the desired result follows.

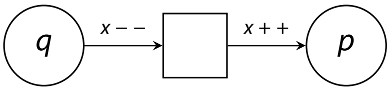
We will construct A_{Net} as follows:

$$Q_\bigcirc = Q\,,$$

i.e. the states owned by the existential player are precisely the states of the two-counter machine A_{2CM}. The states owned by $Q_\Box$ consist of helper states, at most one per transition plus an additional deadlock state *dead*.

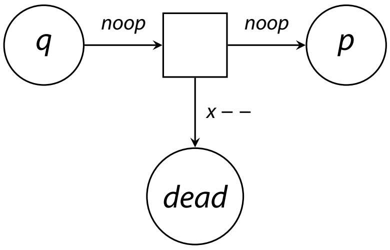
Each transition of A_{2CM} will be replaced by a constant number of transitions in A_{Net}. We will demonstrate how this translations works for transitions involving the first counter x. The transitions for counter y can be implemented similarly.

- Transitions of type $q \xrightarrow{noop} p$, $q \xrightarrow{x{+}{+}} p$ and $q \xrightarrow{x{-}{-}} p$ can be inserted into A_{2CM} without any change. Observe that the semantics of two-counter machines and two-counter nets coincides for these transitions.
- A transition of type $q \xrightarrow{x \neq 0} p$ is replaced by the following gadget.



The semantics is as desired: The transition labeled by $x--$ can only be taken if the value of counter x is non-zero. The increment $x++$ restores the former counter value.

- A transition of type $q \xrightarrow{x=0} p$ can be replaced by a gadget. We let the existential player claim that $x = 0$ in a transition that is labeled by *noop*. The resulting state is owned by the universal player □. She can now check whether this claim is true: If yes, she proceeds to state p, where ○ is in control. If the counter is non-zero, she moves to a deadlock.



We see that □ can move to state *dead* if and only if the existential player has lied and the counter was non-zero, because only in this case, the transition $x--$ is enabled. If the counter was zero, as claimed by ○, □ has no choice but to use the transition that leads to p.

We claim that the translation is correct: The existential player has a winning strategy for reaching state q_f if and only if q_f was reachable in the two-counter machine.

For one direction, assume that q_f is reachable in the two-counter machine, and let

$$(q_0, 0, 0) \xrightarrow{op_1} (q_1, c_1, d_1) \xrightarrow{op_2} \dots \xrightarrow{op_k} (q_k, c_k, d_k) = (q_f, n, m)$$

be the associated sequence of transitions. It induces a winning strategy for ○: In configuration (q_i, c_i, d_i), use the transition labeled by op_{i+1} if it is not a (non-)zero test. If op_{i+1} is $x \neq 0$, use the corresponding transition labeled by $x--$. If op_{i+1} is $x = 0$, use the corresponding transition labeled by *noop*. (And similar for counter y.) Since the transition sequence defining the strategy was valid in the counter-machine, the transitions corresponding to zero tests are only taken when the counter value is actually zero. Consequently, □ has no choice but to use the move that leads to the next state; the transition leading to *dead* is not enabled as the counter cannot be decremented. Hence, the strategy induces a unique play of the game that visits all configuration (q_i, c_i, d_i). In particular, it visits state q_f, and the play is winning.

For the other direction, we show that if q_f is not reachable, then ○ cannot have a winning strategy. To this end, we show that □ has a winning strategy. This strategy is very simple: Whenever $\Box$ is in control, she uses the move to the state *dead* if possible. To prove that this strategy is indeed winning for $\Box$, consider an arbitrary play p from $(q_0, 0, 0)$ that is conform to the strategy. Towards a contradiction, assume that the play visits q_f, $p = (q_0, 0) \dots (q_f, n, m)$. Consider the sequence of transitions of the two-counter machine that corresponds to the play. Since q_f is not reachable by assumption, this is not a valid transition sequence. The only reason for it not being valid can be that a zero test transition was used although the corresponding counter was non-zero. This contradicts the definition of the strategy for $\Box$, which would have taken the move to the state *dead* in this case. ■

12. Context-free games

We conclude our study of games on the (infinite) configuration graphs of automata by considering a case in which reachability games can be decided: We study **context-free games**, here formalized using **Pushdown systems**. We will show that control state reachability games on the configuration graphs of pushdown systems are decidable (i.e. the winner can be computed). For the proof, we will study **Walukiewicz's reduction**, which allows us to turn the infinite state pushdown game into a finite state reachability game. This reduction can be seen as the most important contribution in the area of algorithmic game theory for infinite state games.

Sources
The presentation is loosely based on Walukiewicz's paper [Wal01].

Another presentation of the material can be found in [ZKW].

Pushdown games

Recall the definition of Pushdown systems.

12.1 Definition: Pushdown system
Let Δ be a finite **stack alphabet**. A **pushdown system (PDS)** $P = (Q, \rightarrow)$ over Δ is an automaton with memory Δ^*. As usual Q is a finite set of control states, and $\rightarrow$ is a set of transition rules of the form

$$\rightarrow \subseteq Q \times Ops_\Delta \times Q \,,$$

where

$$Ops_\Delta = \{noop\} \cup \{\mathrm{push}_a, \mathrm{pop}_a \mid a \in \Delta\}$$

is the set of **stack operations**.

The configurations are of the shape (q, m), where $m \in \Delta^*$ is the **stack content**. Here, we fix the convention that the rightmost symbol of m encodes the top-of-stack.

The semantics of PDS is as expected. Push operations add a letter to the top of the stack, pop operations remove the top-of-stack. A transition labeled by pop_a is only enabled when $a \in \Delta$ is indeed the top-of-stack.

For the sake of completeness, we give the formal definition.

12.2 Definition: Semantics of pushdown systems

The configuration graph of a PDS P over Δ is $(Q \times \Delta^*, R)$, where

$$((q, m), (p, w)) \in R$$

if

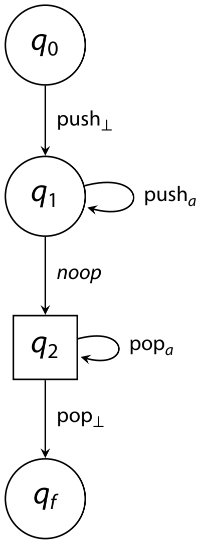
- $w = m$ and there is a transition rule $q \xrightarrow{noop} p$, or
- $w = m.a$ and there is transition rule $q \xrightarrow{\text{push}_a} p$, or
- $m = w.a$ and there is transition rule $q \xrightarrow{\text{pop}_a} p$.

We will now consider the same setting as in the previous section: We assume that an ownership partitioning $Q = Q_\square \uplus Q_\bigcirc$, an initial configuration (q_0, m) and a final state q_f are given, and we ask whether the existential player has a strategy to enforce reaching (q_f, w) (for some w) from (q_0, m).

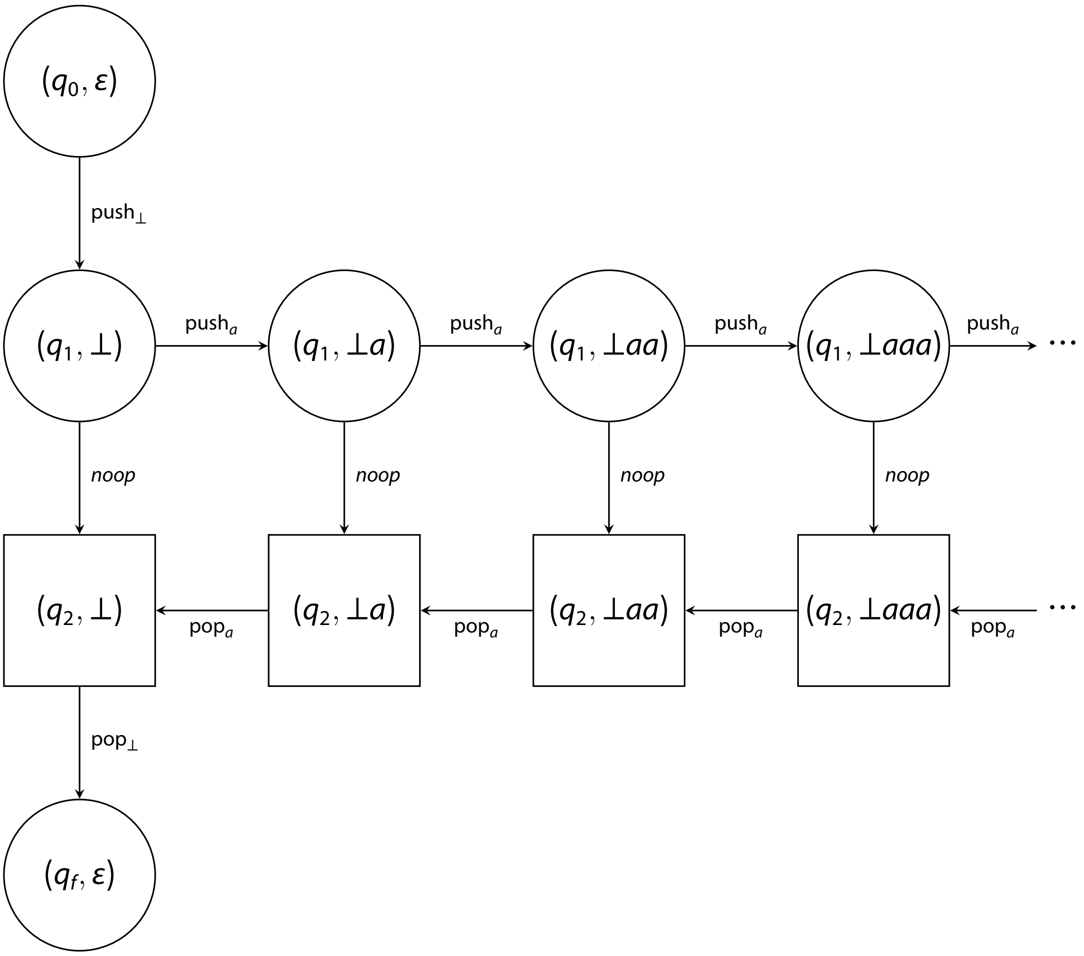
Here, we will always assume that in the initial configuration the stack is empty, i.e. (q_0, ε) is the initial position. Note that in this configuration, no pop transition is enabled.

12.3 Example

We present a PDS game that is a modified version of an example by Zimmermann [ZKW]. Consider the PDS $P = (Q_\square \uplus Q_\bigcirc, \rightarrow)$ over the stack alphabet $\{\bot, a\}$ given by the following graphical representation.



The associated configuration graph is as follows.



Obviously, ○ has a strategy to reach q_f, namely by pushing $\perp$ in q_0 and then directly going to q_2. Even if she decides to use the push$_a$ transition finitely often, she will win because □ has no choice but to move to q_f.

Before proceeding with the theory, let us clarify some notation needed in the rest of the section. Let

$$(q_0, m_0) \xrightarrow{op_1} (q_1, m_1) \xrightarrow{op_2} \dots$$

be a computation of a pushdown system, i.e. a sequence of configurations $(q_i, m_i) \in Q \times \Delta^*$ where each (q_{i+1}, m_{i+1}) results from (q_i, m_i) by applying a transition rule. Assume that (q_{i+1}, m_{i+1}) is obtained from (q_i, m_i) by applying a push-operation, say rule $q_i \xrightarrow{\text{push}_b} q_{i+1}$. In particular, we then have $m_{i+1} = m_i.b$.

$$(q_i, m_i) \xrightarrow{\text{push}_b} (q_{i+1}, m_i.b)\,.$$

Either b stays on the stack for the rest of the computation, or there is a **corresponding pop** in which b is removed from the stack for the first time:

$$\ldots \to \underbrace{(q_i, m_i) \xrightarrow{\text{push}_b} (q_{i+1}, m_i.b)}_{\text{push}} \underbrace{\to \ldots\ldots\ldots \to}_{b \text{ on the stack}} \underbrace{(q_j, m_i.b) \xrightarrow{\text{pop}_b} (q_{j+1}, m_i)}_{\text{corresponding pop}} \to \ldots .$$

During the time that b is on the stack, the prefix m_i of the stack content is not modified. The index $j + 1$ at which the corresponding pop just has occurred can be identified as the smallest index $\ell > i$ such that $m_\ell = m_i$.

The goal of the rest of this section is to prove the following theorem.

12.4 Theorem: Walukiewicz 1996 [Wal01]
Control state reachability games on the configuration graphs of Pushdown systems (*PDS games*) are decidable.

Actually, Walukiewicz has shown that even Parity games are decidable. For the sake of simplicity, we only discuss the case of control state reachability. The extensions of the result are discussed in Remark 12.16.

Walukiewicz's reduction

The proof of the theorem relies on **Walukiewicz's reduction**. From it, we cannot only derive the decision procedure, but it also gives us the strategies that are needed for PDS games.

12.5 Theorem: Walukiewicz's reduction
Given a control state reachability game on a PDS $\mathcal{G}^{PDS}$, we can effectively construct a reachability game $\mathcal{G}^{FS}$ on a finite graph $G = (V_\Box \uplus V_\bigcirc, R)$ with respect to some winning set $B \subseteq V$ and an initial position $x \in V$ such that $\bigcirc$ wins $\mathcal{G}^{PDS}$ if and only if she wins $\mathcal{G}^{FS}$ from x.

The decidability of PDS games follows immediately from the reduction: The finite state game can, once it has been computed, easily be solved using the attractor construction.

12.6 Remark
The size of the underlying graph of $\mathcal{G}^{FS}$ is exponential in the number of states of the PDS. Furthermore, $\mathcal{G}^{FS}$ can be constructed in exponential time. Since reachability games can be solved in linear time, overall we obtain an EXP algorithm for solving PDS games.

One can in fact show that this is optimal: Control state reachability games on PDS are EXP-complete. Walukiewicz's paper [Wal01] contains an indirect proof of this fact. For a proof which is more accessible, see [MSS05].

The construction is quite complicated. We provide a sequence of explanations, each explanation going into more details than the previous, finally culminating in the formal definition

High-level idea: Storing only the top-of-stack

The fundamental idea behind the construction of $\mathcal{G}^{FS}$ is the following: Instead of storing the unbounded stack content, we **only store the top-of-stack**: We will consider positions of shape (q, a), where $q \in Q$ is a state and $a \in \Delta$ is a single symbol. This obviously results in a finite-state game, as $Q \times \Delta$ is a finite set.

In $\mathcal{G}^{FS}$, transitions labeled by *noop* can be executed normally. Push and pop transitions need to be modified: If after using transition rule $q \xrightarrow{\text{push}_b} s$ in position (q, a), we would simply move to (s, b), we would forget a. This becomes a problem when the corresponding pop transition, say $s' \xrightarrow{\text{pop}_b} p$, occurs. As we store the top-of-stack, we can verify that the transition is indeed enabled, but it is not clear what the new top-of-stack should be as we have forgotten the former top-of-stack a.

This problem is solved as follows. Whenever a push should be performed in a play of $\mathcal{G}^{PDS}$, the corresponding play of $\mathcal{G}^{FS}$ splits into two plays:

- Either the push is performed. After the corresponding pop occurs, the play ends.
- Or we skip the subplay of $\mathcal{G}^{PDS}$ between the push that we want to perform and the corresponding pop.

More precisely, consider the position (q, a) of $\mathcal{G}^{FS}$. For a transition $q \xrightarrow{\text{push}_b} s$, there are two possible continuations of the play.

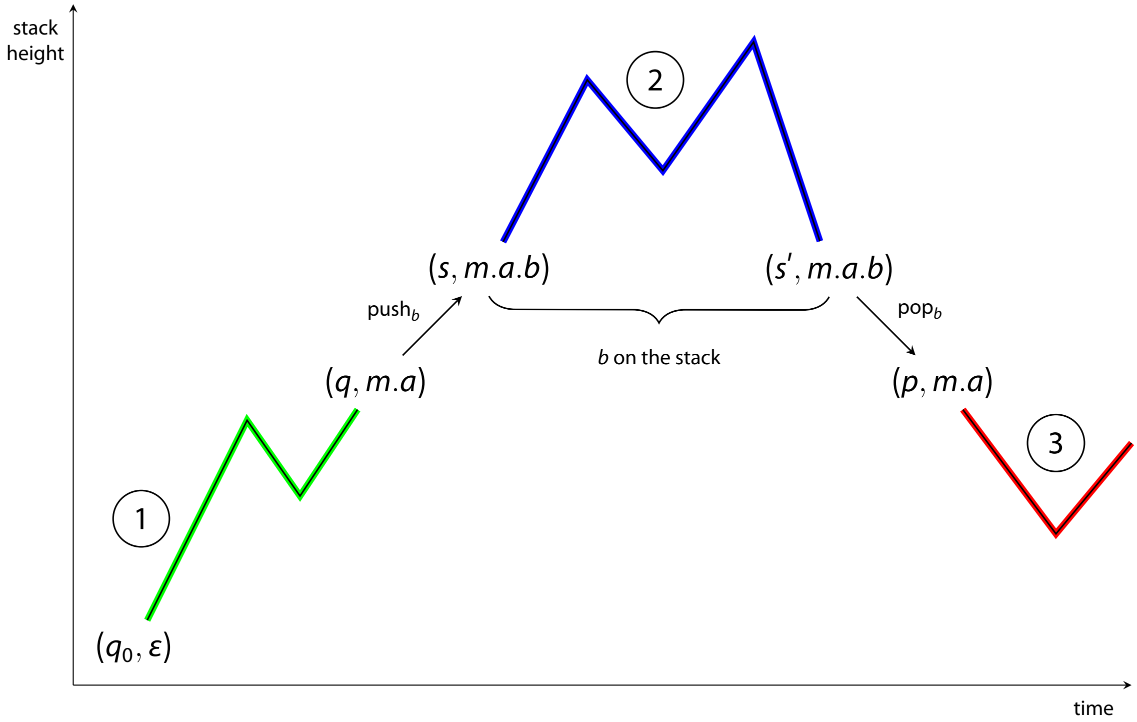
- Either, the **push can be performed**. The play moves to position (s, b). As soon as pop_b occurs, the play will end.
- Or, the play can move to position (p, a) for some state p. We assume that the subplay in which b is on the stack, i.e. the sequence of transitions

  $$q \xrightarrow{\text{push}_b} s \longrightarrow \ldots \longrightarrow s' \xrightarrow{\text{pop}_b} p$$

  has been **skipped**. Note that after pop_b, indeed symbol a should (again) be the top-of-stack.

This idea is depicted in Figure 1.

Play in $\mathcal{G}^{PDS}$:



Corresponding plays in $\mathcal{G}^{FS}$:

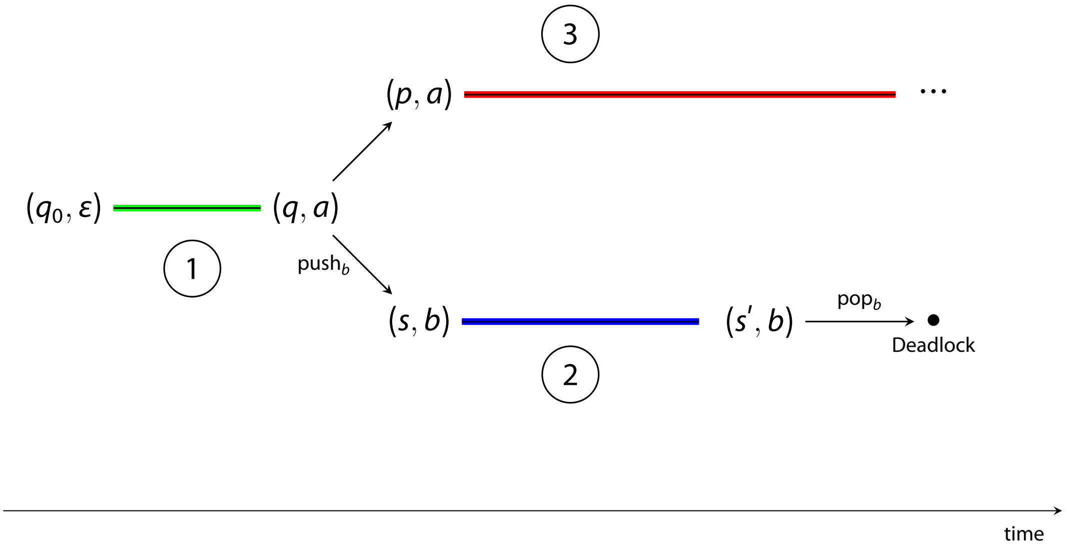


Figure 1: The idea behind the construction of $\mathcal{G}^{FS}$.

This approach clearly solves the above-mentioned problem: Since the play ends after the top-of-stack has been popped, it does not hurt to forget the rest of the stack content.

However, there are a few missing holes that are crucial for the correctness of the reduction:

- Which player wins in case the game ends after a pop?
- Which state(s) p are eligible for jumping to them (instead of performing the push)?

Guess & check

These problems are solved by a **guess and check approach**. The fundamental idea behind guess and check is to guess information non-deterministically, use it, and later check that the guess has been correct. This replaces a deterministic upfront computation of the information.

In our case, the guess and check approach is used whenever a push should be performed. We guess the states p that we can can be reached by the corresponding pop.

In a guess and check algorithm, normally the guessed information is first used, and later verified. Here, we can exploit that we are in a game setting, and have two types nondeterminism – one for each player – at hand. We give the power of making the guess to the existential player. After it has been made, the universal player decides whether to trust and use the guess, or whether it should be verified. In the notation of Figure 1, at the end of ①, the existential player can make a guess, but then we give the universal player the choice between verifying the guess, ②, or trusting it, ③.

Let us clarify what the guessed information is and how it is used. Whenever a push push$_b$ should be performed, the existential player is allowed to make a **prediction** which states p can be reached with the corresponding pop$_b$. Afterwards, it is the universal players choice to decide:

- Whether to **verify the prediction** by performing the push. In this case, the play continues until the corresponding pop$_b$ occurs.
- Whether to **trust the prediction** and jump to one of the states proposed by the existential players. This skips the subplay in which b would be on the stack.

More formally, the prediction picked by the existential player is a set $P \subseteq Q$ of states. The reason why we need a set of states (instead of a single state) is that in the play that unfolds after the push has been performed, both players may influence the outcome. We comment on this in more detail in Remark 12.7.

- Assume the universal player wants to verify the prediction. The push is performed and the play goes on. If the corresponding pop occurs, we know precisely the state p in which the play is. The existential played wins if p is contained in her prediction ($p \in P$), else ($p \notin P$) the universal player wins.
- If the universal player trusts the prediction P, she can pick an arbitrary state $p \in P$ that is contained in the prediction. The play then continues from state p with unchanged top-of-stack.

This in particular fills in the holes in the construction mentioned above.

We explain the details of the construction on an example play.

1. Assume the play is in position (q, a) (i.e. $a \in \Delta$ is the top-of-stack, the rest of the stack is not stored). Furthermore assume that player ☆, the owner of $q \in Q_{☆}$, wants to execute a push transition, say $q \xrightarrow{\text{push}_b} s$.
2. The play moves to the position $\mathit{Push}((q, a), (s, b))$ in which the **intention to execute this push** is signaled. In this position, the existential player ○ is in control (independent of which player owns q). She is allowed to make a **prediction** $P \subseteq Q$. The prediction should be the set of control states that can be reached after pop_b has occurred, the pop corresponding to the push_b that ☆ wants to perform.
3. After the prediction is chosen, the play moves to a state $\mathit{Predict}((q, a), (s, b), P)$ storing the prediction and the push. In this state, the universal player □ is in control. She has two choices:
    - She can **trust the prediction**. In this case, she can pick an arbitrary state $p \in P$. The play continues as if the transition sequence

      $$(q, a) \xrightarrow{\text{push}_b} (s, b) \longrightarrow \ldots \longrightarrow (s', b) \xrightarrow{\text{pop}_b} (p, a)$$

      would have been played: We are in control state $p \in P$ and the top-of-stack is a again (since b was just popped). We say that the part of the play between push and pop has been **skipped**.
    - She can **doubt the prediction** and **verify** it. The push operation is actually performed, and the prediction is stored, i.e. we go to position

      $$(s, b, P) .$$

4. If another push operation is performed, this process repeats. The new prediction replaces the old one, i.e. we store at most one prediction at a time.

5. Consider the case that a pop operation should be performed. Say we are in state (s', b, P), and the owner of s' has selected the transition $s' \xrightarrow{\text{pop}_b} p$. Note that P is the prediction that was made by ○ just before b was pushed. In particular, we are in the case that □ wants to verify precisely this prediction.

   The game moves to a special positions $Pop(p, P)$ storing the target state of the pop and the prediction. This position is a deadlock. It is winning for the existential player if $p \in P$, i.e. p is as predicted, and winning for universal player if $p \notin P$.

12.7 Remark

a) Note that we need a set of states P as prediction instead of just a single state: The play that unfolds after a push has been performed also depends on the behavior of □. A strategy for ○ cannot guarantee that a unique state is reached.

