Automata, Games, and Verification

Prof. Bernd Finkbeiner, Ph.D.

Saarland University

Summer Term 2015

Lecture Notes by

Bernd Finkbeiner, Felix Klein, Tobias Salzmann

Contents

1 Introduction **1**
- 1.1 Model Checking . . . 1
- 1.2 Synthesis . . . 2
- 1.3 The Logic-Automata Connection . . . 3

2 Büchi Automata **5**
- 2.1 Preliminaries . . . 5
- 2.2 Automata over Infinite Words . . . 5
- 2.3 The Büchi Acceptance Condition . . . 6

3 Büchi's Characterization Theorem **9**
- 3.1 Kleene's Theorem . . . 9
- 3.2 ω-Regular Languages . . . 10
- 3.3 Closure Properties of the Büchi-Recognizable Languages . . . 10
- 3.4 Büchi's Characterization Theorem . . . 13

4 Deterministic Büchi Automata **15**

5 Complementation of Büchi Automata **19**

6 McNaughton's Theorem **25**
- 6.1 The Muller Acceptance Condition . . . 25
- 6.2 From Nondeterministic to Semi-Deterministic Automata . . . 28
- 6.3 From Semi-Deterministic Büchi to Deterministic Muller . . . 30
- 6.4 Safra's Construction . . . 31

7 Logics over Infinite Sequences **35**
- 7.1 Linear-time Temporal Logic (LTL) . . . 35
- 7.2 Quantified Propositional Temporal Logic (QPTL) . . . 37
- 7.3 Monadic Second-Order Logic of One Successor (S1S) . . . 38
- 7.4 Weak Monadic Second-Order Logic of One Successor (WS1S) . . . 41

8 Alternating Büchi Automata **43**
- 8.1 Alternating Automata . . . 43
- 8.2 From LTL to Alternating Automata . . . 44
- 8.3 Translating Alternating to Nondeterministic Automata . . . 45

9 Infinite Games **49**
- 9.1 Basic Definitions . . . 49
- 9.2 Reachability Games . . . 50
- 9.3 Büchi Games . . . 53
- 9.4 Parity Games . . . 55

10 Rabin's Theorem **57**
- 10.1 Tree Automata . . . 57
- 10.2 Complementation of Parity Tree Automata . . . 60
- 10.3 Monadic Second-Order Logic of Two Successors (S2S) . . . 61

11 Computation Tree Logic **65**
- 11.1 CTL . . . 65
- 11.2 Alternating Tree Automata . . . 65
- 11.3 From CTL to Alternating Tree Automata . . . 67

12 Summary **69**
12.1 Automata 69
12.2 Characterization Theorems 70
12.3 Translating Branching Modes 70
12.4 Translating Acceptance Conditions 70
12.5 Automata and Games 70
12.6 Determinacy 71
12.7 Logics 71
12.8 Model Checking and Synthesis 71

1 Introduction

The theory of automata over infinite objects provides a succinct, expressive and formal framework for reasoning about reactive systems, such as communication protocols and control systems. Reactive systems are characterized by their nonterminating behaviour and persistent interaction with their environment. In this course we study the main ingredients of this elegant theory, and its application to automatic verification (model checking) and program synthesis.

1.1 Model Checking

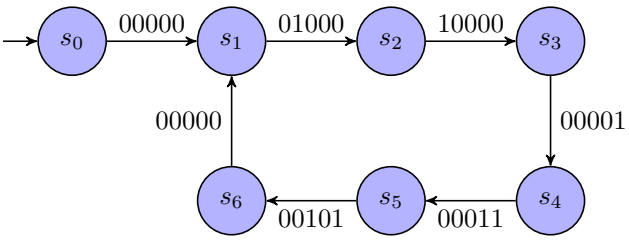
In model checking, we check automatically and exhaustively whether a given system model meets its specification. Typical examples for specifications are the accessibility and mutual exclusion properties of certain critical regions. Automata over infinite objects are used both to represent the system and to represent the specification. The verification problem can be solved by constructing the intersection of the system automaton with an automaton for the negation of the specification, and then checking whether the language of the resulting automaton is empty.