   However, it would be too coarse to just consider the set of all states that are reachable with the desired pop: The existential player ○ can influence the play that happens after the push, so she may be able to avoid some undesirable states.

   We will later see that a strategy $s_{\bigcirc}^{\text{PDS}}$ for ○ on $\mathcal{G}^{PDS}$ induces for each push a (unique) prediction P for $\mathcal{G}^{FS}$. Namely, it defines the set of states reachable by a corresponding pop in plays in which ○ conforms to $s_{\bigcirc}^{\text{PDS}}$, while □ can be pick arbitrary moves. The choice among the states in P corresponds to the choices that □ can make in $\mathcal{G}^{PDS}$ between push and corresponding pop. Intuitively, P contains one state p for each strategy of □ for $\mathcal{G}^{PDS}$.

b) The construction indirectly enforces that the existential player is honest with her prediction, i.e. she has to choose a set of states P such that each state in P is actually reachable, and any state that she cannot prevent being reached is contained in P.

   - In case she picks a set P that is too big (i.e. it contains states that are unreachable), there are two cases: If the additional states are good for ○, then □ will never use these states when skipping subplays. If the additional states are good for □, then □ is free to skip to one of these states (although reaching the state might not have been possible in the original PDS game), and ○ might lose unnecessarily.

   - If she picks a set P that is too small (i.e. it misses out some states that the game might reach), then □ can win by verifying the prediction and reaching one of those missing states.

Hence, a winning strategy for ○ for $\mathcal{G}^{FS}$ will never pick such a prediction.

c) Note that if P is valid prediction, in the sense that the existential player will not lose if □ decides to verify, then so is $P \cup \{q_f\}$. If the universal player skips and jumps to state q_f, she loses instantly.

Formal definition of the construction

It remains to formally state the construction of $\mathcal{G}^{FS}$. For simplicity, all positions of the game will maintain a prediction (unlike in the example play above, where we started with having no prediction). Hence, positions of $\mathcal{G}^{FS}$ are essentially of the shape (q, a, P), where $q \in Q$ is a control state, $a \in \Delta$ is the top-of-stack, and $P \subseteq Q$ is the current prediction. To model the empty stack, we also allow $a = \varepsilon$ and define $\Delta_\varepsilon = \Delta \uplus \{\varepsilon\}$.

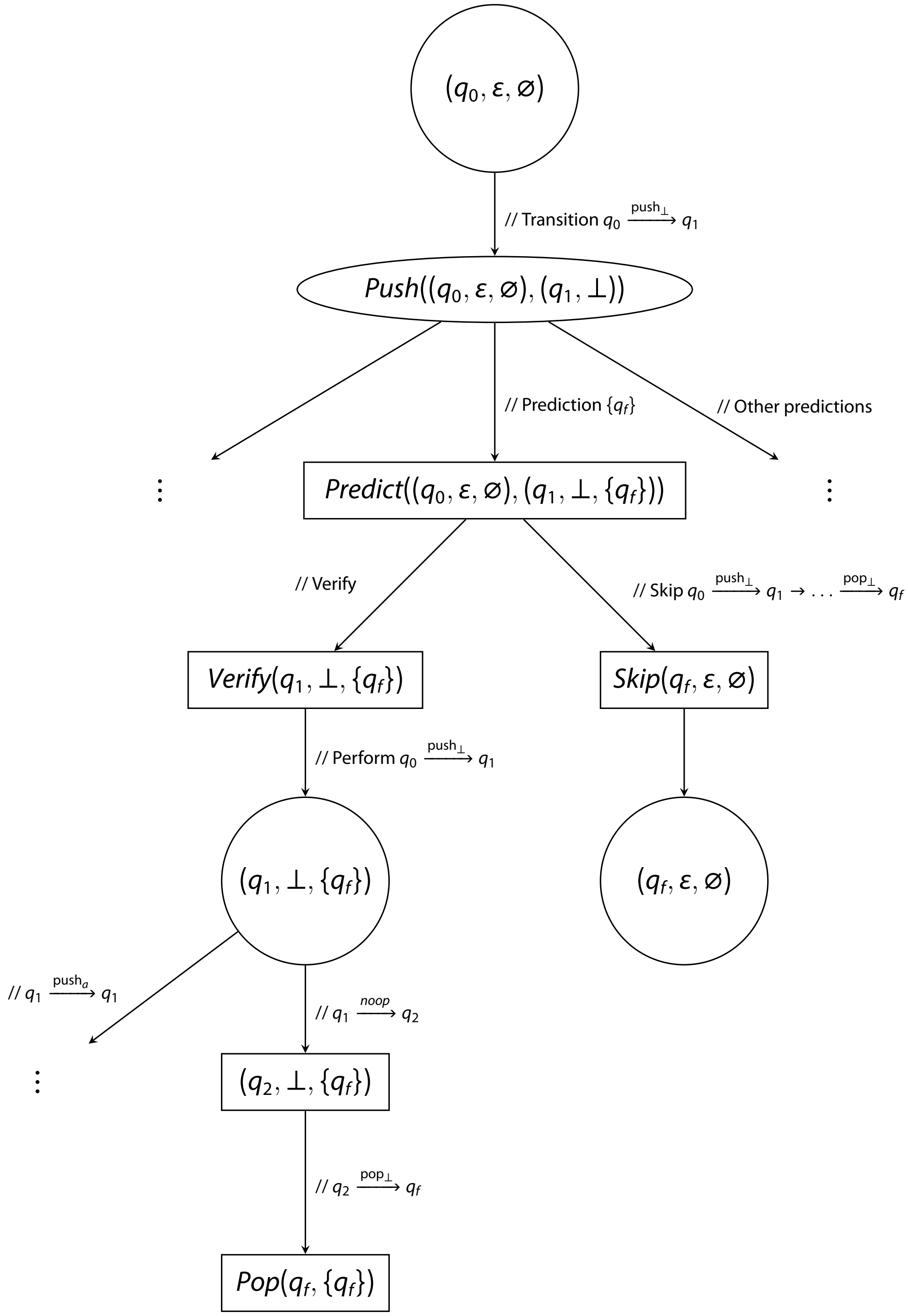
The game will also have intermediary positions of shape $\mathit{Push}((q, a, P), (s, b))$, $\mathit{Predict}((q, a, P), (s, b, P'))$, $\mathit{Verify}(s, b, P')$, $\mathit{Skip}(p, a, P)$, and $\mathit{Pop}(p, P)$ to implement the mechanism described before. The meaning of the states $\mathit{Push}((q, a, P), (s, b))$, $\mathit{Predict}((q, a, P), (s, b, P'))$, and $\mathit{Pop}(p, P)$ has been explained before. The positions of type $\mathit{Predict}((q, a, P), (s, b, P'))$ and $\mathit{Verify}(s, b, P')$ are additional intermediary positions to signal that the universal player has just decided to skip respectively verify a prediction. They are actually not strictly required for the correctness of the construction. However, their presence will greatly simplify the proof of correctness.

The initial configuration is $(q_0, \varepsilon, \varnothing)$, consisting of the initial state, the empty stack and the empty prediction. As ε cannot be popped, starting with the empty prediction does not hurt.

A state is winning if it is of the shape (q_f, a, P), i.e. we have reached the desired control state, or if a pop has occurred that leads to a state that is in the current prediction. Formally, the latter case will correspond to positions of the shape $\mathit{Pop}(p, P)$ with $p \in P$.

Before finally giving the formal definition, we present a part of the game arena of $\mathcal{G}^{fin}$ for the game from Example 12.3. This should be helpful to visualize the construction.

12.8 Example
Consider the PDS game from Example 12.3. The associated finite-state game is as follows.


$(q_0, \varepsilon, \varnothing)$
// Transition $q_0 \xrightarrow{\text{push}_\bot} q_1$
$Push((q_0, \varepsilon, \varnothing), (q_1, \bot))$
// Prediction $\{q_f\}$
// Other predictions
$Predict((q_0, \varepsilon, \varnothing), (q_1, \bot, \{q_f\}))$
// Verify
// Skip $q_0 \xrightarrow{\text{push}_\bot} q_1 \to \dots \xrightarrow{\text{pop}_\bot} q_f$
$Verify(q_1, \bot, \{q_f\})$
$Skip(q_f, \varepsilon, \varnothing)$
// Perform $q_0 \xrightarrow{\text{push}_\bot} q_1$
$(q_1, \bot, \{q_f\})$
$(q_f, \varepsilon, \varnothing)$
// $q_1 \xrightarrow{\text{push}_a} q_1$
// $q_1 \xrightarrow{noop} q_2$
$(q_2, \bot, \{q_f\})$
// $q_2 \xrightarrow{\text{pop}_\bot} q_f$
$Pop(q_f, \{q_f\})$

The states $Pop(q_f, \{q_f\})$ and $(q_f, \varepsilon, \varnothing)$ are deadlocks that are winning for the existential player. Hence, the existential player indeed has a winning strategy for $\mathcal{G}^{FS}$. Similar to the winning strategy for $\mathcal{G}^{PDS}$ that we discussed in Example 12.3, it picks the moves $q_0 \xrightarrow{\text{push}_\bot} q_1$ and $q_1 \xrightarrow{noop} q_2$. Additionally, it needs to pick the prediction $\{q_f\}$.

We leave it as an exercise for the reader to check that picking e.g. the set $\{q_1, q_f\}$ as prediction will not result in a winning strategy.

12.9 Definition: $\mathcal{G}^{FS}$

To the PDS game, we associate the finite state game $\mathcal{G}^{FS}$ on $G = (V, R)$, where V is defined in Figure 2 and R is defined in Figure 3.

The ownership of positions of type (q, a, P) is given by the ownership of q. All push-positions are owned by ○, as she has to make a prediction. All predict-positions are owned by □, as she has to choose between verifying the prediction and skipping the subplay. Ownership on all other types of positions does not matter, as they have at most one successor.

Formally, we have

$$\begin{array}{lll}
owner(q, a, P) & = \text{☆} \quad \text{iff } q \in Q_{\text{☆}} & \forall q \in Q, a \in \Delta_\varepsilon, P \subseteq Q\,, \\
owner(Push((q, a, P), (s, b))) & = \bigcirc & \forall q, s \in Q, a \in \Delta_\varepsilon, b \in \Delta, P \subseteq Q\,, \\
owner(Predict((q, a, P), (s, b, P'))) & = \Box & \forall q, s \in Q, a \in \Delta_\varepsilon, b \in \Delta, P, P' \subseteq Q\,, \\
owner(Pop(p, P)) & = \Box & \forall p \in Q, P \subseteq Q\,, \\
owner(Verify(s, b, P')) & = \Box & \forall s \in Q, b \in \Delta, P' \subseteq Q\,, \\
owner(Skip(p, a, P)) & = \Box & \forall p \in Q, a \in \Delta_\varepsilon, P \subseteq Q\,.
\end{array}$$

The winning set that ○ has to reach consists of all positions (q_f, a, Q') where the control state is q_f and of all pop-positions where the control state is contained in the prediction:

$$\begin{array}{rlllll}
B = & \{ & (q_f, a, P) & | & & a \in \Delta_\varepsilon, P \subseteq Q \quad \} \\
\cup & \{ & Pop(p, P) & | & p \in P, & a \in \Delta_\varepsilon, P \subseteq Q \quad \} \subseteq V\,.
\end{array}$$

The initial position of interest is

$$(q_0, \varepsilon, \emptyset)\,,$$

i.e. we start with the empty stack and the empty prediction.

Proof of correctness

It remains to show that $\mathcal{G}^{FS}$ is indeed the game required for Theorem 12.5. We divide the proof in two steps:

- Proposition 12.10: If ○ has a winning strategy for $\mathcal{G}^{PDS}$, then she has one for $\mathcal{G}^{FS}$.
- Proposition 12.11: If ○ has a winning strategy for $\mathcal{G}^{FS}$, then she has one for $\mathcal{G}^{PDS}$.

We start with the first direction, as it is the easier one.

$$V = \quad \left\{ \quad (q, a, P) \quad \middle| \quad \begin{array}{l} q \in Q, \\ a \in \Delta_\varepsilon, \\ P \subseteq Q \end{array} \right\}$$

// State + top-of-stack + current prediction

$$\cup \quad \left\{ \quad Push((q, a, P), (s, b)) \quad \middle| \quad \begin{array}{l} q, s \in Q, \\ a \in \Delta_\varepsilon, b \in \Delta, \\ P \subseteq Q \end{array} \right\}$$

// Owner of q wants to perform $q \xrightarrow{\text{push}_b} s$

$$\cup \quad \left\{ \quad Predict((q, a, P), (s, b, P')) \quad \middle| \quad \begin{array}{l} q, s \in Q, \\ a \in \Delta_\varepsilon, b \in \Delta, \\ P, P' \subseteq Q \end{array} \right\}$$

// Existential player makes a new prediction

$$\cup \quad \left\{ \quad Verify(s, b, P') \quad \middle| \quad \begin{array}{l} s \in Q, \\ b \in \Delta, \\ P' \subseteq Q \end{array} \right\}$$

// Universal player decides to verify - the push is performed

$$\cup \quad \left\{ \quad Skip(p, a, P) \quad \middle| \quad \begin{array}{l} p \in Q, \\ a \in \Delta_\varepsilon, \\ P \subseteq Q \end{array} \right\}$$

// Universal player trusts the prediction and skips the subplay

$$\cup \quad \left\{ \quad Pop(p, P) \quad \middle| \quad \begin{array}{l} p \in Q, \\ P \subseteq Q \end{array} \right\} .$$

// A pop has been performed, the game ends

Figure 2: The definition of the set of positions V of $\mathcal{G}^{FS}$.

$$R = \quad \left\{ \; (q,a,P) \to (s,a,P) \;\middle|\; q \xrightarrow{noop} s, \quad \begin{array}{l} q,s \in Q, \\ a \in \Delta_\varepsilon, \\ P \subseteq Q \end{array} \right\}$$

// Transition with no operation, keep current prediction

$$\cup \; \left\{ \; (q,a,P) \to Push((q,a,P),(s,b)) \;\middle|\; q \xrightarrow{\mathrm{push}_b} s, \quad \begin{array}{l} q,s \in Q, \\ a \in \Delta_\varepsilon, b \in \Delta, \\ P \subseteq Q \end{array} \right\}$$

// Owner of q wants to perform $q \xrightarrow{\mathrm{push}_b} p$

$$\cup \; \left\{ \; Push((q,a,P),(s,b)) \to Predict((q,a,P),(s,b,P')) \;\middle|\; \begin{array}{l} q,s \in Q, \\ a \in \Delta_\varepsilon, b \in \Delta, \\ P,P' \subseteq Q \end{array} \right\}$$

// Existential player makes a new prediction

$$\cup \; \left\{ \; \begin{array}{rcl} Predict((q,a,P),(s,b,P')) & \to & Verify(s,b,P')\,, \\ Verify(s,b,P') & \to & (s,b,P') \end{array} \;\middle|\; \begin{array}{l} q,s \in Q, \\ a \in \Delta_\varepsilon, b \in \Delta, \\ P,P' \subseteq Q \end{array} \right\}$$

// Universal player wants to verify the prediction, then the push is performed

$$\cup \; \left\{ \; \begin{array}{rcl} Predict((q,a,P),(s,b,P')) & \to & Skip(p,a,P)\,, \\ Skip(p,a,P) & \to & (p,a,P) \end{array} \;\middle|\; p \in P', \quad \begin{array}{l} q,p,s \in Q, \\ a \in \Delta_\varepsilon, b \in \Delta, \\ P,P' \subseteq Q \end{array} \right\}$$

// Universal player trusts the prediction and skips the subplay from q to p

$$\cup \; \left\{ \; (s',a,P) \to Pop(p,P) \;\middle|\; s' \xrightarrow{\mathrm{pop}_a} p, \quad \begin{array}{l} s',p \in Q, \\ a \in \Delta, \\ P \subseteq Q \end{array} \right\} .$$

// A pop has been performed

Figure 3: The definition of the set of moves R of $\mathcal{G}^{FS}$.

12.10 Proposition

If ○ has a winning strategy (for reaching q_f) for the PDS game $\mathcal{G}^{PDS}$ from (q_0, ε), then she has a winning strategy for the finite-state reachability game $\mathcal{G}^{FS}$ from $(q_0, \varepsilon, \varnothing)$.

Proof:

Towards a proof, we fix a winning strategy for $\mathcal{G}^{PDS}$. Since the PDS game is also a reachability game (although on an infinite graph), positional determinacy (Theorem 4.4) applies, and there is a uniform positional winning strategy $s_{\bigcirc}^{\mathrm{PDS}}$. (Note that this strategy is positional, but it still works on the configuration consisting of control state *and* stack content.) Our goal is to translate $s_{\bigcirc}^{\mathrm{PDS}}$ into a winning strategy $s_{\bigcirc}^{\mathrm{FS}}$ for $\mathcal{G}^{FS}$.

Construction of the strategy

We construct $s_{\bigcirc}^{\mathrm{FS}}$ as a non-positional winning strategy. To be able to apply $s_{\bigcirc}^{\mathrm{PDS}}$, we need to recover from a play of $\mathcal{G}^{FS}$ a position of $\mathcal{G}^{PDS}$, i.e. a full configuration consisting of control state and stack content.

Formally, let $p^{\mathrm{FS}} = p_0 p_1 \dots p_k$ be a finite play of $\mathcal{G}^{FS}$ from $(q_0, \varepsilon, \varnothing)$. We define the **associated stack content** $assoc(p^{\mathrm{FS}})$ as follows: Let $Verify(p_1, b_1, P_1), Verify(p_2, b_2, P_2), \dots, Verify(p_k, b_k, P_k)$ be the sequence of all verify positions in p^{FS} in their order of occurrence. Then

$$assoc(p^{\mathrm{FS}}) = b_1 b_2 \dots b_m \,.$$

In particular, we have $assoc(p^{\mathrm{FS}}) = \varepsilon$ if p^{FS} contains no verify-positions. Indeed, the verify-positions correspond to pushes that have been performed. The pushes that are skipped do not contribute to the associated stack content, as in the play of $\mathcal{G}^{PDS}$ we assume that the corresponding pop has also occurred.

We can now define the non-positional strategy $s_{\bigcirc}^{\mathrm{FS}}$ on plays that end with a position of type (q, a, P) as follows.

$$s_{\bigcirc}^{\mathrm{FS}}\big(p^{\mathrm{FS}}.(q,a,P)\big) = \begin{cases} Push((q,a,P)), (s,b))\,, \\ \qquad \text{if } s_{\bigcirc}^{\mathrm{PDS}}(q, assoc(p^{\mathrm{FS}})) = (s, assoc(p^{\mathrm{FS}}).b) \text{ with } q \xrightarrow{\mathrm{push}_b} s\,, \\ (s,a,P)\,, \\ \qquad \text{if } s_{\bigcirc}^{\mathrm{PDS}}(q, assoc(p^{\mathrm{FS}})) = (s, assoc(p^{\mathrm{FS}})) \text{ with } q \xrightarrow{noop} s\,, \\ Pop(p,P)\,, \\ \qquad \text{if } s_{\bigcirc}^{\mathrm{PDS}}(q, assoc(p^{\mathrm{FS}})) = (s, assoc(p^{\mathrm{FS}})^{\mathrm{pop}}) \text{ with } q \xrightarrow{\mathrm{pop}_a} s\,. \end{cases}$$

Here, $assoc(p^{\mathsf{FS}})^{\mathsf{pop}}$ is $assoc(p^{\mathsf{FS}})$ with the rightmost symbol removed. Note that this symbol has to be $a \in \Delta$ whenever $assoc(p^{\mathsf{FS}})$ is non-empty. (In the case that $a = \varepsilon = assoc(p^{\mathsf{FS}})$, a pop-transition cannot be performed anyhow.)

It remains to define $s_{\bigcirc}^{\mathsf{FS}}$ for plays p^{FS} that end with a position of type $Push((q, a, P), (s, b)))$. In such a position, the existential player should make a prediction $P' \subseteq Q$. Our intuition is that P' should contain all states that can be reached by popping b in a play conform to $s_{\bigcirc}^{\mathsf{PDS}}$.

Consider the configuration $(s, m.b)$ of $\mathcal{G}^{PDS}$ with $m = assoc(p^{\mathsf{FS}})$ in which $\mathcal{G}^{PDS}$ after the push has been performed, and consider the set of all plays from $(s, m.b)$ that conform to $s_{\bigcirc}^{\mathsf{PDS}}$:

$$\left\{p^{\mathsf{PDS}} \text{ play} \;\middle|\; p_0^{\mathsf{PDS}} = (s, m.b), p^{\mathsf{PDS}} \text{ conforming to } s_{\bigcirc}^{\mathsf{PDS}}\right\}.$$

We restrict ourselves to plays in which a pop corresponding to b occurs. (There might be plays in which b stays on the stack for the rest of the play.) Recall that the position after which the pop has occurred is the first j such that $p_j^{\mathsf{PDS}} = (p, m)$ for some state p. This means the stack content coincides with m, the stack content before the push, for the first time. The prediction P should consist of states p corresponding to such configurations.

There is one more restriction we need to make: We need to avoid that the universal player can skip the occurrence of the target control state q_f, in case it is between s and p. In this case, □ could avoid losing by skipping this segment of the play. To this end, we just exclude all plays that encounter control state q_f between s and p.

If the universal player decides to verify the prediction, excluding these plays will not hurt: If the play has already visited q_f, it will be won by the existential player, even if it ends in $Pop(p, P)$ with $p \notin P$.

Formally, we define

$$\text{Prediction}(s, m, b) = \left\{ p \in Q \;\middle|\; \begin{array}{c} \exists p^{\mathsf{PDS}} \text{ play with } p_0^{\mathsf{PDS}} = (s, m.b) \text{ conforming to } s_{\bigcirc}^{\mathsf{PDS}}, \\ \exists j \in \mathbb{N}\colon p_j^{\mathsf{PDS}} = (p, m)\,, \\ \forall j' < j\colon p_{j'}^{\mathsf{PDS}} = (p', m') \text{ with } m \neq m' \text{ and } p' \neq q_f \end{array} \right\}.$$

We then can define

$$s_{\bigcirc}^{\mathsf{FS}}\big(p^{\mathsf{FS}}.Push((q, a, P), (s, b)))\big) = Predict\big((q, a, P), (s, b, P')\big)$$
$$\text{with } P' = \text{Prediction}\big(s, assoc(p^{\mathsf{FS}}), b\big)\,.$$

Proving that the strategy is winning

To show that $s_{\bigcirc}^{\mathrm{FS}}$ is indeed winning, we want to use that $s_{\bigcirc}^{\mathrm{PDS}}$ is winning for $\mathcal{G}^{PDS}$.

First note that if (q, a, P) can be reached in $\mathcal{G}^{FS}$ by a play conforming to $s_{\bigcirc}^{\mathrm{FS}}$, say by p^{FS}, then the position $(q, assoc(p^{\mathrm{FS}}))$ can be reached in $\mathcal{G}^{PDS}$ by a play conforming to $s_{\bigcirc}^{\mathrm{PDS}}$: As long as no subplays are skipped, the play of $\mathcal{G}^{FS}$ proceeds exactly as the corresponding play of $\mathcal{G}^{PDS}$. Whenever a subplay is skipped, the state that the play jumps to is a state reachable by playing conforming to $s_{\bigcirc}^{\mathrm{PDS}}$ by the definition of Prediction(s, m, b). A formal proof of this fact using induction is conceptually easy, but tedious.

Let us now assume towards a contradiction that p^{FS} is a maximal play of $\mathcal{G}^{FS}$ conforming to $s_{\bigcirc}^{\mathrm{FS}}$ that is not won by p^{FS}. In particular, it does not contain control state q_f. We distinguish two cases:

- If p^{FS} ends with a pop-position, say $Pop(p, P')$, then it is not winning if and only if $p \notin P'$. However, the predictions are chosen such that this case cannot occur.

  Let $Predict((q, a, P), (s, b, P'))$ be the position in which prediction P' was chosen. This means $P' = $ Prediction$(s, assoc(pf), b)$, where pf is the prefix of the play before the prediction was chosen. The play of $\mathcal{G}^{FS}$ from $Predict((q, a, P), (s, b, P'))$ to $Pop(p, P')$ corresponds to a play of $\mathcal{G}^{PDS}$ from $(s, assoc(pf).b)$ to $(p, assoc(pf))$ that is conform to $s_{\bigcirc}^{\mathrm{PDS}}$. Hence, we have $p \in P'$ by definition.

- Else, i.e. if p^{FS} does not contain a pop-position, consider the play of $\mathcal{G}^{PDS}$ that is corresponding to p^{FS}. As already mentioned, it is conforming to the winning strategy $s_{\bigcirc}^{\mathrm{PDS}}$, and hence, it reaches state q_f after finitely many steps. Since we do not allow to skip subplays in which q_f occurs (see the definition of Prediction(s, m, b)), this means that p^{FS} also needs to contain an occurrence of q_f.

■

It remains to prove the other direction.

12.11 Proposition
If ○ has a winning strategy for the finite-state reachability game $\mathcal{G}^{FS}$ from $(q_0, \varepsilon, \varnothing)$, then she has a winning strategy for the PDS game $\mathcal{G}^{PDS}$ from (q_0, ε).

Again we can assume that we are given a uniform positional winning strategy $s_{\bigcirc}^{\mathrm{FS}}$ for $\mathcal{G}^{FS}$. As $\mathcal{G}^{FS}$ is finite, we can even assume that we have an explicit representation of this strategy.

In the following, we will not only prove the existence of a strategy $\mathcal{G}^{PDS}$, but we will also discuss how to obtain a finite representation.

The game arena of $\mathcal{G}^{PDS}$ is infinite-state: Even single configurations (q, m) cannot be represented using bounded space. However, we would like to obtain a strategy that only needs to process a bounded amount of information in each step. The control states themselves provide too little information for this approach to work. Thus, we consider strategies that work on the finite set of transition rules $\rightarrow$ of the pushdown system. This means the strategy will read the moves of the game that have been used.

Unfortunately, it is not easily possible to obtain such a strategy that is finite-state or even positional. The strategy that we construct will need unbounded memory. To be precise, we will build a so-called **pushdown strategy**, a strategy that maintains an unbounded stack as storage.

We give the idea behind the construction of the strategy and then argue why it needs unbounded memory.

In the notation of Figure 1, the strategy $s_{\bigcirc}^{\mathrm{FS}}$ is winning both the play ② in which the push is performed, as well as the play ③ in which the subplay is skipped. This will be guaranteed, as the universal player has to choose between verifying and skipping: Whenever a position of type $\mathit{Predict}((q, a, P), (s, b, P'))$ is in the winning region of $\bigcirc$, then both $\mathit{Verify}(s, b, P')$ and $\mathit{Skip}(p', a, P)$ (for all $p \in P'$) also have to be in the winning region.

The idea for the construction of $s_{\bigcirc}^{\mathrm{PDS}}$ is as follows: After a push has been made, say $(q, m.a) \rightarrow (s, m.a.b)$, the strategy first simulates $s_{\bigcirc}^{\mathrm{FS}}$ from $\mathit{Verify}(s, b, P')$. Since $s_{\bigcirc}^{\mathrm{FS}}$ is winning, it is guaranteed that if b is ever popped, we land in a state p with $p \in P'$. From this moment on, we can simulate $s_{\bigcirc}^{\mathrm{FS}}$ from $\mathit{Skip}(p, a, P)$ on.

However, this will require us to keep track of one prediction for every push that has been performed: After b has been popped in our example, a is again the top-of-stack. To behave properly (i.e. as required by $s_{\bigcirc}^{\mathrm{FS}}$), we need to know again the prediction that was made when a was pushed. Since the number of pushes is not bounded, we will need an unbounded storage.

The automaton implementing the strategy will always maintain the prediction for the current top-of-stack in the control state. When a push happens, it stores the current prediction on the stack and picks a new prediction for the new top-of-stack (guided by the strategy $s_{\bigcirc}^{\mathrm{FS}}$). When a pop happens, the current prediction can be forgotten and the correct prediction for the *new* top-of-stack is recovered from the stack.

We formally introduce pushdown strategies, define $s_{\bigcirc}^{\mathrm{PDS}}$ and finally prove the correctness.

12.12 Definition

A **pushdown strategy** for player ☆ is defined by a (deterministic) **pushdown transducer** T that reads the moves of a game $G = (V, R)$ (with fixed initial state x_0) and outputs the moves of ☆. More formally, the transducer is a tuple

$$T = (Q_T, R, R, \Delta_T, q_{T0}, \delta, \mathrm{o})$$

where

- Q_T is a finite set of **internal control states**,
- $q_{T0} \in Q_T$ is the **initial state** associated to x_0,
- the set of moves R is the **input** as well as the **output alphabet**,
- Δ_T is the stack alphabet of T,
- $\delta \subseteq Q_T \times R \times Ops_{\Delta_T} \times Q_T$ is the **transition relation**, and
- $\mathrm{o}\colon Q_T \to R$ is the **output function** that determines a the successor $\mathrm{o}(q_T)$ that is put out depending on the current internal state $q_T \in Q_T$.

The transition relation contains pairs of the shape

$$(q_T, r, op, p_T)\,,$$

meaning that when the transducer is in state $q_T \in Q_T$ and reads move $r \in R$ of the game, it can perform operation op (i.e. $noop$ or push_a or pop_a for some $a \in \Delta_T$) on its stack and go to state $p_T \in Q_T$.

We require that T is **deterministic** in the following sense: If there is a transition $(q_T, r, op, p_T) \in \delta$ for some $q_T \in Q_T$, $r \in R$ where the operation is $op = noop$ or a push ($op = \mathrm{push}_a$ for some $a \in \Delta_T$), then there is no other transition (q_T, r, op', p_T') for this q_T and r. Furthermore, for each $q_T \in Q_T$, $r \in R$ and $a \in \Delta_T$, there is at most one p_T such that $(q_T, r, \mathrm{pop}_a, p_T) \in \delta$.

(The transducer should also guarantee that it does not deadlock and whenever it outputs a position, this is actually a valid successor, but we leave these assumptions implicit.)

For such a transducer, we define its configuration $\text{config}(pr) \in Q_T \times \Delta_T^*$ after reading some finite sequence of moves $pr \in R^*$ inductively by

$$\begin{aligned} \text{config}(\varepsilon) &= (q_{T0}, \varepsilon)\,, \\ \text{config}\big(pr'.r\big) &= (p_T, m) \end{aligned}$$

where $(q_T, m') = \text{config}(pr')$, $(q_T, r, op, p_T) \in \delta$, and m is the result of applying op to m', i.e. $m.a = m'$ and $op = \text{pop}_a$, or $m' = m$ and $op = noop$, or $m = m'.a$ and $op = \text{push}_a$.

The strategy induced by the transducer can then by defined by

$$\begin{aligned} s_{☆}^T \;:\; Plays_{☆} &\to V \\ play &\mapsto \text{o}(q_T) \quad \text{where } (q_T, m) = \text{config}(play)\,. \end{aligned}$$

12.13 Remark

Comparing to the definition of finite-state strategies (and the corresponding transducers) in Definition 7.10 we have made several changes that go beyond allowing a stack as storage.

- As discussed argued above, the transducer now reads moves instead of states.
- Consequently, the initial state is associated to a fixed initial position of the game. The trick of choosing the real starting state by reading the first position of the game which we have employed in Definition 7.10 does not work for transducers that read moves: A trivial play p_0 consists of one position, but of no move.
- Although the transducer should be deterministic, we have formalized its transitions by a relation instead of a function. This is because we allow several pop-transitions (for different stack symbols) to be present at the same time, i.e. we may have $(q_T, r, \text{pop}_a, p_a), (q_T, r, \text{pop}_b, p_b) \in \delta$. However, at most one of these transitions is enabled in any configuration, namely the one that pops the current top-of-stack. This allows the transducer to obtain information about the old top-of-stack whenever a pop-occurs.

  The same concept can also be realized using a transition function, but at the cost of more syntax.

It remains to implement the strategy described before by a pushdown transducer T. Formally, we have

$$T = (Q_T, \to, \to, Q_T, q_{T0}, \delta, \text{o})\,,$$

where the components are specified below.

The states Q_T are precisely the positions of $\mathcal{G}^{FS}$ of type (q, a, P). In particular, the state of the transducer stores the control state q of the PDS, the current top-of-stack a, and the prediction P for the push which pushed a.

$$Q_T = \{(q, a, P) \mid q \in Q, a \in \Delta_\varepsilon, P \subseteq Q\} .$$

The initial state is $q_{T0} = (q_0, \varepsilon, \varnothing)$, the initial state of $\mathcal{G}^{FS}$.

The stack alphabet of T is also the set Q_T of states of shape (q, a, P). Whenever a push is performed, the current state is stored on the stack. On a pop, the state is taken from the current top-of-stack.

To define the transition relation, note that our transducer will not read a move $r \in R$ from the infinite set of transitions of the pushdown system, but it will read $q \xrightarrow{op} p \in \rightarrow$, the rule which induces transition r. Note that $\rightarrow$ is a finite set. For example, if the move $(q, m) \rightarrow (p, m.a)$ occurs in the game, the transducer will read $q \xrightarrow{\text{push}_a} p$. (To be consistent with the definition above, one can assume that the transducers reads transitions, but that all transitions that are induced by the same rule cause the same behavior.)

- Upon reading $q \xrightarrow{noop} s$ in state (q, a, P), the transducer performs no stack operation and moves to (s, a, P).

- Upon reading $q \xrightarrow{\text{push}_b} s$ in state (q, a, P), the transducer performs $\text{push}_{(q,a,P)}$, storing the old prediction on the stack. Let

  $$s_{\bigcirc}^{\text{FS}}(Push((q, a, P), (s, b))) = Predict\big((q, a, P), \big(s, a, P'\big)\big) ,$$

  i.e. P' is the new prediction picked by the positional strategy $s_{\bigcirc}^{\text{FS}}$. The new internal state is (s, b, P').

- Upon reading $s \xrightarrow{\text{pop}_b} p$ in state (s, b, P'), the system pops the top-most stack symbol, say (q, a, P). It then moves to (p, a, P).

All other cases can be undefined. Note that the definition guarantees that whenever the transducer is in state (q, a, P), then a is indeed the top-of-stack and q is the control state of the PDS. The transition relation is deterministic as required. Formally, we have

$$\begin{aligned}
\delta \;=\; & \left\{\left((q,a,P), q \xrightarrow{noop} s, noop, (s,a,P)\right) \mid q,s \in Q, a \in \Delta_{\varepsilon}, P \subseteq Q\right\} \\
\cup \; & \left\{\left((q,a,P), q \xrightarrow{\mathrm{push}_b} s, \mathrm{push}_{(q,a,P)}, (s,b,P')\right) \mid \begin{array}{l} s_{\bigcirc}^{\mathrm{FS}}(Push((q,a,P),(s,b))) = \\ \qquad Predict((q,a,P),(s,a,P')) \\ q,s \in Q, a \in \Delta_{\varepsilon}, b \in \Delta, \\ P,P' \subseteq Q, \end{array}\right\} \\
\cup \; & \left\{\left((s,b,P'), s \xrightarrow{\mathrm{pop}_a} p, \mathrm{pop}_{(q,a,P)}, (p,a,P)\right) \mid q,p,s \in Q, b \in \Delta, a \in \Delta_{\varepsilon}, P,P' \subseteq Q\right\}.
\end{aligned}$$

It remains to define the output function o. Note that is sufficient to define the output for states (q,a,P) with $q \in Q_{\bigcirc}$. The definition of the output function is induced by the strategy $s_{\bigcirc}^{\mathrm{FS}}$ for $\mathcal{G}^{FS}$.

$$\mathrm{o}(q,a,P) = \begin{cases} q \xrightarrow{\mathrm{push}_b} s\,, & \text{if } s_{\bigcirc}^{\mathrm{FS}}(q,a,P) = Push((q,a,P),(s,b))\,, \\ q \xrightarrow{noop} s\,, & \text{if } s_{\bigcirc}^{\mathrm{FS}}(q,a,P) = (s,a,P)\,, \\ q \xrightarrow{\mathrm{pop}_a} p\,, & \text{if } s_{\bigcirc}^{\mathrm{FS}}(q,a,P) = Pop(p,P)\,. \end{cases}$$

The fact that $s_{\bigcirc}^{\mathrm{FS}}$ is a valid strategy ensures that the transitions that are printed actually exist.

To finish the proof of Proposition 12.11, we need to show that $s_{\bigcirc}^{\mathrm{PDS}}$ is indeed winning.

Proof:
We have to show that the strategy $s_{\bigcirc}^{\mathrm{PDS}}$ induced by the pushdown transducer T is winning for $\mathcal{G}^{PDS}$ from (q_0,ε). Let p^{PDS} be a maximal play of $\mathcal{G}^{PDS}$ from (q_0,ε) conforming to $s_{\bigcirc}^{\mathrm{PDS}}$.

To prove that p^{PDS} is won by ○, we will construct an associated play p^{FS} of $\mathcal{G}^{FS}$ conforming to the strategy $s_{\bigcirc}^{\mathrm{FS}}$. Since $s_{\bigcirc}^{\mathrm{FS}}$ is winning and T is induced by $s_{\bigcirc}^{\mathrm{FS}}$, we will then obtain that p^{FS} and also p^{PDS} is winning.

The main challenge for the construction of p^{FS} is that plays in $\mathcal{G}^{FS}$ should not contain pops. Therefore, for every pop occurring in p^{PDS}, we delete the whole segment between the corresponding push and the pop from p^{PDS}. The result of applying this operation exhaustively is essentially a play of $\mathcal{G}^{FS}$.

Formally, the construction of p^{FS} is as follows. Initially, we have $p_0^{\mathrm{FS}} = (q_0,\varepsilon,\varnothing)$. Assume we have already constructed the prefix $p_0^{\mathrm{FS}} \ldots p_i^{\mathrm{FS}}$ corresponding to some prefix $p_0^{\mathrm{PDS}} \ldots p_j^{\mathrm{PDS}}$ of p^{PDS}. Let $p_i^{\mathrm{FS}} = (q,a,P)$. To construct p_{i+1}^{FS}, we proceed as follows:

- If p^{PDS}_{j+1} is obtained from p^{PDS}_{j} be a transitioned labeled by *noop*, i.e.

  $$(q, m) \to (s, m) \text{ via } q \xrightarrow{noop} s\,,$$

  then we define p^{FS}_{i+1} to be (s, a, P).

- If p^{PDS}_{j+1} is obtained from p^{PDS}_{j} be a transitioned labeled by push_b, i.e.

  $$(q, m) \to (s, m.b) \text{ via } q \xrightarrow{\mathrm{push}_b} s\,,$$

  we distinguish two cases:

  - If p^{PDS} contains the corresponding pop, i.e. if there is some index $j' > j$ that is the first index such that $p^{\mathrm{PDS}}_{j'} = (p, m)$, then we skip the part between push and pop. More precisely, we append to the part of p^{FS} that has already been constructed the following moves:

    $$Push((q, a, P), (s, b)).Predict\big((q, a, P), (s, b, P')\big).Skip(p, a, P).(p, a, P)\,,$$

    where $s^{\mathrm{FS}}_{\bigcirc}(Push((q, a, P), (s, b))) = Predict((q, a, P), (s, b, P'))$, i.e. P' is the prediction selected by $s^{\mathrm{FS}}_{\bigcirc}$.

    In the next step, we will then construct p^{FS}_{i+5} (the position following (p, a, P)) depending on $p^{\mathrm{PDS}}_{j'+2}$.

  - If p^{PDS} contains no corresponding pop, then the push is actually executed: We append to the part of p^{FS} that has already been constructed the following moves:

    $$Push((q, a, P), (s, b)).Predict\big((q, a, P), (s, b, P')\big).Verify\big(s, b, P'\big).(s, b, P')\,,$$

    where again $s^{\mathrm{FS}}_{\bigcirc}(Push((q, a, P), (s, b))) = Predict((q, a, P), (s, b, P'))$.

    In the next step, the construction proceeds by defining p^{FS}_{i+5} depending on p^{PDS}_{j+2}.

  - Since any pop in p^{PDS} has a corresponding push somewhere earlier in the play, we do not need to consider the case that p^{PDS}_{j+1} is obtained by a pop.

It is again tedious to check that p^{FS} is indeed a maximal play of $\mathcal{G}^{FS}$ that is conforming to $s^{\mathrm{FS}}_{\bigcirc}$. In particular, whenever a pop occurs, the state reached by the pop is contained in the current prediction. This fact is based on the definition of transducer T which relies on $s^{\mathrm{FS}}_{\bigcirc}$, and the fact that p^{PDS} is conforming to the strategy $s^{\mathrm{PDS}}_{\bigcirc}$ induced by the transducer.

To be precise, p^{FS} is the play of $\mathcal{G}^{FS}$ in which the existential player plays conforming to $s_{\bigcirc}^{\mathrm{FS}}$ and the universal player verifies the predictions that correspond to pushes that do not have a corresponding pop in p^{PDS}. For the pushes that do have a corresponding pop in p^{PDS}, □ jumps precisely to the control state which is visited by p^{PDS} after the pop.

Since $s_{\bigcirc}^{\mathrm{FS}}$ is a winning strategy, the play p^{FS} conforming to it must be winning. As p^{FS} is constructed to not contain any pop-position, this means p^{FS} visits control state q_f. Note that if $p_i^{\mathrm{FS}} = (q, a, P)$ for some i, then there is some index j such that the control state of p_j^{PDS} is q. Combing the arguments, we obtain that p^{PDS} visits control state q_f and is winning. ■

Theorem 12.5, and subsequently Theorem 12.4, is now obtained by combining the Propositions 12.10 and 12.11.

Concluding remarks

12.14 Remark

The strategy that we have constructed for the proof of Proposition 12.11 is not just an arbitrary pushdown strategy, it is a so-called **synchronized pushdown strategy**. This means that the transducer implementing the strategy pushes resp. pops precisely when the pushdown system defining the game pushes resp. pops. Consequently, the height of the stack of the pushdown system equals the height of the stack that forms the internal storage of the strategy transducer.

Such strategies have a big advantage over arbitrary pushdown strategies. Assume that $P = (Q, \rightarrow)$ is the underlying PDS for a pushdown game over stack alphabet Δ, and let $T = (Q_T, R, R, \Delta_T, q_{T0}, \delta, o)$ be a synchronized pushdown transducer implementing a strategy for player ☆. Since the stack heights of the PDS and T are equal, we can construct the cross-product, which is again a pushdown system

$$P@T = (Q \times Q_T, \rightarrow')$$

over stack alphabet $\Delta \times \Delta_T$. The idea is to always store the state of the PDS as well as the state of the transducer, and whenever the current PDS state is owned by ☆, then the next move of the system is determined by the output function of T.

The resulting PDS $P@T$ has only one type of non-determinism, namely non-determinism corresponding to player $\overline{☆}$. (The non-determinism for player ☆ has been resolved using T).

Checking properties of the strategy defined by T can now be done by checking properties of $P@T$ using standard algorithms for pushdown automata (e.g. a variant of the CYK algorithm for reachability). For example, assume that one wants to check whether T defines a strategy that is winning for a safety game, i.e. whether it guarantees that some state q_f is never reached. We can check whether any state of the shape (q_f, q_T) is reachable in $P@T$. If and only if the result is negative T indeed defines a winning strategy.

Recall that without the guarantee that the stack heights of two pushdown automata are equal, their cross product is not a pushdown system. (In fact, their cross product can be seen as a proper Turing machine, since the intersection-emptiness problem for context-free languages is undecidable.)

12.15 Remark

In the proof of Theorem 12.5, we have only constructed strategies for the existential player ○. However, a similar construction works for the universal player: A uniform positional winning strategy for □ on $\mathcal{G}^{PDS}$ induces a winning strategy for □ on $\mathcal{G}^{FS}$, and a uniform positional winning strategy for □ on $\mathcal{G}^{FS}$ induces a winning strategy for □ on $\mathcal{G}^{PDS}$ that can be implemented by a synchronized pushdown transducer.

12.16 Remark

For simplicity, we have only considered the case of control state reachability games, while in [Wal01], the more general case of parity games is considered.

Recall that a parity game on a pushdown system $P = (Q_\Box \cup Q_\bigcirc, \rightarrow)$ is given by a priority assignment $\Omega \colon Q \to \mathbb{N}$ on the control states.

The construction of $\mathcal{G}^{FS}$ needs some modifications in this case:

- All states are modified to keep track of the priorities, e.g. we consider states of shape (q, a, P, n). On every transition, the tracked priority is updated to be the maximum of the priorities that have been seen.
- $\mathcal{G}^{FS}$ is now a parity game.
- The priority of state (q, a, n, P) is the priority of q.
- Instead of choosing a single prediction, the existential player picks a family of predictions $(P_n)_n$, one prediction P_n per priority.
- The universal player can pick a priority n and then some $p \in P_n$ for some n and skip the subgame. In this case, the priority n which we assume has occurred in the subgame is the priority of the skip-position. Afterwards, the tracked priority is reset to 0.

- A pop-position $Pop(p, n', (P_n)_n)$ has even priority if and only if $p \in P_{n'}$. To avoid deadlocks, we can add self-loops to pop-positions.
- All other positions have priority 0.

The proof of correctness then only requires minor changes.

12.17 Remark

The trick used in Walukiewicz's reduction is very powerful and extends to classes of systems beyond pushdown systems, namely to **higher-order computation models** (like higher-order collapsible pushdown systems or higher-order recursion schemes) [CW07], and to certain kinds of games on **multi-pushdown systems** [Set09].

The guess & check approach has a long history in the domain of program verification. Using it in combination with the two types of non-determinism in a game comes from the **game semantics** for the **modal μ-calculus**, a certain kind of logics [EJ91].

13. The Borel hierarchy

In this section, we look at sufficient conditions for a Gale-Stewart game being determined. We state the **Borel determinacy theorem** which shows that for a large class of winning conditions, the corresponding games are determined. The theorem results in the so-called **Borel hierarchy** of winning conditions: Each condition that satisfies the requirements of the Borel determinacy theorem is in some level of the hierarchy, which characterizes the complexity of the winning condition. The conditions that we have looked at in Part II of the lecture are in low levels of the hierarchy.

Sources
The presentation here partially follows [ZKW].

The Borel hierarchy and the Borel determinacy theorem

We start by recalling some notation for (sets of) sequences.

13.1 Remark
Let V be a (not necessarily) finite set. We denote by V^* the set of sequences $v_0 \dots v_k$ over V of finite length and by V^ω the set of sequences $v_0 v_1 \dots$ over V^ω of infinite length.

Let $p^{fin} = v_0 \dots v_n, p^{fin'} = u_0 \dots u_k \in V^*, p^{inf} = w_0 w_1 \dots \in V^\omega$. Finite sequences $p^{fin}, pfin'$ can be concatenated, resulting in the finite-length sequence $p^{fin}.p^{fin'} = v_0 \dots v_n u_0 \dots u_k$. A finite sequence p^{fin} can be concatenated with the infinite sequence p^{inf}, resulting in the infinite sequence $p^{fin}.p^{inf} = v_0 \dots v_n w_0 w_1 \dots$.

For sets of sequences, we define their concatenation element-wise. Let $K, K' \subseteq V^*$ and $H \subseteq V^\omega$. We define

$$K.K' = \left\{ p^{fin}.p^{fin'} \in V^* \;\middle|\; p^{fin} \in K, p^{fin'} \in K \right\},$$
$$K.H = \left\{ p^{fin}.p^{inf} \in V^\omega \;\middle|\; p^{fin} \in K, p^{inf} \in H \right\}.$$

Using this notation, we can define the lowest level of the hierarchy, the open sets.

13.2 Definition: Open
Let A be a set. A set $B \subseteq A^\omega$ is **open** if it is of the shape

$$B = K.A^\omega$$

for some set $K \subseteq A^*$.

Intuitively, a set B is open if membership in B only depends on a finite prefix: $p \in V^\omega$ is in $B = K.A^\omega$ if and only if there is a partition $p = p^{fin}.p^{inf}$ such that $p^{fin} \in K$.

13.3 Lemma

The notion of being open defines a **topology**. This means the following properties hold:

a) $\varnothing$ and A^ω are open,

b) any union of open sets is again open, and

c) intersections of finitely many open sets are open.

Proof: Exercise 13.16. ■

13.4 Remark

In fact, the topology defined by the notion of being open above is a well-known topology, namely the **product topology on A^ω with respect to the discrete topology on** A.

In the **discrete topology** on A, each subset of A is open.

For a sequence $p = p_0p_1p_2 \ldots \in A^\omega$ and $j \in \mathbb{N}$, we define $\mathrm{proj}_j(p) = p_j$, the projection of p to the j^{th} component. For sets $B \subseteq A^\omega$, we define $\mathrm{proj}_j(B)$ element-wise, i.e.

$$\mathrm{proj}_j(B) = \left\{\mathrm{proj}_j(p) \,\middle|\, p \in B\right\}.$$

In the **product topology** on A^ω, a set $B \subseteq A^\omega$ is open if and only if it can be written as union

$$B = \bigcup_{i \in I} B_i\,,$$

where I is some index set (that may be infinite, even uncountable), each $B_i \subseteq A^\omega$ is a set and for each i, we have that

$$\mathrm{proj}_j(B_i) = A$$

for all but finitely many $j \in \mathbb{N}$.

(In the general definition, we would additionally require that $\mathrm{proj}_j(B_i) \subseteq A$ is open for all i and j. Here, we consider the discrete topology on A and this condition is trivially satisfied.)

The correspondence stated in the previous remark can be formally proven.

13.5 Lemma
Our definition of being open coincides with the definition of being open in the product topology.

Proof:
Assume $B \subseteq A^\omega$ is a set such that $\mathrm{proj}_i(B) = A$ for all but finitely many $i \in \mathbb{N}$. We show that B is open. Since unions of open sets are again open, Lemma 13.3, this shows that all open sets in the product topology are open according to our definition.