Example 1.1. Mutual execution with program TURN

local t: boolean where initially $t = 0$

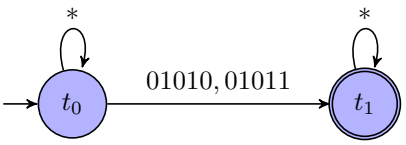
$$P_0 :: \left[\begin{array}{l}\text{loop forever do}\\ \quad\left[\begin{array}{ll}00: & \text{await } t = 0;\\ 01: & \text{critical;}\\ 10: & t := 1;\end{array}\right]\end{array}\right] \;||\; P_1 :: \left[\begin{array}{l}\text{loop forever do}\\ \quad\left[\begin{array}{ll}00: & \text{await } t = 1;\\ 01: & \text{critical;}\\ 10: & t := 0;\end{array}\right]\end{array}\right]$$

Program TURN is a (very) simple solution to the mutual exclusion problem: it ensures that, at any given point of time, at most one process is in the critical region. The following is a representation of the behavior of TURN as an automaton:



The alphabet of the automaton consists of bitvectors of length 5, where the first two bits represent the location of process P_0, the next two bits the location of process P_1, and the final bit the value of t. The automaton is a *safety automaton* (more about this later), it accepts all infinite repetitions of the sequence 00000 01000 10000 00001 00011 00101.

In order to verify that TURN satisfies the mutual exclusion property (P_0 and P_1 are never in location 01 at the same time), we build an automaton that represents the negation (eventually P_0 and P_1 are simultaneously in location 01):



This automaton is a *Büchi automaton* with accepting state t_1. A word is accepted if t_1 is visited infinitely often (more about this later). It is easy to see that for every word accepted by the system automaton, the property automaton stays in t_0 forever and therefore does not accept the word. This means that there does not exist an infinite sequence of bitvectors that is both generated by the program and accepted by the automaton representing the negation of the specification. In other words, TURN is correct.

1.2 Synthesis

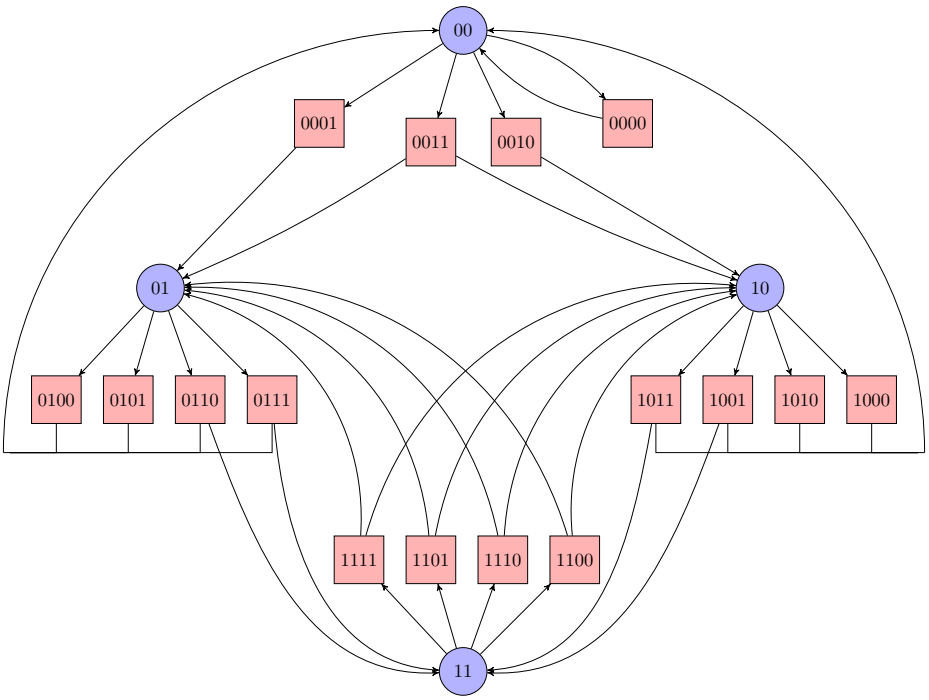
In synthesis, we check automatically if there *exists* a program that satisfies a given specification. If the answer is yes, we also construct such a program. The synthesis problem can be solved by determining the winner of a two-player game between a *system* player and an *environment* player. The system player wins a play if the specification is satisfied. A winning strategy for the system player can be translated into a program that is guaranteed to satisfy the specification.

Example 1.2. Mutual execution by arbitration

$$\text{local } t_0, t_1\text{: boolean where initially } t_0 = t_1 = 0$$

$$P_0 :: \left[\begin{array}{l}\text{loop forever do}\\ \quad\left[\begin{array}{l}0 : \text{await } t_0 = 1\\ 1 : \text{critical;}\end{array}\right]\end{array}\right] \;||\; P_1 :: \left[\begin{array}{l}\text{loop forever do}\\ \quad\left[\begin{array}{l}0 : \text{await } t_1 = 1\\ 1 : \text{critical;}\end{array}\right]\end{array}\right] \;||\; \text{Arbiter:: } \mathbf{?}$$

In this example, the implementation of the arbiter process is left open. We wish to find an implementation that guarantees certain desirable properties such as mutual exclusion. We build a *game arena* where, in each round, first the system player fixes the output t_1 and t_2 of the arbiter process and then the environment player makes a move in processes P_0 or P_1, resolving any nondeterminism, such as whether P_0 or P_1 enters the critical section if both are in location 0 and both t_0 and t_1 have been set to 1 by the arbiter.