Let $i_0 \in \mathbb{N}$ be the greatest index i such that $\mathrm{proj}_i(B) \neq A$. We may write

$$B = \mathrm{proj}_0(B).\mathrm{proj}_1(B) \ldots \mathrm{proj}_{i_0}(B)A^\omega ,$$

which is open by definition.

Assume that $B = K.A^\omega$ is a set that is open according to our definition. We may write $K \subseteq A^*$ as disjoint union

$$K = \bigcup_{i \in \mathbb{N}} K^{(i)}$$

such that each $K^{(i)} = K \cap A^i$ contains exactly the sequences in K of length i. We get

$$B = \bigcup_{i \in \mathbb{N}} K^{(i)} A^\omega .$$

Note that for each set $K^{(i)}A^\omega$, we have $\mathrm{proj}_j\left(K^{(i)}A^\omega\right) = A$ for all $j > i$. This concludes the proof. ■

We can now define the further levels of the Borel hierarchy. The hierarchy consists of two branches, the $\mathbf{\Sigma}$ branch and the $\mathbf{\Pi}$ branch. The open sets are the lowest level of the $\mathbf{\Sigma}$ branch.

13.6 Definition: Borel hierarchy
Let A be a set. We define a hierarchy consisting of elements $\mathbf{\Sigma}^0_\alpha$ and $\mathbf{\Pi}^0_\alpha$ for all ordinal numbers $\alpha > 0$.

Each $\mathbf{\Sigma}^0_\alpha$ resp. $\mathbf{\Pi}^0_\alpha$ is a collection of subsets of A^ω.

- $\mathbf{\Sigma}^0_1$ contains the open sets,
- For each $\alpha > 0$, $\mathbf{\Pi}^0_\alpha$ contains the complements of sets in $\mathbf{\Sigma}^0_\alpha$, and
- For each $\alpha > 1$, $\mathbf{\Sigma}^0_\alpha$ contains countable unions of sets in $\mathbf{\Pi}^0_\beta$ for $0 < \beta < \alpha$.

$$\mathbf{\Sigma}^0_1 = \left\{ B \subseteq A^\omega \;\middle|\; B \text{ open} \right\},$$

$$\mathbf{\Pi}^0_\alpha = \left\{ C \subseteq A^\omega \;\middle|\; A^\omega \setminus C \in \mathbf{\Sigma}^0_\alpha \right\},$$

$$\mathbf{\Sigma}^0_\alpha = \left\{ B \subseteq A^\omega \;\middle|\; B = \bigcup_{i \in \mathbb{N}} C_i \text{, where each } C_i \in \mathbf{\Pi}^0_{\beta_i} \text{ for some } \beta_i < \alpha \right\}.$$

13.7 Remark

- The superscript 0 that all $\mathbf{\Sigma}^0_\alpha$ and $\mathbf{\Pi}^0_\alpha$ have is a part of the name.
- The natural numbers are a special case of ordinal numbers. Thus, the above definition in particular defines $\mathbf{\Sigma}^0_n$ and $\mathbf{\Pi}^0_n$ for all natural numbers $n > 0$.
- We give a down to earth explanation of the first levels of the Borel hierarchy.

  $\mathbf{\Sigma}^0_1$ = open sets,

  $\mathbf{\Pi}^0_1$ = closed sets (complements of open sets),

  $\mathbf{\Sigma}^0_2$ = countable unions of closed sets,

  $\mathbf{\Pi}^0_2$ = complements of countable unions of closed sets
  = countable intersection of open sets,

  $\mathbf{\Sigma}^0_3$ = countable unions of countable intersection of open sets,
- The sets in each branch of the hierarchy form a chain:

  $$\mathbf{\Sigma}^0_1 \subseteq \mathbf{\Sigma}^0_2 \subseteq \mathbf{\Sigma}^0_3 \subseteq \dots$$

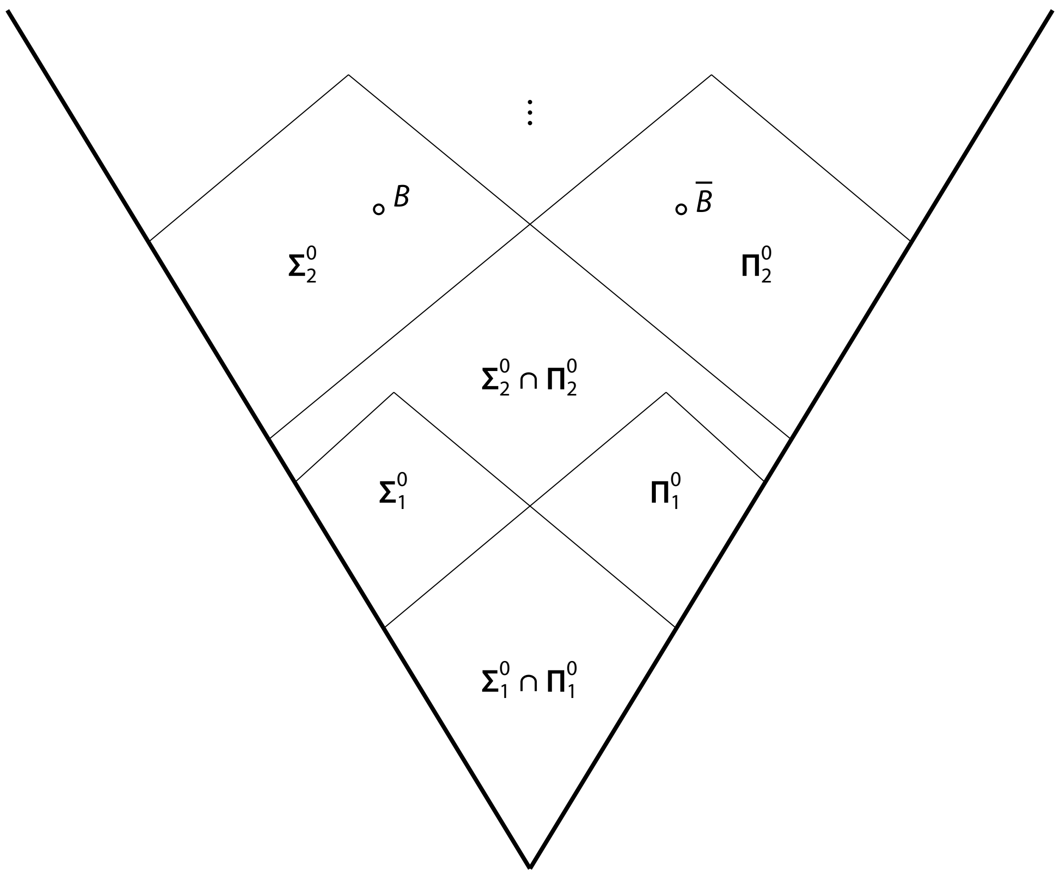
  $$\mathbf{\Pi}^0_1 \subseteq \mathbf{\Pi}^0_2 \subseteq \mathbf{\Pi}^0_3 \subseteq \dots$$

  More generally, if α, β are ordinal numbers with $\beta \leqslant \alpha$, then

  $$\mathbf{\Sigma}^0_\beta \subseteq \mathbf{\Sigma}^0_\alpha \qquad \text{and} \qquad \mathbf{\Pi}^0_\beta \subseteq \mathbf{\Pi}^0_\alpha .$$
- Furthermore, each $\mathbf{\Sigma}^0_\alpha$ contains all $\mathbf{\Pi}^0_\beta$ for $\beta < \alpha$, and similar for $\mathbf{\Pi}^0$.

We do not give a formal proof of these properties here.

The following figure depicts the first few levels of the Borel hierarchy. It takes the properties stated in the above remark into account.



The Borel determinacy theorem states that a game is determined if its winning condition lies in any countable level of the Borel hierarchy. We introduce the Borel algebra to make this formal.

13.8 Definition
The **Borel algebra** $\mathcal{B}$ is the union of the sets $\mathbf{\Sigma}^0_\alpha$ for all countable ordinals α. Equivalently, it can be defined to be the union over $\mathbf{\Pi}^0_\alpha$ for all countable ordinals α.

$$\mathcal{B} = \bigcup_{\alpha \text{ countable ordinal}} \mathbf{\Sigma}^0_\alpha = \bigcup_{\alpha \text{ countable ordinal}} \mathbf{\Pi}^0_\alpha .$$

A set $B \subseteq A^\omega$ is called **Borel set** if it is contained in the Borel algebra, $B \in \mathcal{B}$.

13.9 Remark
The natural numbers are the finite ordinals, and thus a special case of countable ordinals.

Therefore, the Borel algebra in particular contains all $\mathbf{\Sigma}^0_n$ and $\mathbf{\Pi}^0_n$ for $n \in \mathbb{N}, n > 0$.

The collection of open sets is closed under arbitrary unions, but not under countable intersections or complement. The Borel algebra has these properties.

13.10 Lemma

The Borel algebra $\mathcal{B}$ is the smallest collection of subsets of A^ω that contains the open sets and is closed under complement, countable union and countable intersection.

We omit the proof of this lemma.

We can now state the Borel determinacy theorem: Any Borel game, i.e. any game whose winning condition is a Borel set, is determined.

13.11 Theorem: Borel determinacy theorem, Martin 1975 [Mar75; Mar82]

Let A be a set. If $B \subseteq A^\omega$ is a Borel set, then the Gale-Stewart game $\mathcal{G}(A, B)$ is determined.

13.12 Corollary

Let A be a set and $B \subseteq A^\omega$. If B is in $\mathbf{\Sigma}^0_\alpha$ or $\mathbf{\Pi}^0_\beta$ for some countable ordinal α, then $\mathcal{G}(A, B)$ is determined.

The Borel hierarchy allows us to measure the complexity of winning conditions. Let B be a winning condition, then we can ask what is the least α such that $\mathbf{\Sigma}^0_\alpha$ resp. $\mathbf{\Sigma}^0_\beta$ contains B.

In the following, we want to study the complexity of several winning conditions that we have seen so far. Here, we consider Gale-Stewart games with reachability, parity, etc. winning conditions. If one wants to do this for the graph games that we have studied in the earlier sections, one can model the graph game as a Gale-Stewart game. We refer to Exercise 9.14.

13.13 Theorem

Reachability games are in $\mathbf{\Sigma}^0_1$, but not in $\mathbf{\Pi}^0_1$. Analogously, safety games are in $\mathbf{\Pi}^0_1$, but not in $\mathbf{\Sigma}^0_1$.

Proof sketch:

Consider a reachability games with respect to the winning set V_{reach}. Its winning condition is given by the set $B_{win} = V^* V_{reach} V^\omega$, which is open, but not closed (i.e. not the complement of an open set).

Analogously, let V_{reach} denote the losing set of a safety game. We have $B_{win} = V^\omega \setminus V^* V_{reach} V^\omega$, which is closed, but not open. ■

13.14 Theorem

Büchi games are in $\mathbf{\Sigma}^0_2$, but not in $\mathbf{\Pi}^0_2$.

Analogously, coBüchi games are in $\mathbf{\Pi}^0_2$, but not in $\mathbf{\Sigma}^0_2$.

Proof sketch:
Consider a Büchi game with respect to the winning set V_{reach}.

For each $i \in \mathbb{N}$, let

$$B^{(i)} = \underbrace{V^* V_{reach} V^* V_{reach} V^* \ldots V^* V_{reach}}_{i \text{ times}} V^\omega$$

denote the set of plays that visit V_{reach} at least i times. Note that each $B^{(i)}$ is open, but not closed. Consequently, for each i, the set $V^\omega \setminus B^{(i)}$ of plays that visit V_{reach} less than i times is closed, but not open.

The union

$$\bigcup_{i \in \mathbb{N}} V^\omega \setminus B^{(i)} ,$$

is the set of all plays that visit V_{reach} only finitely often, is thus in $\mathbf{\Sigma}^0_2$. Its complement, the set of all plays that visit V_{reach} infinitely often, is in $\mathbf{\Pi}^0_2$.

We could argue more directly and define

$$B_{win} = \bigcap_{i \in \mathbb{N}} B^{(i)} .$$

This is a countable intersection of open sets, thus in $\mathbf{\Pi}^0_2$. ■

13.15 Remark

The complexity of parity games depends on the exact definition. In Section 6, we have considered the highest priority occurring infinitely often, but restricted ourselves to a finite number of priorities (even when the arena is infinite). With this definition, parity games are in $\mathbf{\Sigma}^0_3 \cap \mathbf{\Pi}^0_3$, i.e. in $\mathbf{\Sigma}^0_3$ and in $\mathbf{\Pi}^0_3$, but not in $\mathbf{\Sigma}^0_2 \cup \mathbf{\Pi}^0_2$, i.e. neither in $\mathbf{\Sigma}^0_2$ nor in $\mathbf{\Pi}^0_2$. The same result holds for Muller games[1]

One can drop the restriction of having only finitely many priorities. However, one then needs to define a winner in the case that $\mathrm{Inf}(\Omega(p))$ has no well-defined maximum.[2] Parity games of this type are in higher levels of the Borel hierarchy.

[1] To define Muller games on an infinite arena, one usually assumes that there is a **coloring function** $c\colon V \to C$ that assigns each position one of **finitely many colors**. The winner now depends on the set of infinitely occurring colors $\mathrm{Inf}(c(p))$ in a play p.

[2] In this setting, one usually considers parity games in which the minimal priority occurring infinitely often is determining the winner, since any non-empty set of natural numbers has a minimum, but not necessarily a maximum.

Exercises

13.16 Exercise: Open sets
Let A be a set, and $B, B' \subseteq A^\omega$.

a) Prove that the empty set $\varnothing \subseteq A^\omega$ and A^ω itself are open.

b) Prove that if B and B' are open, then also their union $B \cup B'$ is open.

c) Prove that if B and B' are open, then also their intersection $B \cap B'$ is open.

Remark: This almost proves that the notion of being open defines a topology on A^ω, see Lemma 13.3. It remains to prove that arbitrary unions of open sets are open, which can be done similar to Part b).

Part IV.
Applications

Contents

14. Multiprocessor online scheduling

As a practical application of the reachability games that we studied in Section 4, we want to consider scheduling problems. A **scheduling problem** is of the following shape: Given a list of jobs and a list of processors, find a **scheduling**, an assignments of jobs to processors that has certain properties.

14.1 Example
Consider the well-known NP-complete **partition problem**.

Partition problem (PARTITION)
Given: A multiset S of natural numbers.
Question: Is there a partition $S = S_1 \uplus S_2$ such that $\sum_{s \in S_1} s = \sum_{s \in S_2} s$?

It can be seen as a scheduling problem: Given a list of jobs, each job having a given computation time, is there a scheduling of the jobs on two uniform processors such that both processors finish at the exact same time?

Sources
The content of this section is based on the papers [GGN17] and [Gee+18].

A multiprocessor online scheduling problem

The problem that we will consider in the following is an **online** scheduling problem. Instead of having a list of jobs that is known beforehand, we have a set of tasks that can generate jobs at runtime. The **(online) scheduler** has to react at runtime to jobs that are generated by the task without knowing when jobs will be generated in the future.

More precisely, our tasks are **sporadic**: Each task has a **minimal interarrival time** T, a timespan that is guaranteed to elapse between two generations of jobs for the task. Assume a job for the task is generated at time t. As soon as the minimal interarrival time has elapsed at time $t + T$, a new job of the task can be generated. It may not be generated immediately, it can be generated at an arbitrary later point in time that is not known to the scheduler.

One might think that the worst case for the scheduler occurs if every task generates a job immediately as soon as T has elapsed. This is not true: By allowing later generations, the future of the system becomes non-deterministic, which makes it harder for the online scheduler that has no knowledge of the future, see Exercise 14.16.

Each task has a **computation time** C, the time that a job for this task needs on the processor to be finished.

Furthermore, each task has a **relative deadline** D. Whenever a job of the task is generated, say at time t, it needs to be finished within a timespan of length D, i.e. at time $t + D$.

We will assume that we have some fixed number m of uniform **processors** to which we want to schedule the jobs. We discretize the model and assume that one computation step of the processors (called **tick**) decreases the remaining computation time of each scheduled job by 1. We assume that after each computation step, the jobs can be freely migrated between processors without causing a delay. Furthermore, we assume that each job has to be processed sequentially. This means that not more than one processor can work on the same job during one tick.

In the following, we will formally define the resulting **multiprocessor online feasibility of sporadic tasks problem (MOFST)**.

14.2 Definition
The input of MOFST is a set $\mathcal{T}$ of **tasks**, each task $\tau \in \mathcal{T}$ being a tuple $(C_\tau, D_\tau, T_\tau) \in (\mathbb{N} \setminus \{0\})^3$ consisting of the **computation time** C_τ, the **relative deadline** D_τ, and the **minimal interarrival time** T_τ.

Such an input gives rise to a system as described above. We can model the system naively as follows.

A configuration at time t of the system consists of

- a list of pending jobs $\mathcal{J}$, each job j specified by its remaining computation time RCT_j, and the time RD_j until its deadline (at time $t + RD_j$), and
- for each task τ in $\mathcal{T}$ the minimal time NAT_τ until its next arrival.

Initially, we consider the configuration at time 0, with an empty list of jobs, where each task τ has earliest arrival time $NAT_\tau = 0$.

In each tick, three things happen:

- The **tasks** may generate new jobs for eligible tasks: For each tasks τ that has $NAT_\tau = 0$, a new job j may be spawned. This job has $RCT_j = T_\tau$ and $RD_j = D_\tau$. If this happens, the remaining minimal interarrival time is reset, $NAT_\tau = T_\tau$.
- The **scheduler** may select up to m jobs and decrease their remaining computation time by one. If this results in $RCT_j = 0$, the job is deleted from the job list.

- The time until the deadline RD_j is decreased by one for each job, and for all tasks τ with $NAT_\tau > 0$, NAT_τ is decreased by one.

If a job has a negative deadline, i.e. $RD_j < 0$ after a time step, it has **missed** its deadline.

We call an input **feasible** for online scheduling if there is an online scheduler that schedules jobs such that no job ever misses its deadline, no matter when the jobs are generated at runtime.

Multiprocessor online feasibility of sporadic tasks problem (MOFST)

Given: A set of tasks $\mathcal{T}$, a number m of processors.

Question: Is the input feasible for online scheduling?

Note that the job list may contain more than one job per task while still being feasible, namely if $T_\tau < D_\tau$ for a task. In any configuration in which a job has not missed its deadline, the number of pending jobs for task τ is bounded by $\left\lceil \frac{D_\tau}{T_\tau} \right\rceil$.

We want to store a state as compact as possible, in particular we want to get rid of the job list. To this end, we assume that $T_\tau \geqslant D_\tau$ for each job τ. This means that for no task, two jobs can be pending at the same time without the earlier one already having missed its deadline. One can get rid of this assumption, but it has to be handled with care. Since it does not contribute to the concepts that we want to highlight here, we omit this.

MOFST as a safety game

In the following, we will model an instance of MOFST as a safety game.

- The reachability objective is given by the losing set of configurations in which a job will miss its deadline.
- The existential player ○ represents the tasks. As usual, she wants to satisfy the reachability objective. Her goal is to generate jobs such that a job will miss its deadline.
- The universal player □ represents the scheduler. She is trying to satisfy the complementary safety objective. She needs to schedule the jobs such that no job ever misses its deadline.

We represent configuration as above by **system states** $S \in \mathit{States}$. A state S is a tuple $S = (NAT_S, RCT_S)$, where

- $NAT_S\colon \mathcal{T} \to \mathbb{N}$ assigns to each task τ its earliest next arrival time $NAT_S(\tau) \leqslant T_\tau$, and
- $RCT_S\colon \mathcal{T} \to \mathbb{N}$ assigns to each task τ its remaining computation time $RCT_S(\tau) \leqslant C_\tau$.

In comparison to the configurations above, we have gotten rid of the job list using the assumption that we made. Furthermore, we have dropped the time until the deadline. We will see later that the deadline is still implicitly given by the two values that we store.

We call a task τ **active** in state S if $RCT_S(\tau) > 0$. This means that for this task, there is a pending job.

We call a task τ **eligible** in state S if $RCT_S(\tau) = 0$ and $NAT_S(\tau) = 0$. This means that for this task, there is no pending job, and its minimal interarrival time has elapsed since the last generation of a job.

It might seem strange that for a task τ to be eligible, it needs to have $RCT_S(\tau) = 0$. This is no real restriction, since if $NAT_S(\tau) = 0$, but $RCT_S(\tau) > 0$, than it has missed its deadline by the assumption $T_\tau > D_\tau$ that we made.

The actions of the existential player correspond to picking a set of eligible tasks and generating corresponding pending jobs. This resets the remaining computation time of these jobs to their computation time. The earliest next arrival time of the jobs that were scheduled is reset to T_τ.

Formally, for a state $S \in States$ and a set $\mathcal{T}' \subseteq \{\tau \in \mathcal{T} \mid \tau \text{ is eligible in } S\}$, $Succ_{\bigcirc}(S, \mathcal{T}')$ is the state S' with

$$RCT_{S'}(\tau) = \begin{cases} C_\tau, & \text{if } \tau \in \mathcal{T}' \\ RCT_S(\tau), & \text{else,} \end{cases}$$

and

$$NAT_{S'}(\tau) = \begin{cases} T_\tau, & \text{if } \tau \in \mathcal{T}' \\ NAT_S, (\tau) & \text{else.} \end{cases}$$

The moves of the universal player correspond to picking a set of active tasks and scheduling their corresponding pending jobs. This means that their computation time is decreased by one. Furthermore, we assume that the tick happens after the universal player has picked the jobs that should be scheduled, meaning the earliest interarrival time of all jobs is decreased by one.

Formally, for a state $S \in \mathit{States}$ and a set $\mathcal{T}' \subseteq \{\tau \in \mathcal{T} \mid \tau \text{ is active in } S\}$ of size at most m (the number of processors), $\mathit{Succ}_{\Box}(S, \mathcal{T}')$ is the state S' with

$$RCT_{S'}(\tau) = \begin{cases} RCT_S(\tau) - 1, & \text{if } \tau \in \mathcal{T}' \\ RCT_S(\tau), & \text{else,} \end{cases}$$

and $NAT_{S'}(\tau) = NAT_S(\tau) - 1$ for all τ.

$$NAT_{S'}(\tau) = \begin{cases} NAT_S(\tau) - 1, & NAT_S(\tau) > 0, \\ 0, & \text{else.} \end{cases}$$

The **game arena of the scheduling game** has as positions the elements of $\mathit{States} \times \{\bigcirc, \Box\}$, where the second component indicates the active player. The arcs can be partitioned into the arcs $R_{\Box}$ originating in positions owned by the universal player, and the arcs $R_{\bigcirc}$ originating in positions owned by the existential player,

$$\begin{aligned} R_{\Box} &= \left\{(S, \Box) \to (S', \bigcirc) \mid \mathcal{T}' \subseteq \{\tau \in \mathcal{T} \mid \tau \text{ is active in } S\}, |\mathcal{T}'| \leqslant m, S' = \mathit{Succ}_{\Box}(s, \mathcal{T}')\right\}, \\ R_{\bigcirc} &= \left\{(S, \bigcirc) \to (S', \Box) \mid \mathcal{T}' \subseteq \{\tau \in \mathcal{T} \mid \tau \text{ is eligible in } S\}, S' = \mathit{Succ}_{\bigcirc}(s, \mathcal{T}')\right\}. \end{aligned}$$

As one can see, the players alternately take turns.

We still need to specify the winning condition of the game. Instead of checking whether a job has actually missed its deadline, we will check whether it surely will miss its deadline. To this end, we define a function $\mathit{Laxity}_S \colon \mathcal{T} \to \mathbb{Z}$ by

$$\mathit{Laxity}_S(\tau) = D_\tau \underbrace{-\ (\ T_\tau - NAT_S(\tau)\ )}_{\text{time since last generation}} - RCT_S(\tau)\,.$$

Intuitively, the laxity measures for how many steps τ could stay idle in state S without risking missing its deadline: We take the deadline D_τ, subtract the time $T_\tau - NAT_S(\tau)$ that has elapsed since the last generation, and obtain the remaining time until the deadline. In the resulting timespan, we have to schedule the job for the task for at least $RCT_S(\tau)$ many ticks to avoid it missing its deadlines.

If the laxity of a task is negative, it will definitely miss its deadline, even if the corresponding job is scheduled consecutively in all following ticks.

14.3 Definition
The **scheduling game** is the reachability game on the previously defined game arena

$$G = (\mathit{States} \times \{\Box\} \uplus \mathit{States} \times \{\bigcirc\}, R_{\Box} \uplus R_{\bigcirc})$$

with respect to the winning set

$$B = \{(S, \bigcirc) \mid \exists \tau \text{ active in } S : \mathit{Laxity}_S(\tau) < 0\}\ .$$

Note that the winning set only consists of positions owned by the existential player. This is because we assume that the tick happens after the universal player picked the scheduling, so when the existential player is active, a tick has just elapsed.

14.4 Theorem
An input for MOFST is feasible if and only if the universal player has a winning strategy for the corresponding scheduling game from the position $(S_{init}, \bigcirc)$, where $\mathit{NAT}_{S_{init}}(\tau) = \mathit{RCT}_{S_{init}}(\tau) = 0$ for all tasks τ.

In the initial position, we assume that no job is pending and all tasks are eligible for generation. The existential player can start by generating a set of tasks.

A winning strategy for the universal player from this position directly corresponds to a scheduling policy.

14.5 Remark
Any play that is winning for the existential player, i.e. a play reaching a state S such that $\mathit{Laxity}_S(\tau) < 0$ for some active task τ also contains a position $(S', \bigcirc)$ with $\mathit{Laxity}_{S'}(\tau) = -1$. This allows us to redefine the winning set to

$$B = \{(S, \bigcirc) \mid \exists \tau \text{ active in } S : \mathit{Laxity}_S(\tau) = -1\}\ .$$

The size of *States* is

$$\prod_{\tau \in \mathcal{T}} (C_\tau + 1) \cdot \prod_{\tau \in \mathcal{T}} (T_\tau + 1) \leqslant (\max_{\tau \in \mathcal{T}} C_\tau + 1)^{|\mathcal{T}|} \cdot (\max_{\tau \in \mathcal{T}} T_\tau + 1)^{|\mathcal{T}|}\ .$$

Even if we assume that the number of tasks $|\mathcal{T}|$ is a constant, the size is polynomial in the numbers occurring in the tuples $\tau \in \mathcal{T}$, meaning in their unary encoding. If we assume that the numbers are encoded in binary, the size of *States* is exponential in the size of the input.

We have now obtained a reachability game on a large, but finite game arena. It can be solved using the attractor algorithm to determine whether the input is feasible. If the input is feasible, a uniform positional winning strategy for the universal player is the desired scheduling policy.

Unfortunately, the size of the arena makes this approach impractical for real-life applications. The i-step attractors that have to be computed are very large, and the winning strategy has to store one successor for each of the many positions owned by the universal player. Here, we will focus on the first problem. Our goal is to find compact representations for the attractors.

We will define the concepts for general reachability games, and then use them for the scheduling game.

TBA-simulations and attractor minimization

Let $G = (V_{\Box} \cup V_{\bigcirc}, R)$ be a reachability game on a finite graph with respect to the winning set $B \subseteq V$. We assume that G contains no deadlocks. Note that the scheduling game satisfies this property, since for each player, picking $\mathcal{T}' = \varnothing$ is always possible.

Recall that a relation $\trianglelefteq \subseteq V \times V$ is called a **partial order** if it has the following properties:

- **Reflexivity**: $\forall x \in V$: $x \trianglelefteq x$.
- **Transitivity**: $\forall x, y, z \in V$: If $x \trianglelefteq y$ and $y \trianglelefteq z$, then $x \trianglelefteq z$.
- **Antisymmetry**: $\forall x, y \in V$: If $x \trianglelefteq y$ and $y \trianglelefteq x$, then $x = y$.

In the following, we will assume that $\trianglelefteq$ is some fixed partial order on V.

Given a set $X \subseteq V$ of positions, we call $x \in X$ a **minimal element** of X if there is no element in X that is strictly smaller than x. In other words, for all $y \in X$, x is smaller than y, $x \trianglelefteq y$, or they are incomparable.

We define the operator Min that takes a set X and returns $\mathrm{Min}(X) \subseteq X$, the set of minimal elements of X. It can be computed by iteratively removing non-minimal elements from X. For each $y \in X$, Min contains an element x that is smaller than y [1].

Note that the elements in $\mathrm{Min}(X)$ form a so-called **antichain**: Two non-equal elements $x \neq y$ are not comparable. Assume that one would be smaller, then the other one would not be minimal. (Here, antisymmetry is important!)

Our aim is to define a variant of the attractor algorithm that works on minimal elements. This means that instead of $\mathrm{Attr}^i_{\bigcirc}$, we consider $\mathrm{Min}\left(\mathrm{Attr}^i_{\bigcirc}\right)$ which is hopefully much smaller. For this optimization to be valid, we need that $\mathrm{Min}\left(\mathrm{Attr}^i_{\bigcirc}\right)$ is a precise representation of $\mathrm{Attr}^i_{\bigcirc}$. The following notion will make this formal.

[1] For this statement to be true, we need to guarantee that minimal elements exist. This is true because any partial order on a finite set is well-founded.

We call a set X **upward closed** (with respect to the fixed partial order $\trianglelefteq$) if for each element $x \in X$, all elements $y \in V$ that are larger than x, are also contained in X, expressed as formula:

$$\forall x \in X \colon \forall y \in V \colon x \trianglelefteq y \text{ implies } y \in X\,.$$

Given an arbitrary set $X \subseteq V$, we let the **upward closure** of X, denoted by $X\uparrow$, be the set that contains for each element in X all larger elements:

$$X\uparrow = \{y \in V \mid \exists x \in X \colon x \trianglelefteq y\}\,.$$

It can easily be checked that the upward closure of a set X is indeed always upward closed. To be precise, the upward closure is the smallest upward-closed set containing X. A set X is upward closed if and only if it is its own upward closure, $X = X\uparrow$.

For upward-closed sets, the set of minimal elements considered before is an exact representation. The original set can be recovered by taking the upward closure.

14.6 Lemma

Let $X \subseteq V$ be upward closed, i.e. $X = X\uparrow$, then $X = \mathrm{Min}(X)\uparrow$.

Proof: Exercise 14.17, Part c). ■

It remains to characterize the partial orders such that the i-step attractors are upward-closed.

14.7 Definition

We call $\trianglelefteq$ a **turn based alternating simulation relation (tba-sim)** if it only relates positions owned by the same player,

$$\trianglelefteq \; \subseteq (V_\Box \times V_\Box) \cup (V_\bigcirc \times V_\bigcirc)\,,$$

and for all $x, y \in V$ with $x \trianglelefteq y$, the following properties hold:

- If $x \in B$, then $y \in B$.
- If $y \in V_\bigcirc$, then for all successors x' of x, there is a successor y' of y such that $x' \trianglelefteq y'$.
- If $y \in V_\Box$, then for all successors y' of y, there is a successor x' of x such that $x' \trianglelefteq y'$.

The following diagrams represent the latter two properties.

$$\begin{tikzcd} V_{\bigcirc} \ni y \arrow[r, dashed, "\exists"] & y' \\ x \arrow[r, "\forall"] \arrow[u, phantom, sloped, "\trianglelefteq"] & x' \arrow[u, phantom, sloped, "\trianglelefteq"] \end{tikzcd} \qquad \begin{tikzcd} V_{\square} \ni y \arrow[r, "\forall"] & y' \\ x \arrow[r, dashed, "\exists"] \arrow[u, phantom, sloped, "\trianglelefteq"] & x' \arrow[u, phantom, sloped, "\trianglelefteq"] \end{tikzcd}$$

Intuitively, $x \trianglelefteq y$ means that it is easer for the existential player to win from y than from x.

- The goal for the existential player is to reach B. Instead of reaching a position $x \in B$, the existential player can also win by reaching any larger position y, because it also has to be in B by the first condition.
- Whenever the existential player has a move in some position x, she has a better move in any larger position y. Better means that the result of the move y' from the larger position is larger than the result x' of the move in the small position.
- Whenever the universal player has a move from y to y', and $x \trianglelefteq y$, then there is a move from x to some x' with $x' \trianglelefteq y'$. This means that a larger position cannot suddenly give new possibilities to the universal player.

The following proposition makes this intuition precise by stating that indeed all i-step attractors are upward closed with respect to tba-sims.

14.8 Proposition
Let $\trianglelefteq$ be a tba-sim. Then for each $i \in \mathbb{N}$, $\mathrm{Attr}^i_{\bigcirc}(B)$ is upward closed.

Recall that

$$\mathrm{Attr}^{i+1}_{\bigcirc}(B) = \mathrm{Attr}^i_{\bigcirc}(B) \cup \mathrm{CPre}_{\bigcirc}(\mathrm{Attr}^i_{\bigcirc}(B)) .$$

Towards a proof of the proposition, we prove the following lemma.

14.9 Lemma
Let X be upward closed, and let $\trianglelefteq$ be a tba-sim. Then $\mathrm{CPre}_{\bigcirc}(X)$ is upward-closed.

Proof:
Let $x \in \mathrm{CPre}_{\bigcirc}(X)$ be arbitrary, and let $x \trianglelefteq y$. We have to show that $y \in \mathrm{CPre}_{\bigcirc}(X)$.

Note that we can assume that x, y are owned by the same player.

Assume that $x, y \in V_{\bigcirc}$ are owned by the existential player. Since $x \in \mathrm{CPre}_{\bigcirc}(X)$, x has at least one successor $x' \in X$. By the definition of tba-sim, there is a successor y' of y with

$x' \trianglelefteq y'$. We obtain $y' \in X\uparrow = X$. Since y is owned by the existential player, this proves $y \in \mathrm{CPre}_{\bigcirc}(X)$.

Assume that $x, y \in V_{\Box}$ are owned by the universal player. By the definition of tba-sim, for each such successor y' of y, there is a successors x' of x with $x' \trianglelefteq y'$. Since $x \in \mathrm{CPre}_{\bigcirc}(X)$, all these successors x' are contained in X. Since $x' \trianglelefteq y'$, we have $y' \in X\uparrow = X$ for all successors y'. Thus, $y \in \mathrm{CPre}_{\bigcirc}(X)$ as required. ■

Proof of Proposition 14.8:
We proceed by induction on i.

In the base case $i = 0$, we need to show that $\mathrm{Attr}^0_{\bigcirc}(B) = B$ is upward-closed. Let $x \in B$, and let $x \trianglelefteq y$. By the first condition of being a tba-sim, we have $y \in B$.

For the induction step, assume that $\mathrm{Attr}^i_{\bigcirc}(B)$ is upward closed. By Lemma 14.9, $\mathrm{CPre}_{\bigcirc}(\mathrm{Attr}^i_{\bigcirc}(B))$ is also upward closed. To conclude that $\mathrm{Attr}^{i+1}_{\bigcirc}(B) = \mathrm{Attr}^i_{\bigcirc}(B) \cup \mathrm{CPre}_{\bigcirc}(\mathrm{Attr}^i_{\bigcirc})$ is upward closed, note that the union of upward closed sets is upward closed in general, see Exercise 14.17 Part b). ■

The propositions means that each i-step attractor can be represented by its minimal elements without losing precision. As a consequence, we can define a variant of the attractor algorithm that directly works on the minimal elements. To this end, we define a variant of CPre that returns the minimal elements of the controlled predecessors,

$$\mathrm{MinCPre}_{\bigcirc}(X) = \mathrm{Min}(\mathrm{CPre}_{\bigcirc}(X\uparrow)) .$$

Using MinCPre, we can state the desired variant of the attractor algorithm.

$$\begin{aligned}\mathrm{MinAttr}^0_{\bigcirc}(B) &= \mathrm{Min}(B)\\ \mathrm{MinAttr}^{i+1}_{\bigcirc}(B) &= \mathrm{Min}\Big(\mathrm{MinAttr}^i_{\bigcirc}(B) \cup \mathrm{MinCPre}_{\bigcirc}\big(\mathrm{MinAttr}^i_{\bigcirc}(B)\big)\Big)\end{aligned}$$

14.10 Proposition
Let $\trianglelefteq$ be a tba-sim. Then for each $i \in \mathbb{N}$,

$$\mathrm{MinAttr}^i_{\bigcirc}(B) = \mathrm{Min}\big(\mathrm{Attr}^i_{\bigcirc}(B)\big) \quad \text{and}$$
$$\mathrm{Attr}^i_{\bigcirc}(B) = \mathrm{MinAttr}^i_{\bigcirc}(B)\uparrow .$$

Proof: Follows easily by induction with Proposition 14.8. ■

The consequence of this proposition is that to solve the reachability game, we can iteratively compute the sets $\mathrm{MinAttr}^{i}_{\bigcirc}(B)$ until they stabilize, i.e. until we reach an index i_0 with $\mathrm{MinAttr}^{i_0}_{\bigcirc}(B) = \mathrm{MinAttr}^{i_0+1}_{\bigcirc}(B)$. Then we know that $\mathrm{MinAttr}^{i_0}_{\bigcirc}(B)$ are the minimal elements of the winning region of the existential player. To check whether a position y is winning for the existential player, we have to check whether there is an element $x \in \mathrm{MinAttr}^{i_0}_{\bigcirc}(B)$ with $x \trianglelefteq y$.

Still, we are not done. We need to identify a non-trivial tba-sim that we can use for the algorithm. The trivial partial order $\{(x, x) \mid x \in V\}$ is a tba-sim, but for this order, the MinAttr algorithm will just be the normal attractor algorithm. The more elements are $\trianglelefteq$-related, the smaller the sets of minimal elements will become, and the more compact and thus more efficient the MinAttr algorithm will be. But the denser a relation $\trianglelefteq$ is, the harder it will be for it to satisfy the required condition for being a tba-sim.

Furthermore, if we implement the MinAttr algorithm naively, it will not lead to the desired boost in performance, in fact, it will most likely exhibit a performance that is worse than the one of the attractor algorithm.

In the first step, we need to obtain the minimal elements of B. If we do this by iteratively removing non-minimal elements from the set B that potentially can already be very large, this step might be very expensive.

In the following steps, we need to compute $\mathrm{MinCPre}_{\bigcirc}\big(\mathrm{MinAttr}^{i}_{\bigcirc}(B)\big)$. If we do this by definition, we will expand $\mathrm{MinAttr}^{i}_{\bigcirc}(B)$ to $\mathrm{Attr}^{i}_{\bigcirc}(B)$, then compute its controlled predecessors, and minimize again.

Note that there is a third step in the MinAttr algorithm that might seem problematic, namely the minimization after taking the union of $\mathrm{MinCPre}_{\bigcirc}\big(\mathrm{MinAttr}^{i}_{\bigcirc}(B)\big)$ and $\mathrm{MinAttr}^{i}_{\bigcirc}(B)$. But we expect these sets to be small, and thus minimizing their union in a naive way will not be very harmful.

A TBA-simulation for the scheduling game

In the following, we will move back to scheduling games. For these scheduling games, we will define a partial order, state that it is a tba-sim, and show that the two problematic operations mentioned above can be implemented in a clever way.

14.11 Definition

The **idle-ext task simulation $\underline{\blacktriangleleft}$** is a relation on $States \times \{\bigcirc, \square\}$ defined as follows. We have $(S, \star)\underline{\blacktriangleleft}(S', \star')$ iff $\star = \star'$ and for all taks $\tau \in \mathcal{T}$, we have

- $RCT_S(\tau) \leqslant RCT_{S'}(\tau)$,

- $RCT_S(\tau) = 0$ implies $RCT_{S'}(\tau) = 0$, and

- $NAT_S(\tau) \geqslant NAT_{S'}(\tau)$.

As stated before, the relation intuitively states that a state is larger if and only if it is easier for the existential player, the player representing the tasks, to win from this state. This means that each task has a longer remaining computation time (first condition) and can be generated again earlier (third condition). The second condition might look counter-intuitive; recall that having $RCT_S(\tau) = 0$ was a condition for a task to be eligible for generation.

14.12 Lemma
$\underline{\blacktriangleleft}$ is a partial order.

14.13 Theorem
$\underline{\blacktriangleleft}$ is a turn based alternating simulation relation.

The proofs are left to the reader as an exercise.

In the following we explain how $\mathrm{MinAttr}^0_{\bigcirc}(B) = \mathrm{Min}(B)$ and $\mathrm{MinCPre}_{\bigcirc}$ can be computed efficiently for $\underline{\blacktriangleleft}$.

First, we consider the computation of $\mathrm{Min}(B)$ for $\underline{\blacktriangleleft}$. Recall that

$$B = \{(S, \bigcirc) \mid \exists \tau \text{ active in } S : Laxity_S(\tau) < 0\}$$

and that

$$Laxity_S(\tau) = D_\tau + NAT_S(\tau) - T_\tau - RCT_S(\tau)\,.$$

If we have that τ is active in some state S, and $Laxity_S(\tau) < 0$, then

$$NAT_S(\tau) \leqslant T_\tau - D_\tau + RCT_S(\tau) - 1\,,$$

since we have $RCT_\tau > 0$.

For a single task τ, we define the set

$$Bad_{\underline{\blacktriangleleft}\tau} = \{(S, \bigcirc) \mid \exists j \in \{1, \ldots, C_\tau\}\colon NAT_S(\tau) = T_\tau - D_\tau + C_\tau - j\,, RCT_S(\tau) = C_\tau - (j-1)\}$$

One can check that for the scheduling game with the single task τ, we have $\mathrm{Min}(B) = \mathit{Bad}_{\underline{\blacktriangleleft}\tau}$. It remains to extend this concept to games with several tasks. We define $B_{\underline{\blacktriangleleft}\tau}$ to be the set of all states that are losing because of task τ, i.e.

$$B_{\underline{\blacktriangleleft}\tau} = \left\{ (S', \bigcirc) \;\middle|\; \begin{array}{ll} \exists (S, \bigcirc) \in \mathit{Bad}_{\underline{\blacktriangleleft}\tau}\colon & \mathit{NAT}_S(\tau) = \mathit{NAT}_{S'}(\tau) \text{ and } \mathit{RCT}_S(\tau) = \mathit{RCT}_{S'}(\tau), \\ \forall \tau' \neq \tau\colon & \mathit{NAT}_{S'}(\tau') = T_{\tau'} \text{ and } \mathit{RCT}_{S'}(\tau') \in \{0, 1\} \end{array} \right\} .$$

Finally, we define $B_{\underline{\blacktriangleleft}}$ as the union of the $B_{\underline{\blacktriangleleft}\tau}$,

$$B_{\underline{\blacktriangleleft}} = \bigcup_{\tau \in \mathcal{T}} B_{\underline{\blacktriangleleft}\tau} .$$

14.14 Lemma
$B_{\underline{\blacktriangleleft}} = \mathrm{Min}(B)$.

This finishes the first part of our study. We still have to show how to compute $\mathrm{MinCPre}_{\bigcirc}$. Let X be an antichain, i.e. a set where the elements are pairwise incomparable. Then we have

$$\begin{aligned} \mathrm{MinCPre}_{\bigcirc}(X) &= \mathrm{MinExPre}(X \cap V_{\Box}) \cup \mathrm{MinUnivPre}(X \cap V_{\bigcirc}), \\ &\quad \text{where} \\ \mathrm{MinExPre}(Y) &= \mathrm{Min}\big(\{x \in V \mid \exists \text{ successor } x' \text{ of } x \text{ with } x' \in Y\uparrow\}\big) , \\ \mathrm{MinUnivPre}(Y) &= \mathrm{Min}\big(\{x \in V \mid \forall \text{ successors } x' \text{ of } x\colon x' \in Y\uparrow\}\big) , \end{aligned}$$

where Y is an antichain. Note that if X is an antichain, then $X \cap V_{\bigcirc}$ and $X \cap V_{\Box}$ are antichains, too.

From the usual definition of the controllable predecessors, the above definition might look strange: We have a universal quantification for the existential player, the player whose perspective we take when computing the attractor, and an existential quantification for the universal player. When we consider $X \cap V_{\bigcirc}$, all predecessors will be owned by the universal player, thus the universal quantification is as expected. Similarly, all predecessors of vertices in $X \cap V_{\Box}$ are owned by the existential player, and we have an existential quantification.

It might look like the statement is missing an outermost minimization, i.e. one could think that the definition has to be $\mathrm{MinCPre}_{\bigcirc}(X) = \mathrm{Min}(\mathrm{MinExPre}(X \cap V_{\bigcirc}) \cup \mathrm{MinUnivPre}(X \cap V_{\Box}))$. This is not the case since the two sets each contain only positions owned by one of the players, and a tba-sim does not relate positions owned by different players.

We can now consider the cases of the universal player, i.e. $\mathrm{MinUnivPre}(X \cap V_{\square})$, and the case of the existential player, i.e. $\mathrm{MinExPre}(X \cap V_{\bigcirc})$ separately.

The case of the existential player is very easy. Instead of having to expand an antichain X to its upward closure $X\uparrow$, then taking the predecessors and minimizing again, we can directly take the predecessors of the minimal elements, and then minimize.

14.15 Lemma
Let $Y \subseteq V_{\bigcirc}$ be an antichain. Then

$$\mathrm{MinExPre}(Y) = \mathrm{Min}\big(\{x \in V \mid \exists \text{ successor } x' \text{ of } x \text{ with } x' \in Y\}\big)\,.$$

To get the efficient computation of $\mathrm{MinExPre}(Y)$ that we desire, note that it is possible to deterministically compute the predecessors for each position $x' \in Y$. This means that instead of iterating over all $x \in V$ and checking their successors, we can backtrack from the given set.

The case of the universal player is not that easy. It seems that considering some elements from $Y\uparrow$ that are not in the antichain Y cannot be avoided. We refer to [GGN17] for an algorithm that performs well in practice.

In [GGN17], several algorithms for solving scheduling games have been implemented and compared, including the naive attractor algorithm and the optimization discussed here. In random-generated examples, the optimized version outperforms the naive version by a factor of about 5 in running time, and of about 10 in space consumption.

Exercises

14.16 Exercise: An intricate scheduling problem
Consider the set of tasks $\mathcal{T} = \{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6\}$, where the computation time C_τ, the relative deadline D_τ, and the minimal interarrival time T_τ are given by the following table.

	C_τ	D_τ	T_τ
τ_1	2	2	5
τ_2	1	1	5
τ_3	1	2	6
τ_4	2	4	100
τ_5	2	6	100
τ_6	4	8	100

We assume that we have 2 processors. Recall that the jobs can be freely migrated between processors after each tick, but they have to be processed sequentially, i.e. not both processors can work on the same job during one tick.

a) Assume that each task generates a job as soon as the minimal interarrival time has elapsed, i.e. all tasks generate a job at time 0, τ_1 and τ_2 generate a job at time 5, τ_3 generates a job at time 6, and so on.

   Consider the time interval $[0, 8]$. Show that there is a scheduling of the jobs for this interval that makes no job miss its deadline.

   Give a graphic representation of your scheduling.

b) Prove that the input is infeasible for online scheduling if we allow the tasks to delay the generation of jobs.

   Hint: Towards a contradiction, assume that an online scheduler exists. Show that by time 8, at least one job has missed its deadline. Structure your proof as follows:

   - Assume that all tasks generate a job at time 0. Note that this fixes the jobs for the time interval $[0, 5)$, and since the online scheduler has no knowledge when which job will be generated later, fixes a scheduling on the interval.
   - For this fixed scheduling, there are two cases:
     - Case 1: The job generated by task τ_5 is not scheduled on any processor in the time interval $(2, 4]$.
     - Case 2: The job generated by task τ_5 is scheduled for at least one step on a processor in the time interval $(2, 4]$.

     Show that for each of the cases, there is a possible generation of jobs that makes a job miss its deadline.

Note: One can extend Part a) of the exercise beyond the interval $[0, 8]$ to an infinite run. Even if we drop the condition that each job is generated as soon as it becomes eligible and allow arbitrary delays, but we assume that the exact time of generation is known by the scheduler beforehand, the systems stays schedulable. This means the system is feasible for clairvoyant scheduling, but not feasible for online scheduling. You have proven the latter in Part b) of the exercise.

For the full, 28 pages long proof of the feasibility for clairvoyant scheduling, see [FGB10].

14.17 Exercise

Let $\trianglelefteq$ be a partial order on some set V.

a) Let $X, Y \subseteq V$ be subsets of V with $X \subseteq Y$. Prove that $X\uparrow \subseteq Y\uparrow$.

Does $\mathrm{Min}(X) \subseteq \mathrm{Min}(Y)$ also hold?

b) Prove that the union of upward-closed sets is again upward closed.

c) Prove Lemma 14.6:
Let $X \subseteq V$ be upward closed, i.e. $X = X\uparrow$, then $X = \mathrm{Min}(X)\uparrow$.

Hint: Prove both inclusions separately. For one inclusion, you can use Part a).

14.18 Exercise: The subword relation

Let Σ be some fixed, finite, non-empty alphabet. We consider the set of words Σ^* over Σ.

We define the **subword relation** $\preceq$ on Σ^* as follows: We have $v \preceq w$ if v can be obtained from w by deleting letters. This means that $w = a_0 a_1 \ldots a_k$ for some $a_i \in \Sigma$, and $v = a_{j_0} a_{j_1} \ldots a_{j_\ell}$ for $0 \leqslant j_0 < j_1 < \ldots < j_\ell \leqslant k$.

For example, consider the alphabet $\{a, b\}$ and $w = aba$. The words $\varepsilon, a, b, aa, ab, ba, aba$ are smaller with respect to $\preceq$ than w.

a) Prove that $\preceq$ is a partial order.

b) For each of the following languages over $\Sigma = \{a, b, c\}$, each represented by a regular expression, present their minimal elements and check whether they are upward-closed.