We depict states in which the system player gets to chose the next move with circles, and states in which the environment player gets to chose the next move with rectangles. The

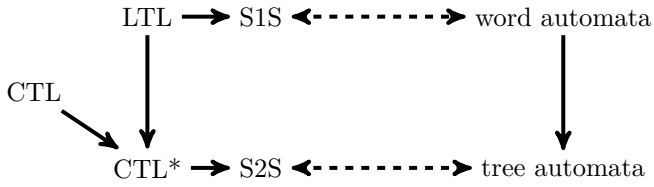
states of the system player consist here of all possible combinations of locations of processes P_0 and P_1, i.e., bitvectors of length 2, where the first bit indicates the location of P_0, and the second bit indicates the location of P_1. The states of the environment player are bitvectors of length 4, where the additional third and forth bit represent the values of t_0 and t_1, respectively, that were chosen by the system player in the previous move.

Suppose now that our specification consists again of the mutual exclusion property, i.e., we wish to ensure that state 11 is never visited. Our game is now a *safety* game, which consists of a game arena and a set of *safe* states (more about this later). Here, the safe states are all states except 11. The system player wins all plays in which the game stays in the safe states forever. Note that the system player has a very simple *strategy* to win any game that starts in state 00: simply choose the move to state 0000, i.e., block both processes. This arbiter is indeed correct, at least with respect to the mutual exclusion property!

More comprehensive specifications lead to more interesting winning strategies and more useful synthesized programs. We might, for example, require that every process visits its critical section infinitely often, i.e., the play must visit states 10 or 11 infinitely often, and also states 01 or 11 infinitely often. A winning strategy for the system player might then, for example, alternate between 0010 and 0001 in subsequent visits to 00, and choose 1000 from 10, and 0100 from 01, in order to avoid visits to 11.

1.3 The Logic-Automata Connection

In applications like verification and synthesis, the automata- and game-theoretic machinery is usually "hidden" behind a logical formulation of the problem. For example, we might express the mutual exclusion property as a formula of linear-time temporal logic (LTL) such as $\Box\neg(at_1(P_0) \wedge at_1(P_1))$. The verification or synthesis algorithm then relies on a translation of the formula into an equivalent automaton, and, for synthesis, on a further translation into a game. The following picture shows some useful logics together with their corresponding automata:



- Linear-time temporal logic (LTL) describes sets of infinite words.

  Example: $\Box\Diamond\, at_1(P_0)$, meaning that P_0 is infinitely often at location 1.

- Computation-tree logic (CTL / CTL*) describes sets of infinite trees.

  Example: $\mathsf{EF}\, at_1(P_0) \wedge \mathsf{EF}\, at_1(P_1)$, meaning that there exists a computation path in which P_0 reaches location 1, and there is a (possibly different) computation path in which P_1 reaches location 1.

- Monadic second-order logic with one successor (S1S) is a logic over infinite words. Its expressiveness exceeds that of LTL.

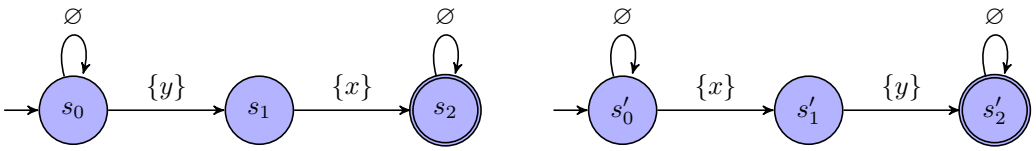
  Example: $\forall x\ .\ x \in P \rightarrow S(x) \in P$, meaning that a (given) set of natural numbers P is either empty or consists of all positions starting from some position of the word.

- Monadic second-order logic with two successors (S2S) is a logic over binary trees. Its expressiveness exceeds that of CTL* (on binary trees).

  Example: $\forall x\ .\ x \in P \rightarrow S_1(x) \in P \vee S_2(x) \in P$, meaning that a (given) set of nodes P contains from each node $n \in P$ an entire path starting in n.

Besides verification and synthesis, an immediate and very important application of the connection between logic and automata is to decide the *satisfiability* problem of the various logics.