- $a\Sigma^* b\Sigma^* c$
- $ab \cup b\Sigma^* a \cup aabb$
- $c\Sigma^+ c$

Recall that $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$.

c) Let $w \in \Sigma$ be a word. How can one obtain a representation of the upward closure of the singleton set containing w, i.e. $\{w\}\uparrow$?

14.19 Exercise: A not so intricate scheduling problem

Consider the instance of MOFST with the tasks $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$ specified by the table below, and $m = 2$ processors.

	C_τ	D_τ	T_τ
τ_1	1	1	2
τ_2	2	2	2
τ_3	1	2	2

Construct and solve the scheduling game for this input.

15. Rabin's tree theorem

Let us now consider a theoretical application of game theory. As discussed in the introduction, the theory of games with perfect information can be used to obtain proofs for deep results in automata theory. In this section, we will see how we can use parity games to prove **Rabin's tree theorem**. Rabin's tree theorem states that the class of regular languages of infinite trees is closed under complementation. We will also discuss why this result is so important. Furthermore, we will see how we can use parity games to solve the language emptiness problem for tree automata.

Sources
The content of this section is based on Roland Meyer's notes on the topic.
They can be found here:
37_parity_tree_automata_part_3_MSOT.pdf

Infinite ranked trees

First, let us introduce infinite trees. We will consider trees whose nodes are labeled by letters from a finite alphabet. Each letter in the alphabet has an associated rank that determines the number of successors in the tree.

15.1 Definition
A **ranked alphabet** is a finite, non-empty set Σ together with a function $\text{rank}\colon \Sigma \to \mathbb{N}$ assigning each symbol a **rank**.

If $a \in \Sigma$ and $\text{rank}(a) = k$, we write $a_{/k} \in \Sigma$. We usually call just Σ ranked alphabet and mean that the rank-function is implicitly given.

15.2 Definition
Let Σ be a ranked alphabet with $\text{rank}(a) > 0$ for all $a \in \Sigma$.

An **infinite ranked Σ-labeled tree**, shortly referred to as **Σ-tree**, is a tree $\mathcal{T}$ in which each node $v \in \mathcal{T}$ is labeled by a symbol $a \in \Sigma$. If the label of a node is $a \in \Sigma$, then it has exactly $\text{rank}(a)$ many successors.

We formalize this as follows: We identify each node v with its **address**,
a sequence of natural numbers, i.e. in $\mathbb{N}^*$.

- The address of the root node is ε.

- Let v be the address of a node, and let $a_{/k} \in \Sigma$ be the label of this node. Then also $v.0, \ldots, v.(k-1)$, are valid addresses, namely the addresses of the successors of v.

This allows us to see $\mathcal{T}$ as an infinite, prefix closed subset of $\mathbb{N}^*$ together with a labeling function

$$\text{label}: \mathcal{T} \to \Sigma\,.$$

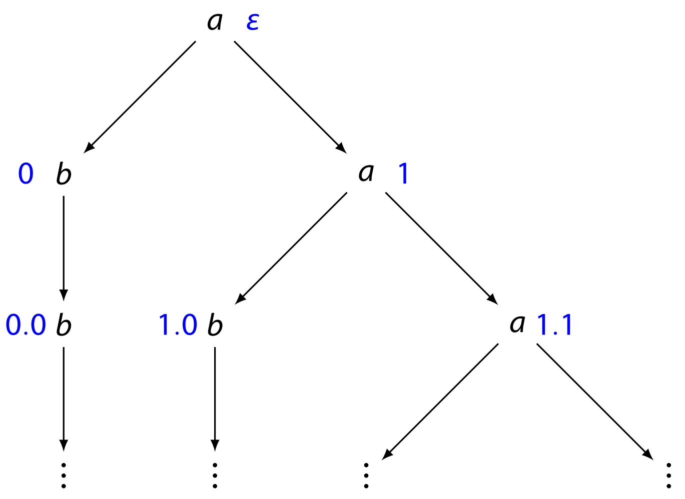
As for ranked alphabets, we say that $\mathcal{T}$ is a Σ-Tree and mean that the labeling function is implicitly given.

A **branch** of such a tree $\mathcal{T}$ is an infinite path starting in the root. It can be identified with a sequence $\pi \in \mathbb{N}^\omega$ such that for each $i \in \mathbb{N}$, the prefix of length i is a valid address in $\mathcal{T}$, i.e. $\pi_0 \ldots \pi_{i-1} \in \mathcal{T}$. For each $i \in \mathbb{N}$, the prefix $\pi_0\pi_1 \ldots \pi_{i-1}$ is the address of the i^{th} node in the path.

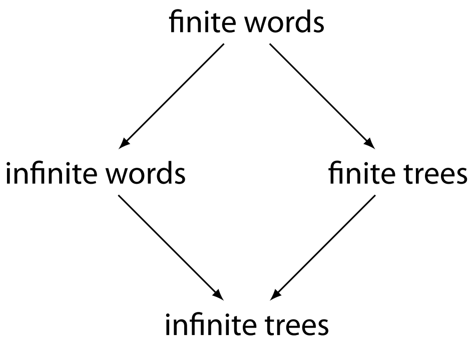
Note that the assumption $\text{rank}(a) > 0$ guarantees that all branches of any Σ-tree are infinite: There can be no leaves, since a leaf would have a label $a_{/k}$ with $k > 0$, and thus also have $k > 0$ many successors. The theory can easily be extended to allow trees in which some branches are finite, but this would lead to nasty case distinctions.

15.3 Example

Consider the ranked alphabet $\Sigma = \{a_{/2}, b_{/1}\}$. The following figure depicts a prefix of a Σ-tree. Next to each node, the text in blue color annotates its address.



One may ask why infinite trees are of interest.



Finite words can model finite executions of systems. Going from finite words to infinite ones is needed to model reactive systems, systems that may run forever, e.g. operating systems and database servers. Trees are needed to model branching behavior. Consequently, infinite trees can model the branching behavior of reactive systems.

Parity tree automata

We cannot even represent a single infinite tree explicitly in memory, much less sets of such trees. To solve this problem, we will consider automata that operate on such trees. An automaton then serves as a finite description for the set of trees it accepts.

15.4 Definition
A **parity tree automaton (PTA)** A is given by a tuple

$$A = (\Sigma, Q, q_0, \rightarrow, \Omega)\,,$$

where

- Σ is a ranked alphabet,
- Q is a finite set of control states,
- $q_0 \in Q$ is the initial state,
- $\Omega \colon Q \rightarrow \mathbb{N}$ is a function assigning each state a **priority**, and
- $\rightarrow\, = (\rightarrow_a)_{a\in\Sigma}$ is a family of transition relations, where $\rightarrow_a \subseteq Q \times Q^{\mathrm{rank}(a)}$ for each $a \in \Sigma$.

Note that the automaton is non-deterministic: For a symbol a and a state q, there might exist several (or no) vectors of states $\vec{q} \in Q^{\mathrm{rank}(a)}$ with $(q, \vec{q}) \in\rightarrow_a$.

To check whether a tree is accepted by an automaton, we need to consider runs. For finite words, we can see a run of a finite automaton on a word as an assignment of states to each letter of the word (namely the state in which the automaton is after reading each letter). Similarly, for infinite trees a run is an assignment of states to nodes of the tree.

15.5 Definition

Let $\mathcal{T}$ be a Σ-tree, and $A = (\Sigma, Q, q_0, \rightarrow, \Omega)$ a PTA.

A **run** of A on $\mathcal{T}$ is a function

$$\text{run}\colon \mathcal{T} \rightarrow Q$$

that assigns each node a control state such that the following properties hold:

- $\text{run}(\varepsilon) = q_0$, i.e. the root node is assigned the initial state q_0.

- For each $v \in \mathcal{T}$ with $\text{label}(v) = a_{/k} \in \Sigma$ and its successors $v.0 \dots v.k-1$, we have

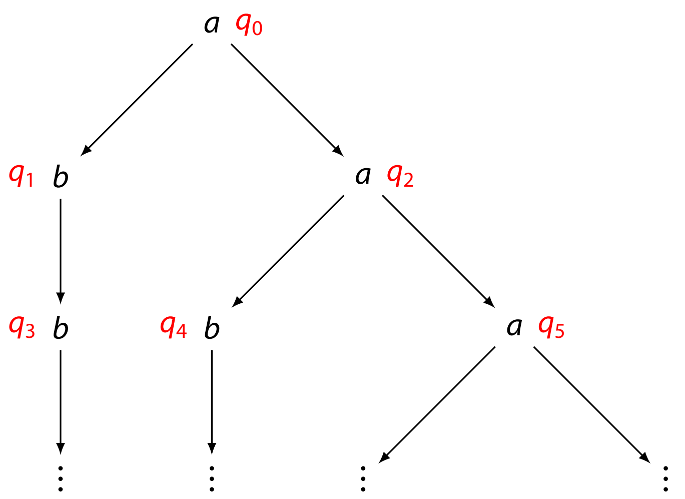
  $$(\text{run}(v), (\text{run}(v0), \dots, \text{run}(v.k-1))) \in \rightarrow_a \ .$$

  This means the assignment of states is consistent with the transitions of the automaton.

A run is called **accepting** if on every branch π of the tree, $\max \text{Inf}(\Omega(\text{run}(\pi)))$ is even, i.e. the highest priority occurring infinitely often is even. Here, we have extended the function run to branches, i.e. it takes a branch and yields the infinite sequences of states seen on the branch. As in the previous section, we have lifted Ω from single elements to sequences in the obvious way.

15.6 Example

We consider a state labeling on the tree from Example 15.3. Next to each node, the text in red color annotates its address.



For the given state labeling to be a prefix of a valid run of a PTA A, we need the following conditions to hold:

- q_0 is the initial state of A,
- $(q_0, (q_1, q_2)) \in \rightarrow_a$,
- $(q_1, (q_3)) \in \rightarrow_b$,
- $(q_2, (q_4, q_5)) \in \rightarrow_a$.

A language $\mathcal{L}$ of Σ-trees is a set of Σ-trees (just like a language of words was just a set of words without further restrictions).

15.7 Definition
The **language** $\mathcal{L}(A)$ **of a PTA** $A = (\Sigma, Q, q_0, \rightarrow, \Omega)$ is the set of all Σ-trees on which A has an accepting run,

$$\mathcal{L}(A) = \{\mathcal{T} \mid \mathcal{T}\ \Sigma\text{-tree}, \exists \text{ accepting run of } A \text{ on } \mathcal{T}\}\,.$$

15.8 Definition
A language $\mathcal{L}$ of Σ-trees is called **regular** if it is PTA-recognizable, i.e. there is a PTA $A = (\Sigma, Q, q_0, \rightarrow, \Omega)$ with $\mathcal{L} = \mathcal{L}(A)$.

As suggested by the name, the regular languages of infinite trees are indeed a generalization of the regular languages of finite words to the setting of infinite trees.

15.9 Definition

Let $\mathcal{L}$ be a language of Σ-trees. Its complement $\overline{\mathcal{L}}$ is the set of all Σ-trees that are not in $\mathcal{L}$,

$$\overline{\mathcal{L}} = \{\mathcal{T}\ \Sigma\text{-tree} \mid \mathcal{T} \notin \mathcal{L}\}\ .$$

Rabin's tree theorem

We have now gathered the prerequisites to state Rabin's tree theorem.

15.10 Theorem: Rabin's tree theorem

The class of regular languages of infinite, labeled, ranked trees is closed under complementation. Given a PTA A, we can effectively construct a PTA $\overline{A}$ accepting the complement language, $\mathcal{L}(\overline{A}) = \overline{\mathcal{L}(A)}$.

Note that the second line of the theorem provides a strictly stronger statement than the first line. The closure property just means that for any regular language $\mathcal{L}$ of Σ-trees, the complement language $\overline{\mathcal{L}}$ is also regular. Using automata, this means that for any PTA A, there is a PTA $\overline{A}$ accepting the complement language $\overline{\mathcal{L}(A)}$. However, this does not necessarily imply that we are able to explicitly construct this PTA $\overline{A}$.

As mentioned earlier, (possibly infinite) languages of infinite trees cannot be explicitly stored, we thus represent them by automata. In order to manipulate languages, we want to manipulate the automata describing them. Rabin's tree theorem tells us that this is possible for taking the complement: To obtain a description of the complement of a language, we construct an automaton based on the given automaton for the original language.

15.11 Remark

One might ask whether Rabin's tree theorem is a deep result (and we thus expect its proof to be complicated).

We recall the closure properties of regular languages of finite words. By definition, regular languages of finite words are only closed under union, concatenation and Kleene-star. That they are also closed under complementation is a theorem that we have seen in a basic course on automata theory (e.g. "Theoretische Informatik I").

Recall that the trick for finite automata was to swap the final with the non-final states. The question is whether this trick is also applicable here.

First, note that even for finite automata, the trick required the automaton to be deterministic. The language of a non-deterministic finite automaton is the set of all words that have an accepting run. If we swap the final with the non-final states in such an automaton, we obtain the set of all words that had a non-accepting run in the original automaton. This is not the complement of the language, which is the set of all words that had *no* accepting run in the original automaton.

For finite automata, the requirement of being deterministic posed no real problem, since we can apply the powerset construction to a given non-deterministic finite automaton to obtain a language-equivalent deterministic finite automaton. For parity tree automata, this is not possible. One can prove that for top-down tree automata, non-determinism is strictly more powerful than determinism. This applies to parity tree automata: There are languages of Σ-trees that are regular, i.e. can be recognized by a non-deterministic PTA, but that are not recognized by any deterministic PTA. We give an example in Exercise 15.43.

Assume for a moment we would restrict ourselves to deterministic PTA. Note that a deterministic PTA has a unique run on a tree. One may ask whether the trick of swapping final and non-final states works in this setting. Assume that some deterministic PTA $A = (\Sigma, Q, q_0, \rightarrow, \Omega)$ is given. To implement the trick, we define a new priority function $\Omega' \colon Q \rightarrow \mathbb{N}$ by

$$\Omega'(q) = \Omega(q) + 1 \ .$$

Note that for an infinite sequence of states p, we have that $\max \mathrm{Inf}(\Omega'(p))$ is even if and only if $\max \mathrm{Inf}(\Omega(p))$ was odd.

Consider the deterministic PTA $A' = (\Sigma, Q, q_0, \rightarrow, \Omega')$. It does not accept the complement language of $\mathcal{L}(A)$: A' accepts all trees $\mathcal{T}$ in which for all branches π, $\max \mathrm{Inf}(\Omega'(\mathrm{run}(\pi)))$ is even. This means that in *all* branches π, $\max \mathrm{Inf}(\Omega(\mathrm{run}(\pi)))$ is odd. This is not equal to the complement language, the language of trees in which $\max \mathrm{Inf}(\Omega(\mathrm{run}(\pi)))$ is odd for *at least* one branch π.

Together, these two issues indicate that proving Rabin's tree theorem will be much more involved than proving that regular languages of finite words are closed under complementation.

To prove the theorem, we want to use parity games. Before we consider parity games for languages, we restrict ourself to the case of a single fixed tree.

Given a Σ-tree $\mathcal{T}$ and a PTA A, we want to construct a parity game that is won by the existential player if and only if the tree $\mathcal{T}$ is accepted by the automaton A. To this end, the existential player represents the automaton, she has to select transitions that result in an accepting run. The universal player wants to show that the tree is not accepted by choosing a branch on which the acceptance condition is not satisfied.

15.12 Definition
Let $\mathcal{T}$ be a Σ-tree and let $A = (\Sigma, Q, q_0, \rightarrow, \Omega_A)$ be a PTA.

We define the parity game $\mathcal{G}(\mathcal{T}, A)$ as follows:

- $V_{\bigcirc} = \mathcal{T} \times Q$, i.e. a position (v, q) of the existential player consists of an address $v \in \mathcal{T} \subseteq \mathbb{N}^*$ of a node in the tree, and a state $q \in Q$.

- $V_{\Box} = \mathcal{T} \times Q^{\leqslant n}$, where $n = \max_{a \in \Sigma} \operatorname{rank}(a)$, i.e. positions $(v, \vec{q})$ of the universal player consist of addresses v and a vector of states $\vec{q}$.

- The arcs are defined per player as follows:

  $$\begin{aligned} R \quad = \quad & \{((v, q), (v, \vec{q})) \mid (v, q) \in V_{\bigcirc}, \exists (q, \vec{q}) \in \rightarrow_a, \text{ where } a = \operatorname{label}(v)\} \\ \uplus \quad & \{((v, \vec{q}), (v.i, \vec{q}_i)) \mid (v, \vec{q}) \in \mathcal{T} \times Q^k \subseteq V_{\Box}, i \in \{0, \ldots, k-1\}\} \, . \end{aligned}$$

  This means the players take turns. The existential player, representing the automaton, picks a transition that respects the old state and the label of the current node.

  The universal player iteratively picks a branch of the tree by selecting a successor of the current node. The new state is then the corresponding component of the state vector that was picked by the existential player earlier.

- The priority function is defined as follows.

  $$\Omega(v, q) = \Omega_A(q) \, ,$$
  $$\Omega(v, \vec{q}) = 0 \, .$$

  On the right-hand side, Ω_A refers to the priority function of the automaton A. The vertices in $V_{\Box}$ have no relevant priority, only the priorities of the existential player's positions matter, as they represent nodes of the tree in the run.

With this construction, the desired correspondence holds.

15.13 Lemma
$\mathcal{T}$ is accepted by A if and only if the existential player wins the parity game $\mathcal{G}(\mathcal{T}, A)$ from position (ε, q_0),

$$\mathcal{T} \in \mathcal{L}(A) \quad \text{iff} \quad (\varepsilon, q_0) \in W_{\bigcirc}^{\mathcal{G}(\mathcal{T}, A)} \, .$$

Proof:
Using Theorem 6.7, we know that exactly one player has a positional winning strategy from (ε, q_0).

A winning strategy for the existential player yields an assignment of states that guarantees the highest priority occurring infinitely often on each branch to be even, i.e. an accepting run.

A winning strategy for the universal player identifies for each possible run a branch for which the highest priority occurring infinitely often is odd, i.e. a witness for the tree to be non-accepting.

The reader is encouraged to work out the details, see Exercise 15.45. ■

Note that it is important that we first let the existential player pick the transition and then let the universal player pick the successor. This allows the universal player to react to the way in which the existential player chose to resolve the non-determinism of the automaton.

15.14 Remark
Although the proof of Lemma 15.13 is straightforward, there is something surprising about the result.

If $\mathcal{T}$ is not in $\mathcal{L}(A)$, then any run of A on $\mathcal{T}$ is not accepting. This means that one can find a branch π of $\mathcal{T}$ on which $\max \mathrm{Inf}(\Omega(\mathrm{run}(\pi)))$ is odd. This branch is then a witness for the run not being accepting.

The difference to the result above is that we assume that the run is given, and then identify the branch violating the acceptance condition. In the parity game, the universal player needs to identify the violating branch on the fly: In each step, she has to prolong the branch by one move without knowing the full run. She only knows the run on the prefix of the tree that has been explored so far, but she does not know how the existential player will resolve the non-determinism of the automaton on the parts of the tree that are yet to come.

Proof approach (informal):

To prove Rabin's tree theorem, we need to lift the parity game approach from a single fixed tree to all trees. Nevertheless, Lemma 15.13 will be very helpful. For some fixed tree $\mathcal{T}$, the universal player $\square$ has a positional winning strategy for $\mathcal{G}(\mathcal{T}, \mathcal{A})$ if and only if $\mathcal{T} \notin \mathcal{L}(A)$. This statement is obtained by negating both sides of the equivalence stated in Lemma 15.13 and applying the positional determinacy of parity games. Hence, we have that $\overline{\mathcal{L}(A)}$ is the set of all trees $\mathcal{T}$ such that □ has a positional winning strategy for $\mathcal{G}(\mathcal{T}, A)$:

$$\overline{\mathcal{L}(A)} = \{\mathcal{T}\ \Sigma\text{-tree} \mid \exists s_{\Box} \text{ positional winning strategy for } \mathcal{G}(\mathcal{T}, A)\} .$$

Our goal is to construct an automaton $\overline{A}$ with $\mathcal{L}(\overline{A}) = \overline{\mathcal{L}(A)}$ that checks precisely this property.

However, the property contains an existential quantification, i.e. the automaton needs to check whether there is *some* strategy, which is a hard task. We solve this problem by considering a modified problem: We construct an automaton A' that gets as input a tree $\mathcal{T}$ **and** a position strategy $s_{\Box}$. Instead of checking whether there is *some* strategy that is winning, the automaton just has to check whether the *given* strategy is winning.

Once A' has been constructed, we can project away the strategy component of the input. The result of the projection is the desired automaton $\overline{A}$ that checks for the existence of a winning strategy. We comment on this final step later in more details. Note that handling existential quantification by first extending the input and later projecting away the extension is a standard trick in automata theory, used e.g. in the proof of Büchi's theorem.

Encoding strategies / Alphabet extension:

It remains to discuss the construction of A'. One problem is that PTAs only support trees as input. To be able to make the strategy a part of the input, we encode it in the tree.

To this end, let $D = \{0, \ldots, n-1\}$ be the set of **directions**, where $n = \max_{a \in \Sigma} \operatorname{rank}(a)$, i.e. the indices of the children that a node in a Σ-tree might have. A positional strategy for the universal player for $\mathcal{G}(\mathcal{T}, A)$ can be seen as a function

$$s_{\Box} \colon \mathcal{T} \times Q^{\leq n} \to D ,$$

since a move of the universal player essentially consists of picking a successor of the current position. Namely, the universal player picks the next node of the branch which should be a witness for the run not being accepting.

We use **currying**[1] to rewrite it as

$$s_{\Box} \colon \mathcal{T} \to \left(Q^{\leq n} \to D\right) .$$

[1] Currying, named after Haskell B. Curry, is the concept of seeing a function taking several parameters, say $f \colon A \times B \to C$, as a function taking the first parameter and returning a function that takes the remaining parameters. In the example, this would mean we define a function $f^c \colon A \to (B \to C)$ such that $f^c(a)$ is the function with $\left(f^c(a)\right)(b) = f(a, b)$. This is commonly implemented in functional programming languages, to ease notation and allow for simple use of partially evaluated functions, e.g. in Haskell.

Instead of assigning to each tuple $(v, \vec{q})$ consisting of address and state vector a child node $s_\Box(v, \vec{q})$, we assign to each address v a function $f(v) \colon Q^{\leqslant n} \to D$ such that for each state vector $\vec{q}$, $f(v)(\vec{q})$ is the selected child node.

We define $S = Q^{\leqslant n} \to D$ as the set of functions from state vectors to child nodes. As explained above, a strategy is of type $s_\Box \colon \mathcal{T} \to S$, i.e. it assigns to each address an element from S. Note that the set S is finite.

To encode strategies into trees, we will consider trees over the extended alphabet $\Sigma \times S$. This means that for each address, we have an associated element in S (in addition to the label from Σ). As explained above, such a tree can be seen as a Σ-tree extended with a strategy. Vice versa, if a Σ-tree $\mathcal{T}$ and a strategy $s_\Box$ are given, one can construct a $\Sigma \times S$-tree that is basically $\mathcal{T}$ extended by $s_\Box$.

We make this formal in the following.

15.15 Definition

Let Σ be a ranked alphabet. We define the enhanced ranked alphabet $\Sigma \times S$ with $\operatorname{rank}(a, s) = \operatorname{rank}(a)$. We define the two projections

$$\begin{aligned} \operatorname{proj}_\Sigma \;&:\; \Sigma \times S \to \Sigma \\ &\quad (a, s) \mapsto a\,, \\ \operatorname{proj}_S \;&:\; \Sigma \times S \to S \\ &\quad (a, s) \mapsto s\,, \end{aligned}$$

For a $\Sigma \times S$-tree $\mathcal{T}'$, we define $\operatorname{proj}_\Sigma(\mathcal{T}')$ to be the Σ-tree in which all labels (a, s) are replaced by $\operatorname{proj}_\Sigma(a, s) = a$.

For a $\Sigma \times S$-tree $\mathcal{T}'$, we furthermore define $s_\Box(\mathcal{T}')$, a strategy for $\Box$ defined as follows:

$$\begin{aligned} s_\Box(\mathcal{T}') \;&:\; \mathcal{T} \times Q^{\leqslant n} \to D \\ &\quad (v, \vec{q}) \mapsto (\operatorname{proj}_S \operatorname{label}_{\mathcal{T}'}(v))(\vec{q})\,. \end{aligned}$$

Proof approach (formal):

For the proof of Rabin's tree theorem, we consider the language $\mathcal{L}'$ of $\Sigma \times S$-trees $\mathcal{T}'$ trees such that the strategy-component is a winning strategy for the universal player for the acceptance game on the tree formed by the Σ-component,

$$\mathcal{L}' = \{\mathcal{T}\ \Sigma \times S\text{-tree} \mid s_\Box(\mathcal{T}') \text{ is a winning strategy for } \mathcal{G}(\operatorname{proj}_\Sigma(\mathcal{T}'), A)\}\,.$$

We proceed as follows:

1. We prove that $\mathrm{proj}_\Sigma(\mathcal{L}') = \overline{\mathcal{L}(A)}$.

2. We construct a PTA A' with $\mathcal{L}(A') = \mathcal{L}'$, proving that $\mathcal{L}'$ is regular.

3. We prove that then also $\mathrm{proj}_\Sigma(\mathcal{L}') = \overline{\mathcal{L}(A)}$ is regular.
   We finally obtain $\overline{A} = \mathrm{proj}_\Sigma(A')$ with $\mathcal{L}(\overline{A}) = \overline{\mathcal{L}(A)}$.

The first and third step are easy, the second step is the crucial part of the proof.

Step 1: Proving that $\mathbf{proj}_\Sigma(\mathcal{L}') = \overline{\mathcal{L}(A)}$

We show that projecting the strategy-component of the trees in $\mathcal{L}'$ away indeed gives us the complement of $\mathcal{L}(A)$. By $\mathrm{proj}_\Sigma(\mathcal{L}')$ we mean the set of all Σ-trees obtained by applying proj_Σ to all $\Sigma \times S$-trees in $\mathcal{L}'$, $\mathrm{proj}_\Sigma(\mathcal{L}') = \{\mathrm{proj}_\Sigma(\mathcal{T}') \mid \mathcal{T}' \in \mathcal{L}'\}$.

15.16 Lemma
$\mathrm{proj}_\Sigma(\mathcal{L}') = \overline{\mathcal{L}(A)}$.

Proof:
By Lemma 15.13, a tree $\mathcal{T}$ is not in $\mathcal{L}(A)$ if and only if the universal player has a positional strategy for the parity game $\mathcal{G}(\mathcal{T}, A)$.

For any tree $\mathcal{T}' \in \mathcal{L}'$, there is a positional winning strategy on $\mathcal{G}(\mathcal{T}, A)$, where $\mathcal{T} = \mathrm{proj}_\Sigma(\mathcal{T}')$, namely the one defined by the strategy-parts of the labels. This means $\mathrm{proj}_\Sigma(\mathcal{T}') \in \overline{\mathcal{L}(A)}$.

If a tree $\mathcal{T}$ is not in $\mathcal{L}(A)$, we can take the strategy and enhance the tree by putting the strategy on each node as a second component of the label, obtaining the $\Sigma \times S$-tree $\mathcal{T}'$. The tree $\mathcal{T}'$ is in $\mathcal{L}'$ by definition, and we have $\mathrm{proj}_\Sigma(\mathcal{T}') = \mathcal{T}$. ■

Step 2: Constructing A' with $\mathcal{L}(A') = \mathcal{L}'$, proving that $\mathcal{L}'$ is regular.

This is the difficult part of the proof. To obtain A' we proceed in several steps:

- we construct a word automaton $A_{branches}$ out of the given PTA by decomposing it into branches,

- we complement this word automaton to obtain a word automaton B,

- and we lift B to obtain again a tree automaton A'.

We start with explaining how to decompose a tree into its branches.

15.17 Remark

If we have $\operatorname{rank}(a) = 1$ for all symbols $a \in \Gamma$ of a ranked alphabet, each Γ-tree is actually an infinite word, since there is no branching.

We call a PTA over such an alphabet a **parity word automaton**, and a regular language of Γ-trees a **regular language of infinite words**, or **ω-regular language**.

15.18 Definition

Let Σ' be a ranked-alphabet, and let $D = \{0, \ldots, n-1\}$, where $n = \max_{a \in \Sigma'} \operatorname{rank}(a)$. We define the ranked alphabet $\Sigma' \times D$ with $\operatorname{rank}(a, d) = 1$.

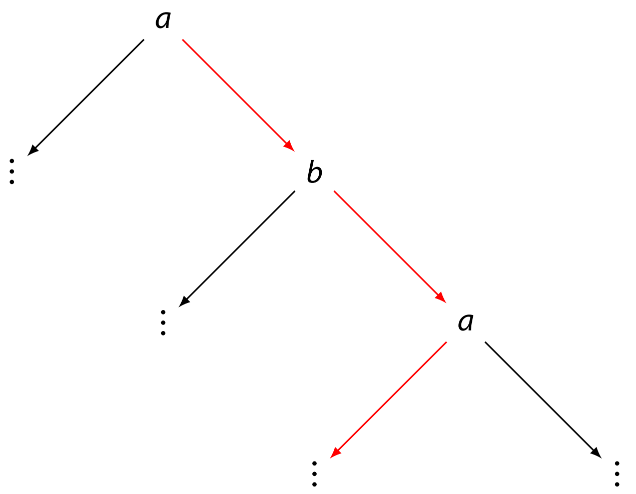
Every branch π of a Σ'-tree π can be seen as a word over $\Sigma' \times D$:

- The first component of each entry gives the label,
- the second component gives the successor that will be picked.

Given a tree $\mathcal{T}'$, we can define the word-language $\operatorname{Branches}(\mathcal{T}')$ of all its branches.

15.19 Example

Consider the ranked alphabet $\Sigma' = \{a_{/2}, b_{/2}\}$. The following figure depicts a prefix of a Σ'-tree.



The branch marked by the red arcs is represented by the following word over $\{a, b\} \times \{0, 1\}$:

$$(a, 1)(b, 1)(a, 0) \ldots \in (\Sigma's \times D)^{\omega} .$$

Note that compared to our previous definitions of branches, we have to put the label inside the representation of the branch.

In our case, the base alphabet will be $\Sigma \times S$, i.e. we consider a tree extended with a strategy. In the following, we construct an automaton $A_{branches}$ over $\Sigma \times S \times D$ that runs on branches of such an extended tree.

Let $A = (\Sigma, Q, q_0, \rightarrow, \Omega)$ be the given PTA for the language that we want to complement. We construct a parity word automaton

$$A_{branches} = (\Sigma \times S \times D, Q, q_0, \rightarrow', \Omega)$$

such that a transition (q, q') is in $\rightarrow'_{a,s,d}$ if and only if there is a transition $(q, \vec{q}) \in \rightarrow_a$ of A such that $s(\vec{q}) = d$ and $q' = \vec{q}_d$. Note that the other components Q, q, Ω coincide with those of A.

For a $\Sigma \times S$-tree $\mathcal{T}'$, this automaton checks whether the strategy-components of the labeling is **not** a winning strategy for the universal player. Consider the set Branches$(\mathcal{T}')$ of branches of $\mathcal{T}'$.

- All branches that are not selected by the strategy are rejected since there is no suitable transition for them in the transition relation $\rightarrow'$.
- A branch that is selected by the strategy is accepted if and only if there is a sequence of transitions of $A_{branches}$ leading to acceptance. To this sequence of transitions corresponds a sequence of transitions of A that will also ensure that the branch in $\text{proj}_\Sigma(\mathcal{T}')$ is accepted. This means that the branch is not a witness for the acceptance condition being violated, and thus the universal player's strategy is not winning.

We make this observation formal in the following lemma.

15.20 Lemma
Let $\mathcal{T}'$ be a $\Sigma \times S$ tree. The strategy obtained by proj_S is a winning strategy for the universal player on $\mathcal{G}(\mathcal{T}, A)$, where $\mathcal{T} = \text{proj}_\Sigma(\mathcal{T}')$, if and only if $\mathcal{L}(A_{branches}) \cap \text{Branches}(\mathcal{T}') = \emptyset$.

Proof:
Assume that the intersection $\mathcal{L}(A_{branches}) \cap \text{Branches}(\mathcal{T}')$ is non-empty. Then there is a branch π of $\mathcal{T}'$ that is accepted by $A_{branches}$, and thus there is an accepting run of $A_{branches}$ on this branch π.

Consider the play of $\mathcal{G}(\mathcal{T}, A)$ in which the universal player's moves conform to the strategy obtained by proj_S, and the existential players moves correspond to the transitions picked in the run r. Since the run is accepting, the play is won by the existential player. This proves that the universal player's strategy cannot be winning.

For the other direction, assume that $\mathcal{L}(A_{branches}) \cap \text{Branches}(\mathcal{T}') = \emptyset$. Consider a play of $\mathcal{G}(\mathcal{T}, A)$ in which the universal player's moves conform to the strategy obtained by proj_S. To the play corresponds a run of $\mathcal{L}(A_{branches})$ on the branch π that is selected by the universal player during the play: The existential player's moves in the play correspond to transitions of A and also to transitions of $A_{branches}$. Since the branch π is not in $\mathcal{L}(A_{branches})$, this run cannot be accepting, and the play is won by the universal player. ■

Using basic set theory, we can rewrite the emptiness of the intersection as an inclusion in the complement.

15.21 Corollary
Let $\mathcal{T}'$ be a tree. The strategy obtained by proj_S is a winning strategy for the universal player on $\mathcal{G}(\mathcal{T}, A)$, where $\mathcal{T} = \text{proj}_\Sigma(\mathcal{T}')$, if and only if $\text{Branches}(\mathcal{T}') \subseteq \overline{\mathcal{L}(A_{branches})}$.

Our goal is to use this corollary to prove the regularity of $\mathcal{L}'$. The problem is that it is not clear whether $\overline{\mathcal{L}(A_{branches})}$ is a regular language. In fact, proving the regularity of this lecture seems to require Rabin's tree theorem.

Luckily, $\mathcal{L}(A_{branches})$ is just a regular language of infinite **words**. These are much easier to handle than regular tree languages. We can use the following theorem without proof, which states that parity word automata are determinizable. (Note that this is not true for parity tree automata!)

15.22 Theorem: Safra [Saf88]
Let A_w be a parity word automaton. One can construct a deterministic parity word automaton A'_w with $\mathcal{L}(A_w) = \mathcal{L}(A'_w)$.

15.23 Remark: On the proof of Safra's result
The proof uses the Safra-construction, another big result from automata theory. It can be seen as an extended version of the powerset construction used to determinize finite automata. It also leads to a blow-up in the number of states. If the original automaton had k states, its determinization has up to $2^{\mathcal{O}(k \cdot \log k)}$ states.

15.24 Remark
By deterministic, we mean that for each state q and each symbol $a \in \Sigma'$, there is a unique transition $(q, q') \in \rightarrow_a$.

A deterministic automaton has a unique run on each infinite word. Whether the word is in the language of the automaton depends on whether this run is accepting.

Using the theorem, we get that regular languages of infinite words are closed under complement. We can use the same trick as for the complementation of NFAs: We invert the final states. Technically, this means we manipulate the priority assignment.

15.25 Corollary
Let A_w be a parity word automaton. One can construct a deterministic parity word automaton $\overline{A'_w}$ with $\mathcal{L}(\overline{A'_w}) = \overline{\mathcal{L}(A_w)}$.

Proof:
Using Theorem 15.22, we can construct the deterministic parity word automaton $A'_w = (\Gamma, Q', q'_0, \rightarrow', \Omega')$. We define $\overline{A_w} = (\Gamma, Q', q'_0, \rightarrow', \Omega'')$ with $\Omega''(q) = \Omega'(q) + 1$. Note that the unique run of $\overline{A_w}$ on a word π is accepting if and only if the unique run of A'_w on the word was non-accepting. We obtain $\mathcal{L}(\overline{A'_w}) = \overline{\mathcal{L}(A'_w)} = \overline{\mathcal{L}(A_w)}$. ■

This allows us to construct a deterministic parity word automaton accepting $\overline{\mathcal{L}(A_{branches})}$. Let $B = (\Sigma \times S \times D, Q^B, q_0^B, \rightarrow^B, \Omega^B)$ be this automaton. We will use it to construct a parity tree automaton for $\mathcal{L}'$.

15.26 Proposition
$\mathcal{L}'$ is regular.

Proof:
We define the parity tree automaton

$$A' = (\Sigma \times S, Q^B, q_0^B, \rightarrow', \Omega^B)$$

where the transition relation $\rightarrow'$ is defined as follows: A transition $(q, \vec{q})$ is in $\rightarrow'_{a,s}$ if and only if for each $d \in D$, we have that $(q, \vec{q}_d) \in \rightarrow^B_{a,s,d}$ is the unique transition of B for the source state q and the symbol (a, s, d). Note that the other components Q, q, Ω coincide with those of B.

In a run of A' on a tree $\mathcal{T}'$, it essentially simulates B along each branch of the tree.

It remains to argue that A' indeed accepts $\mathcal{L}'$.

By the construction of A', a tree $\mathcal{T}'$ is accepted by A' if and only if B accepts each of its branches $\pi \in \text{Branches}(\mathcal{T}')$. Since $\mathcal{L}(B) = \overline{\mathcal{L}(A_{branches})}$, this means that $\text{Branches}(\mathcal{T}') \subseteq \overline{\mathcal{L}(A_{branches})}$. By Corollary 15.21, this is the case if and only if the strategy obtained by proj_S is a winning strategy for the universal player on $\mathcal{G}(\mathcal{T}, A)$, where $\mathcal{T} = \text{proj}_\Sigma(\mathcal{T}')$. This was the condition for being in $\mathcal{L}'$. ■

Step 3: Conclude that $\mathrm{proj}_\Sigma(\mathcal{L}') = \overline{\mathcal{L}(A)}$ is regular

We can see the projection as a special case of the more general concept of rank-preserving functions.

15.27 Definition
Let Σ_1, Σ_2 be ranked alphabets. We call a function $f\colon \Sigma_1 \to \Sigma_2$ **rank-preserving** if for all $a \in \Sigma_1$, we have $\mathrm{rank}(f(a)) = \mathrm{rank}(a)$.

Given a Σ_1-tree $\mathcal{T}$, we define $f(\mathcal{T})$ to be the Σ_2 tree in which all labels $a \in \Sigma_1$ are replaced by $f(a) \in \Sigma_2$. Note that since f is rank-preserving, $f(\mathcal{T})$ is indeed a valid Σ_2-tree.

Regular languages of infinite trees are effectively closed under rank-preserving functions.

15.28 Lemma
Let $\mathcal{L}$ be a regular language of Σ_1-trees, let $f\colon \Sigma_1 \to \Sigma_2$ be rank-preserving. Then

$$f(\mathcal{L}) = \{f(\mathcal{T}) \mid \mathcal{T} \in \mathcal{L}\}$$

is a regular language of Σ_2-trees.

Proof: Exercise 15.44, Part c). ■

For the desired statement to follow, it remains to observe that the projection onto Σ is indeed rank-preserving.

15.29 Lemma
The projection $\mathrm{proj}_\Sigma\colon \Sigma \times S \to \Sigma$ is rank-preserving.

Proof:
By definition, we have $\mathrm{rank}(a, s) = \mathrm{rank}(a)$. ■

Finally, we are able to compose our results into a proof of Rabin's tree theorem.

Proof of Theorem 15.10:
The language $\mathcal{L}'$ is regular by Proposition 15.26. Furthermore, we have $\mathrm{proj}_\Sigma(\mathcal{L}') = \overline{\mathcal{L}(A)}$ by Lemma 15.16.

The projection is rank-preserving by Lemma 15.29. Thus, $\overline{\mathcal{L}(A)}$ is regular by Lemma 15.28.

Note that the Proposition 15.26 and Lemma 15.28 can be strengthened to effectively return the desired automata. This proves the second part of Rabin's tree theorem. Note that for the construction, we need Safra's construction (Theorem 15.22) which we have not explained. ■

15.30 Remark

The PTA A' constructed in the proof of Proposition 15.26 is deterministic, since the parity word automaton B was deterministic. This means we can represent $\mathcal{L}'$ using a deterministic PTA.

At first glance, this seems to violate our result that deterministic PTAs are strictly less expressive than non-deterministic ones (Exercise 15.43). This contradiction is resolved by looking in detail at the alphabet over which $\mathcal{L}'$ and A' are defined: It is the enhanced alphabet $\Sigma \times S$.

If we project the strategy-component away to obtain the automaton $\overline{A}$ for $\overline{\mathcal{L}(A)}$, we may obtain a non-determinism automaton: There might be two letters (a, s) and (a, s') for which the Σ-component is the same, but the strategy-component differs. For each source state q, automaton A' will have a unique transition for each of them. The automaton $\overline{A}$ cannot distinguish these letters, it will have (at least) two transitions for the letter a.

This means that enhancing the tree by the strategy did not only make our theory work, it also allows the language to be recognizable by a deterministic PTA.

To conclude this section, we want to check the emptiness of PTA languages via parity games. Given a PTA A, we want to decide whether $\mathcal{L}(A) = \emptyset$ holds, i.e. whether A is actually the finite representation for a set consisting of at least one tree.

To do so, we construct a finite parity game $\mathcal{G}(A)$. The idea is to drop the $\mathcal{T}$ from the positions in $\mathcal{G}(\mathcal{T}, A)$. The $\mathcal{T}$ component was used to force the existential player to respect the labeling of the given tree. Now we are interested in whether *there is* a tree. To model this, we allow the existential player to pick an arbitrary transition, without having to respect the label of the automaton. This means that during a play, the existential player can construct the tree as she likes.

Formally, we define the game $\mathcal{G}(A)$ as follows:

- $V_{\bigcirc} = Q$,
- $V_{\Box} = Q^{\leq n}$, where $n = \max_{a \in \Sigma} \operatorname{rank}(a)$,
- $V = V_{\bigcirc} \uplus V_{\Box}$,

- $$\begin{aligned} R \;=\; & \{((q, \vec{q})) \mid q \in V_{\bigcirc}, \exists a \exists (q, \vec{q}) \in \rightarrow_a\} \\ \uplus\; & \{((\vec{q}, \vec{q}_i)) \mid \vec{q} \in Q^k \subseteq V_{\Box}, i \in \{0, \ldots, k-1\}\}\,. \end{aligned}$$
- $\Omega(q) = \Omega_A(q)\,,$
  $\Omega(\vec{q}) = 0\,.$

15.31 Proposition

The language of A is non-empty if and only if the existential player wins the parity game $\mathcal{G}(A)$ from position q_0,

$$\mathcal{L}(A) \neq \varnothing \quad \text{iff} \quad q_0 \in W_{\bigcirc}^{\mathcal{G}(A)}\,.$$

The proof is an easy extension of the proof of Lemma 15.13.

Note that – in contrast to the game $\mathcal{G}(\mathcal{T}, A)$ – the game arena of $\mathcal{G}(A)$ is finite. Thus, Zielonka's recursive algorithm can be used to actually solve it.

If $\mathcal{L}(A) \neq \varnothing$, then the positional winning strategy for the existential player provides a finite description of a tree in $\mathcal{L}(A)$.

Monadic second-order logic over infinite binary trees / S2S

The importance of Rabin's tree theorem comes from the relation of parity tree automata to a certain kind of logic. Parity tree automata are equivalent to formulas in **monadic second-order (MSO) logic over infinite trees**. Rabin's tree theorem is crucial for proving the equivalence. The translation together with the decision procedure for language emptiness means that the satisfiability of formulas in monadic second order logic over infinite trees is decidable. The latter result is also sometimes called Rabin's tree theorem.

Without loss of generality, we will assume that all symbols in the alphabet have rank exactly 2. For this reason, monadic second order logic over infinite trees is also called **S2S**, **s**econd-order logic with **2 s**uccessors.

The following diagram depicts the translation and the algorithmic problems on each side.

$$\begin{tikzcd}
\text{PTA } A \arrow[r, leftrightarrow] \arrow[d, dashed, no head] & \text{S2S-formula } \varphi \arrow[d, dashed, no head] \\
\begin{array}{c}\text{Emptiness} \\ \mathcal{L}(A) = \varnothing?\end{array} & \begin{array}{c}\text{Satisfiability} \\ \exists \mathcal{T} \colon \mathcal{S}(\mathcal{T}) \models \varphi?\end{array}
\end{tikzcd}$$

15.32 Remark

There is also monadic second order logic over infinite words. It is also called S1S, second order logic with one successor. It corresponds to parity word automata.

In second-order logic, there are two types of variables:

- **First-order variables**, usually denoted by lowercase letters x, y, z, represent values, i.e. in our case nodes of a tree.
- **Second-order variables**, usually denoted by uppercase letters X, Y, Z, represent sets of values, i.e. in our case sets of nodes of a tree.

The logic is called **monadic**, because second-order variables denote sets of values. In polyadic logic, they can denote sets of tuples of values.

We will now formally introduce the syntax and semantics of S2S. We first introduce the syntax, e.g. terms and formulas. We give a brief explanation of the meaning of the syntax in *italic*. This should not be seen as a formal definition of the semantics.

15.33 Definition: Syntax of S2S

A (first-order) **term** s of S2S *representing a node of a tree* is

- either the symbol ε *representing the root node of the tree*
- or a first-order variable x *representing the node to which x is mapped in the assignment under consideration*.

An **atomic formula** of S2S is, for terms s, s',

- $s = s'$ *expressing that s and s' should represent the same node*,
- $s \sqsubseteq s'$ *expressing that s should represent an (indirect) ancestor of the node represented by s'*,
- $P_a(s)$ for each symbol $a_{/2} \in \Sigma$ *expressing that the node represented by s is labeled by a*,
- $S_i(s, s')$ for $i \in \{0, 1\}$ *expressing that s' represents the left (if i* $= 0$*) resp. right (if i* $= 1$*) successor of s*,
- $s \in X$ for a second-order variable X *expressing that the set of nodes represented by X contains the node represented by s*.

The **formulas** of S2S are defined inductively as follows:

- Every atomic formula is a formula.

- If φ, φ' are formulas, then the following are formulas

$$\neg\varphi \qquad \varphi \wedge \varphi' \qquad \varphi \vee \varphi' \qquad \exists x\colon \varphi \qquad \forall x\colon \varphi \qquad \exists X\colon \varphi \qquad \forall X\colon \varphi\ .$$

A variable is **free** in a formula if it is not bound by a preceding quantifier. We call a formula φ **closed** if it has no free variables.

15.34 Remark

The syntax of S2S is the syntax of monadic second-order predicate logic with equality over the following signature:

- $\varepsilon_{/0}$ is the only function symbol and constant (arity 0).
- The predicates are the binary predicates $\sqsubseteq_{/2}, S_{0/2}, S_{1/2}$ and for each symbol $a \in \Sigma$ the unary predicate $P_{a/1}$.

Note that since the logic is monadic, all function symbols and predicates take first-order terms as their parameters. The membership predicate $x \in X$ (that is not explicitly given in the signature) is the only way to involve second-order variables.

To evaluate S2S – like any other kind of predicate logic – we need a structure. A structure consists of a set of data values and an interpretation of the function and predicate symbols. We are interested in structures that are given by Σ-trees.

15.35 Definition

Let $\mathcal{T}$ be a Sigma-tree. Then we define $\mathcal{S}(\mathcal{T})$ to be the structure where

- the data values are $\mathcal{T}$, i.e. the nodes (addresses) of the tree are the data values. First-order terms represent nodes, second-order variables represent sets of nodes.
- We have $\varepsilon^{\mathcal{T}} = \varepsilon \in \mathcal{T}$, i.e. ε is indeed interpreted to denote the root node.
- For two nodes v, v', we have
$$v \sqsubseteq^{\mathcal{T}} v' = \text{true}$$
iff v is a prefix of v', i.e. v is an ancestor of v'.
- For two nodes v, v' and $i \in \{0, 1\}$, we have
$$S_i^{\mathcal{T}}(v, v') = \text{true}$$
iff $v' = v.i$, i.e. v' is the left ($i = 0$) resp. right ($i = 1$) successor of v in the tree.

- For a node v and $a \in \Sigma$, we have

  $$P_a^{\mathcal{T}}(v) = \text{true}$$

  iff label(v) = a, i.e. v is labeled by a.

To evaluate a formula, a structure $\mathcal{S}(\mathcal{T})$ is not sufficient, we also need an interpretation

$$\mathcal{I}_{\mathcal{T}}\colon (\text{First-order Variables} \to \mathcal{T}) \cup (\text{Second-Order Variables} \to \mathcal{P}(T)),$$

also called valuation or assignment. This function maps each free first-order variable x to a node $\mathcal{I}_{\mathcal{T}}(x)$ in $\mathcal{T}$, and each second-order variable X to a set of nodes $\mathcal{I}_{\mathcal{T}}(X) \subseteq \mathcal{T}$.

The evaluation of formulas can then be defined by structural induction. Even when the formula under consideration is closed, we will need an interpretation during the induction after the quantifiers have been resolved.

15.36 Definition: Semantics of S2S
Let $\mathcal{S}(\mathcal{T})$ be a structure and let $\mathcal{I}_{\mathcal{T}}$ be a corresponding interpretation.

For a term s, let

$$\mathcal{I}(s) = \begin{cases} \mathcal{I}(x) \in \mathcal{T} & s = x \text{ Variable} \\ \varepsilon^{\mathcal{T}} = \varepsilon \in \mathcal{T} & s = \varepsilon \end{cases}$$

be the node of $\mathcal{T}$ represented by s.

Then we can inductively define the **models** or **satisfies relation** for formulas.

$$\begin{array}{ll}
\mathcal{S}(\mathcal{T}), \mathcal{I} \models s = s' & \text{if } \mathcal{I}(s) = \mathcal{I}(s')\,, \\
\mathcal{S}(\mathcal{T}), \mathcal{I} \models s \sqsubseteq s' & \text{if } \mathcal{I}(s) \sqsubseteq^{\mathcal{T}} \mathcal{I}(s')\,, \\
\mathcal{S}(\mathcal{T}), \mathcal{I} \models S_i(s, s') & \text{if } S_i^{\mathcal{T}}(\mathcal{I}(s), \mathcal{I}(s'))\,, \\
\mathcal{S}(\mathcal{T}), \mathcal{I} \models P_a(s) & \text{if } P_a^{\mathcal{T}}(\mathcal{I}(s))\,, \\
\mathcal{S}(\mathcal{T}), \mathcal{I} \models s \in X & \text{if } \mathcal{I}(s) \in \mathcal{I}(X)\,, \\
\mathcal{S}(\mathcal{T}), \mathcal{I} \models \neg\varphi & \text{if } \mathcal{S}(\mathcal{T}), \mathcal{I} \not\models \varphi\,, \\
\mathcal{S}(\mathcal{T}), \mathcal{I} \models \varphi \wedge \varphi' & \text{if } \mathcal{S}(\mathcal{T}), \mathcal{I} \models \varphi \text{ and } \mathcal{S}(\mathcal{T}), \mathcal{I} \models \varphi'\,, \\
\mathcal{S}(\mathcal{T}), \mathcal{I} \models \varphi \vee \varphi' & \text{if } \mathcal{S}(\mathcal{T}), \mathcal{I} \models \varphi \text{ or } \mathcal{S}(\mathcal{T}), \mathcal{I} \models \varphi'\,, \\
\mathcal{S}(\mathcal{T}), \mathcal{I} \models \exists x\colon \varphi & \text{if there is } v \in \mathcal{T} \text{ such that } \mathcal{S}(\mathcal{T}), \mathcal{I}[x \mapsto v] \models \varphi\,, \\
\mathcal{S}(\mathcal{T}), \mathcal{I} \models \forall x\colon \varphi & \text{if for all } v \in \mathcal{T} \text{ we have } \mathcal{S}(\mathcal{T}), \mathcal{I}[x \mapsto v] \models \varphi\,, \\
\mathcal{S}(\mathcal{T}), \mathcal{I} \models \exists X\colon \varphi & \text{if there is } V \subseteq \mathcal{T} \text{ such that } \mathcal{S}(\mathcal{T}), \mathcal{I}[X \mapsto V] \models \varphi\,, \\
\mathcal{S}(\mathcal{T}), \mathcal{I} \models \forall X\colon \varphi & \text{if for all } V \subseteq \mathcal{T} \text{ we have } \mathcal{S}(\mathcal{T}), \mathcal{I}[X \mapsto V] \models \varphi\,.
\end{array}$$

If φ is closed, the initial interpretation does not matter. If $\mathcal{S}(\mathcal{T})$ together with any interpretation satisfies φ, we write $\mathcal{S}(\mathcal{T}) \vDash \varphi$ and say that $\mathcal{T}$ satisfies φ or that $\mathcal{T}$ is a **model** for φ.

15.37 Example

Consider the following closed formula.

$$\varphi \equiv \exists X\colon \varepsilon \in X \wedge \forall x\colon x \in X \rightarrow (P_a(x) \wedge \exists y\colon y \in X \wedge (S_0(x,y) \vee S_1(x,y)))$$

A tree is a model for φ if and only if it contains at least one branch labeled only by *a*s.

15.38 Remark

ε and $\sqsubseteq$ are *syntactic sugar*, the other predicates are powerful enough to express them.

a) The formula

$$\mathrm{root}(x) = \forall y\colon \neg S_0(y,x) \wedge \neg S_1(y,x)$$

is satisfied by $\mathcal{S}(\mathcal{T}), \mathcal{I}$ if and only if $\mathcal{I}(x)$ is the root of $\mathcal{T}$. It expresses that x has no predecessor (and the root is the only node with this property).

b) The formula

$$\mathrm{ancestor}(x,y) = \forall X\colon (x \in X \wedge \forall y\colon y \in X \rightarrow \forall z\colon (S_0(y,z) \vee S_1(y,z)) \rightarrow z \in X) \rightarrow y \in X$$

is satisfied by $\mathcal{S}(\mathcal{T}), \mathcal{I}$ if and only if $\mathcal{I}(x)$ is a prefix of $\mathcal{I}(y)$. It expresses that every set that contains x and is closed under taking the successors also has to contain y. Since this then holds for the smallest such set, that is the set of indirect successors of x, we have that x is an ancestor of y.

One can introduce more syntactic sugar, e.g. one usually writes $s \neq s'$ and $s \notin X$ instead of $\neg(s = s')$ and $\neg(s \in X)$. The other Boolean operators like $\leftrightarrow$ (equivalence) and $\oplus$ (XOR) can be expressed using conjunction, negation and disjunction.

The crucial algorithmic problem is (as for many other kinds of logic) satisfiability.

S2S-Satisfiability

Given: A closed S2S-formula φ

Question: Is there a tree $\mathcal{T}$ with $\mathcal{S}(\mathcal{T}) \vDash \varphi$?

15.39 Theorem: Rabin's tree theorem

S2S-Satisfiability is decidable.

The proof works as follows: We can translate a given formula φ into an equivalent PTA A_φ, and check language emptiness for A_φ using Proposition 15.31. The translation is the following theorem.

15.40 Theorem: Rabin's tree theorem
S2S-formulas and PTAs are equivalent:

a) For a given closed PTA A we can effectively construct a closed S2S-formula φ_A such that a tree models φ_A if and only if it is accepted by A_φ.

$$\mathcal{L}(A) = \{\mathcal{T} \mid \mathcal{S}(\mathcal{T}) \vDash \varphi_A\} .$$

b) For a given closed S2S-formula φ we can effectively construct a PTA A_φ such that a tree $\mathcal{T}$ is accepted by A_φ if and only if it is as a model for φ,

$$\mathcal{L}(A_\varphi) = \{\mathcal{T} \mid \mathcal{S}(\mathcal{T}) \vDash \varphi\} .$$

Sketch of the Proof:

a) For a given automaton A, it is not too hard to construct a formula φ_A expressing that A has an accepting run on a tree.

b) Given a formula, we need to construct a tree automaton.

  - For the atomic formulas one can directly create PTAs.
  - Negation, conjunction and disjunction are imitated by the corresponding operations complementation, union and intersection on PTA languages. (This is where the first formulation of Rabin's tree theorem comes into play.)
  - Dealing with variables requires a trick. (This was known before in the literature from Büchi's theorem on the equivalence of WMSO-definable languages and the regular languages of finite words).

    Let us assume that there are only existential quantifiers. This can be enforced by rewriting $\forall x\colon \varphi$ as $\neg\exists x\colon \neg\varphi$ (and similar for second-order variables).

    To deal with variables, one enhances the alphabet. Instead of using Σ as the alphabet, we use $\Sigma \times \mathbb{B}^{V_1} \times \mathbb{B}^{V_2}$ where V_1, V_2 are the sets of free first- resp. second-order variables. This means each position is labeled not only by a symbol in Σ, but also by vectors of Boolean values denoting which variables are represented by the position. For a position v labeled by $(a, \vec{x}, \vec{X})$, we have $\mathcal{I}(y) = v$ if $\vec{x}_y = 1$, and similarly $v \in \mathcal{I}(Y)$ if $\vec{X}_Y = 1$.

Whenever we have an existential quantifier binding a variable, we project away the corresponding component of the vector. For first order variables y, we have to enforce that there is a unique position with $\vec{x}_y = 1$. This can be done by intersecting with a suitable PTA language.

Since the original formula was closed, all additional components will be projected away during the inductive construction. The final automaton will be an automaton just over the alphabet Σ.

■

15.41 Corollary

The class of S2S-definable languages, i.e. the class of languages

$$\mathcal{L}(\varphi) = \{\mathcal{T} \mid \mathcal{S}(\mathcal{T}) \vDash \varphi\}$$

where φ is a closed S2S-formula, is exactly the class of regular languages of infinite trees.

15.42 Example

a) The language of A_1 from Exercise 15.43 can be expressed by the following S2S-formula:

$$P_a(\varepsilon) \wedge \forall x \forall y \forall z\colon (S_0(x,y) \wedge S_1(x,z)) \to ((P_a(x) \to P_b(y) \wedge P_b(z)) \wedge (P_b(x) \to P_a(y) \wedge P_a(z)))\,.$$

b) The language of A_2 from Exercise 15.43 can be easily expressed by the following S2S-formula:

$$\exists x\colon P_a(x) \wedge \forall y\colon x \neq y \to P_b(y)\,.$$

c) Expressing the language of A_3 is more complicated, see Exercise 15.48.

Exercises

15.43 Exercise

Consider the ranked alphabet $\Sigma = \{a_{/2}, b_{/2}\}$. Note that Σ-trees are so-called *full* infinite binary trees.

a) Consider the PTA $A_1 = (\Sigma, \{q_0, q_1\}, q_0, \rightarrow, \Omega)$ with

$$\begin{aligned} \rightarrow_a &= \{(q_0, (q_1, q_1))\} , \\ \rightarrow_b &= \{(q_1, (q_0, q_0))\} , \\ \Omega(q_0) &= \Omega(q_1) = 0 . \end{aligned}$$

Describe its language $\mathcal{L}(A_1)$.

b) Consider the PTA $A_2 = (\Sigma, \{q_+, q_-\}, q_+, \rightarrow, \Omega)$ with

$$\begin{aligned} \rightarrow_a &= \{(q_+, (q_-, q_-))\} , \\ \rightarrow_b &= \{(q_+, (q_+, q_-)),\ (q_+, (q_-, q_+)),\ (q_-, (q_-, q_-))\} , \\ \Omega(q_-) = 0 , &\qquad \Omega(q_+) = 1 . \end{aligned}$$

Formally prove that $\mathcal{L}(A_2)$ is exactly the set of Σ-trees in which exactly one node is labeled by a.

Remark: A_2 is non-deterministic, and one can prove that there is no deterministic PTA A accepting the same language.

c) Present a PTA A_3 whose language is the set of Σ-trees in which exactly one branch contains infinitely many nodes labeled by a.

Argue that your automaton indeed has this property.

15.44 Exercise: Closure properties of regular languages of infinite trees
Prove that regular languages of infinite trees are closed under union, intersection, and projection.

Let $A = (\Sigma, Q, q_0, \rightarrow, \Omega)$, $A' = (\Sigma, Q', q_0', \rightarrow', \Omega')$ be PTAs over the same ranked alphabet Σ.

a) Show how to construct a PTA $A_\cup$ with $\mathcal{L}(A_\cup) = \mathcal{L}(A) \cup \mathcal{L}(A')$.

b) Show how to construct a PTA $A_\cap$ with $\mathcal{L}(A_\cap) = \mathcal{L}(A) \cap \mathcal{L}(A')$.

Hint: Use Rabin's tree theorem.

c) Let Σ' be a ranked alphabet, and $f\colon \Sigma \rightarrow \Sigma'$ be a **rank preserving function**, i.e. we have $\text{rank}_\Sigma(a) = \text{rank}_{\Sigma'}(f(a))$ for all $a \in \Sigma$. For a Σ-tree $\mathcal{T}$, we define $f(\mathcal{T})$ to be the Σ'-tree in which the label a of each node is replaced by $f(a)$. Note that the fact that f is rank-preserving is crucial for $f(\mathcal{T})$ being a Σ'-tree.

For a language of Σ-trees $\mathcal{L}$, we define

$$f(\mathcal{L}) = \{f(\mathcal{T}) \mid \mathcal{T} \in \mathcal{L}\} .$$

Show how to construct a PTA $A_f = (\Sigma', Q_f, q_{0f}, \rightarrow_f, \Omega_f)$ with $\mathcal{L}(A_f) = f(\mathcal{L}(A))$.

15.45 Exercise

Let $\mathcal{T}$ be a Σ-tree and let A be a PTA. Consider the parity game $\mathcal{G}(\mathcal{T}, A)$ as defined in Definition 15.12.

a) The game arena of $\mathcal{G}(\mathcal{T}, A)$ is not necessarily deadlock-free.

   In which case can deadlocks occur?

   Modify the game arena such that it becomes deadlock free such that the validity of Lemma 15.13 is preserved.

   How can one modify the automaton without changing its language such that $\mathcal{G}(\mathcal{T}, A)$ is deadlock-free without modification?

b) Assume that the existential player has a positional winning strategy $s_\bigcirc$ from position (ε, q_0) in $\mathcal{G}(\mathcal{T}, A)$.

   Present an accepting run of A on $\mathcal{T}$.

   Hint: Construct the run inductively, guided by $s_\bigcirc$.

c) Assume that the universal player has a positional winning strategy $s_\Box$ from position (ε, q_0) in $\mathcal{G}(\mathcal{T}, A)$.

   For each candidate run of A on $\mathcal{T}$, identify a branch on which the acceptance condition is violated.

15.46 Exercise

In this exercise, we want to apply Rabin's tree theorem to the automaton A_1 from Part a) of Exercise 15.43.

a) Construct the set $S = Q^{\leqslant n} \rightarrow D$.

   Hint: To avoid the following construction becoming excessively large, restrict the domain to vectors of states that can actually occur.

b) Construct the parity word automaton $A_{branches}$.

c) Make $A_{branches}$ deterministic by adding an error-state and the corresponding transitions. (For each symbol a, s, d, and each state q, there needs to be exactly one transition $(q, q') \in \rightarrow_{a,s,d}$.) Complement $A_{branches}$ to obtain the automaton B with $\mathcal{L}(B) = \overline{\mathcal{L}(A_{branches})}$.

d) Construct the parity tree automaton A' for $\mathcal{L}'$ that simulates B on all branches of a tree.

e) Project A' to Σ to obtain the automaton $\overline{A_1}$. Check that $\mathcal{L}(\overline{A_1}) = \overline{\mathcal{L}(A_1)}$ indeed holds by describing the language of $\overline{A_1}$.

15.47 Exercise

a) Let A be a PTA, and assume that the existential player wins the parity game $\mathcal{G}(A)$ from the initial position q_0.

Explain how a winning strategy for the existential player can be used to define a tree in $\mathcal{T} \in \mathcal{L}(A)$. Make this formal by explaining the construction of the set of nodes $\mathcal{T}$ and its labeling function $\text{label}_{\mathcal{T}}$.

b) Consider automaton A_2 from Part b) of Exercise 15.43. Transform the automaton to a language-equivalent automaton that has at least one transition $(q, \vec{q}) \in \rightarrow_a$ for each source state q and symbol a. (This will ensure that the parity game is deadlock-free.)

Construct the parity game $\mathcal{G}(A)$ and identify a positional winning strategy for the existential player. How does the tree described by the strategy look like?

Hint: Restrict yourself to positions Q^2 of the universal player that can actually occur during a play of the game. This prevents the game arena from becoming excessively large.

15.48 Exercise

Consider the Alphabet $\Sigma = \{a_{/2}, b_{/2}\}$. Our goal is to create a closed S2S-formula for the language $\mathcal{L}$ of trees in which exactly one branch contains infinitely many as (known from Part c) of Exercise 15.43).

a) Consider the following S2S formula that has the free second-order variable X.

$$\begin{aligned}
\text{Branch}(X) \quad = \quad & \varepsilon \in X && (1)\\
\wedge \quad & \forall x\colon x \in X \rightarrow \exists y \exists z\colon S_0(x,y) \wedge S_1(x,z) \wedge (y \in X \oplus z \in X) && (2)\\
\wedge \quad & \forall y\colon (y \in X \wedge y \neq \varepsilon) \rightarrow \exists x\colon x \in X \wedge (S_0(x,y) \vee S_1(x,y)) && (3)
\end{aligned}$$

Here, ⊕ is XOR and → is implication. They can be easily rewritten using negation, conjunction, and disjunction.

Argue that Branch(X) evaluates to true under a structure $\mathcal{S}(\mathcal{T})$ and an interpretation $\mathcal{I}_\mathcal{T}$ if and only if $\mathcal{I}_\mathcal{T}(X)$ is a set of positions that forms a branch of $\mathcal{T}$. Explain the purpose of each Line (1) - (3).

b) In S2S, we only have an equality predicate for first-order terms. Construct a formula Equal(X, Y) with two free second-order variables X, Y that evaluates to true under a structure $\mathcal{S}(\mathcal{T})$ and an interpretation $\mathcal{I}_\mathcal{T}$ if and only if $\mathcal{I}_\mathcal{T}(X) = \mathcal{I}_\mathcal{T}(Y)$.

c) Construct formulas $\text{Fin}_a(X)$ respectively $\text{Inf}_a(X)$ with one free second-order variable X that evaluate to true under a structure $\mathcal{S}(\mathcal{T})$ and an interpretation $\mathcal{I}_\mathcal{T}$ if and only if $\mathcal{I}_\mathcal{T}(X)$ contains only finitely many respectively infinite many nodes labeled by a.

For simplicity, you may suppose that $\mathcal{I}_\mathcal{T}(X)$ is a branch of $\mathcal{T}$.

d) Combine the previous parts of this exercises to construct a closed S2S-formula $\varphi_\mathcal{L}$ that evaluates to true under a structure $\mathcal{S}(\mathcal{T})$ if and only if $\mathcal{T} \in \mathcal{L}$.

References

[Bou01] C. L. Bouton. *Nim, A Game with a Complete Mathematical Theory*. In: Annals of Mathematics 3.1/4 (1901), pp. 35–39.

[Cac02] T. Cachat. *Symbolic Strategy Synthesis for Games on Pushdown Graphs*. In: ICALP. Vol. 2380. LNCS. Springer, 2002, pp. 704–715.

[Cal+17] C. S. Calude, S. Jain, B. Khoussainov, W. Li, and F. Stephan. *Deciding parity games in quasipolynomial time*. In: STOC. 2017, pp. 252–263.

[CHP08] K. Chatterjee, T. A. Henzinger, and N. Piterman. *Algorithms for Büchi Games*. In: arXiv preprint arXiv:0805.2620 (2008).

[CW07] T. Cachat and I. Walukiewicz. *The Complexity of Games on Higher Order Pushdown Automata*. In: CoRR abs/0705.0262 (2007).

[DJW97] S. Dziembowski, M. Jurdzinski, and I. Walukiewicz. *How much memory is needed to win infinite games?* In: LICS. 1997, pp. 99–110.

[EJ91] E. A. Emerson and C. S. Jutla. *Tree Automata, Mu-Calculus and Determinacy (Extended Abstract)*. In: FOCS. 1991, pp. 368–377.

[ELT16] M. Englert, R. Lazić, and P. Totzke. *Reachability in Two-Dimensional Unary Vector Addition Systems with States is NL-Complete*. In: LICS. ACM, 2016, pp. 477–484.

[EM79] A. Ehrenfeucht and J. Mycielski. *Positional strategies for mean payoff games*. In: International Journal of Game Theory 8.2 (1979), pp. 109–113.

[FGB10] N. Fisher, J. Goossens, and S. Baruah. *Optimal Online Multiprocessor Scheduling of Sporadic Real-time Tasks is Impossible*. In: Real-Time Syst. 45.1-2 (2010), pp. 26–71.

[FZ12] W. Fridman and M. Zimmermann. *Playing Pushdown Parity Games in a Hurry*. In: GandALF. 2012, pp. 183–196.

[Gee+18] G. Geeraerts, J. Goossens, T.-V.-A. Nguyen, and A. Stainer. *Synthesising Succinct Strategies in Safety Games with an Application to Real-time Scheduling*. In: Theor. Comput. Sci. 735 (2018), pp. 24–49.

[GGN17] G. Geeraerts, J. Goossens, and T.-V.-A. Nguyen. *A Backward Algorithm for the Multiprocessor Online Feasibility of Sporadic Tasks*. In: ACSD. 2017, pp. 116–125.

[GH82] Y. Gurevich and L. Harrington. *Trees, Automata, and Games*. In: STOC. 1982, pp. 60–65.

[Hag+18] M. Hague, R. Meyer, S. Muskalla, and M. Zimmermann. *Parity to Safety in Polynomial Time for Pushdown and Collapsible Pushdown Systems*. In: CoRR abs/1805.02963 (2018). Accepted for publication at CONCUR 2018. arXiv: 1805.02963.

[HL11] M. Hofmann and M. Lange. *Automatentheorie und Logik*. Springer, 2011.

[HMM16] L. Holik, R. Meyer, and S. Muskalla. *Summaries for Context-Free Games*. In: FSTTCS. Vol. 65. LIPIcs. Schloss Dagstuhl, 2016, 41:1–41:16.

[HMM17] M. Hague, R. Meyer, and S. Muskalla. *Domains for Higher-Order Games*. In: MFCS. 2017, 59:1–59:15.

[JL17] M. Jurdzinski and R. Lazic. *Succinct progress measures for solving parity games*. In: LICS. 2017, pp. 1–9.

[Joba] B. Jobstmann. *Büchi and co-Büchi Generalized Reactivity-1 Games*. http://www-verimag.imag.fr/~jobstman/teaching/Dagstuhl2011/4_buchi.pdf.

[Jobb] B. Jobstmann. *Reachability and Büchi Games*. http://richmodels.epfl.ch/_media/w2_wed_3.pdf.

[Kar78] R. M. Karp. *A characterization of the minimum cycle mean in a digraph*. In: Discrete Mathematics 23.3 (1978), pp. 309–311.

[Kho] Y. Khomskii. *Infinite Games - Summer course at the University of Sofia, Bulgaria*. https://www.math.uni-hamburg.de/home/khomskii/infinitegames2010/InfiniteGamesSofia.pdf.

[KN01] B. Khoussainov and A. Nerode. *Automata Theory and its Applications*. Vol. 21. Progress in Computer Science and Applied Logic. Springer, 2001.

[KO09] N. Kobayashi and C. L. Ong. *A Type System Equivalent to the Modal Mu-Calculus Model Checking of Higher-Order Recursion Schemes*. In: LICS. 2009, pp. 179–188.

[Kum] K. N. Kumar. *Notes on Automata, Logic, Games and Algebra - Lecture 9: Büchi Games over Infinite Graphs*. http://www.cmi.ac.in/~kumar/words/lecture09.pdf.

[Lip76] R. J. Lipton. *The Reachability Problem Requires Exponential Space*. Tech. rep. Yale University, Department of Computer Science, 1976.

[Mar75] D. A. Martin. *Borel Determinacy*. In: Annals of Mathematics 102.2 (1975), pp. 363–371.

[Mar82] D. A. Martin. *A purely inductive proof of Borel determinacy*. In: Proc. Sympos. Pure Math. Vol. 42. AMS, 1982, pp. 303–308.

[McN93] R. McNaughton. *Infinite Games Played on Finite Graphs*. In: Ann. Pure Appl. Logic 65.2 (1993), pp. 149–184.

[Min67] M. L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.

[Mos91] A. Mostowski. *Games with forbidden positions*. Tech. rep. Uniwersytet Gdański, Instytut Matematyki, 1991.

[MSS05] A. Muscholl, T. Schwentick, and L. Segoufin. *Active Context-Free Games*. In: Theory of Computing Systems 39.1 (2005), pp. 237–276.

[Rac78] C. Rackoff. *The covering and boundedness problems for vector addition systems*. In: TCS 6.2 (1978).

[Saf88] S. Safra. *On the complexity of omega-automata*. In: SFCS. 1988, pp. 319–327.

[Set09] A. Seth. *Games on Multi-stack Pushdown Systems*. In: LFCS. 2009, pp. 395–408.

[SW01] U. Schwalbe and P. Walker. *Zermelo and the Early History of Game Theory*. In: Games and Economic Behavior 34.1 (2001), pp. 123–137.

[V A88] L. G. K. V. A. Gurvich A. V. Karzanov. *Cyclic games and an algorithm to find minimax cycle means in directed graphs*. In: U.S.S.R. Comput. Math. Math. Phys. 28.9 (1988), pp. 1407–1417.

[Wal01] I. Walukiewicz. *Pushdown Processes: Games and Model-Checking*. In: IC 164.2 (2001), pp. 234–263.

[Zer13] E. Zermelo. *Über eine Anwendung der Mengenlehre auf die Theorie des Schachspiels*. In: Proceedings of the Fifth International Congress Mathematics. Cambridge University Press, 1913, pp. 501–504.

[Zie98] W. Zielonka. *Infinite Games on Finitely Coloured Graphs with Applications to Automata on Infinite Trees*. In: Theor. Comput. Sci. 200.1-2 (1998), pp. 135–183.

[ZKW] M. Zimmermann, F. Klein, and A. Weinert. *Infinite Games - Lecture notes*. `https://www.react.uni-saarland.de/teaching/infinite-games-16/lecture-notes.pdf`.

[ZP96] U. Zwick and M. Paterson. *The complexity of mean payoff games on graphs*. In: Theoretical Computer Science 158.1 (1996), pp. 343–359.