Example 1.3. Suppose we wish to know if there exist two natural numbers x and y such that $x = y + 1$ and $y = x + 1$. This is expressed as the S1S formula $x = S(y) \ \wedge \ y = S(x)$. We translate each conjunct into an automaton:



Here the alphabet consists of subsets of $\{x, y\}$. A variable appears at exactly one position in a word, namely at the position that corresponds to its value. It is easy to see that there does not exist a word that is accepted by both automata. Hence, our formula is unsatisfiable.

2 Büchi Automata

We now introduce our first type of automata over infinite words. Büchi automata are named after Julius Richard Büchi, a Swiss logician and mathematician (∗1924, †1984). Büchi automata were invented during the 1960s for the purpose of developing a decision procedure for S1S. Even though Büchi automata are a fairly simple variation of standard finite automata, they already suffice to characterize the entire class of ω-regular languages.

2.1 Preliminaries

We begin with some basic notations. We use the symbol $\mathbb{N}$ to denote the set $\{0, 1, 2, 3, \ldots\}$ of natural numbers and $\mathbb{N}^+$ to denote the set $\{1, 2, 3, \ldots\}$ of positive natural numbers. For $n, m \in \mathbb{N}$ with $n \leq m$ we use $[n, m]$ to denote the set $\{n, n+1, \ldots, m\}$ and $[n]$ for the special case of $[0, n-1]$. An *alphabet* is a nonempty, finite set of symbols, usually denoted by Σ. The elements of an alphabet are called letters. Let an alphabet Σ be given, then the concatenation $w = w(0)w(1)...w(n-1)$ of finitely many letters of Σ is called a *finite word* over Σ, where n defines the length of w also denoted by $|w|$. The only word of length 0 is the empty word denoted by ϵ. The *set of all finite words* over Σ is denoted by Σ^*. For $\Sigma^* \setminus \{\epsilon\}$ we use the shortcut Σ^+. Given some word w, we have that for each $n \in [|w|]$ the n-th letter of w is denoted by $w(n)$.

The concatenation of infinitely many letters defines an infinite word which has infinite length. The *set of all infinite words* is denoted by Σ^ω. If Σ is an alphabet, then each subset of Σ^* is a *language over finite words*. Each subset of Σ^ω is a *language over infinite words*, also referred to as an *ω-language*.

2.2 Automata over Infinite Words

The operational behavior of an automaton over infinite words is very similar to that of a standard automaton over finite words; starting with an initial state, the automaton constructs a run by reading one letter of the input alphabet at a time and transitioning to a successor state permitted by its transitions. Most components of an automaton over infinite words are thus familiar from the standard automata over finite words: an alphabet Σ, a finite set Q of states, a set of initial states I, a set of transitions T. Automata over infinite words differ from automata over finite words, and different types of automata over infinite words differ among each other in the *acceptance condition*, which is responsible for determining whether an infinite run of the automaton is accepting or not. We leave this component generic in our general definition of automata over infinite words, and later give a first concrete definition for the case of Büchi automata.

Definition 2.1. An *automaton over infinite words* is a tuple $\mathcal{A} = (\Sigma, Q, I, T, Acc)$, where

- Σ is a finite *alphabet*,
- Q is a finite set of *states*,
- $I \subseteq Q$ is a subset of *initial states*,
- $T \subseteq Q \times \Sigma \times Q$ is a set of *transitions*, and
- $Acc \subseteq Q^\omega$ is the *accepting condition*.

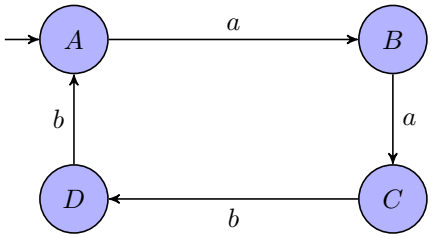
In the following, we refer to an automaton over infinite words (whenever clear from context) simply as an automaton. Now we define how an automaton uses an infinite word as input.

Note that we do not refer to acceptance in this definition.

Definition 2.2. A *run* of an automaton $\mathcal{A}$ on an infinite input word α is an infinite sequence of states $r = r(0)r(1)r(2)\ldots$ such that the following hold:

- $r(0) \in I$
- for all $i \in \mathbb{N}$, $(r(i), \alpha(i), r(i+1)) \in T$

Example 2.1. The following is a graphical representation of an automaton over the alphabet $\Sigma = \{a, b\}$ and with set of states $Q = \{A, B, C, D\}$, initial set of states $I = \{A\}$, and set of transitions $T = \{(A, a, B), (B, a, C), (C, b, D), (D, b, A)\}$:



On the infinite input word $aabbaabb\ldots$, the (only) run of the automaton is

$$ABCDABCDABCD\ldots.$$

An automaton is *deterministic* if $|I| \leq 1$ and $|\{(q, \sigma, q') \in T \mid q' \in Q\}| \leq 1$ for every $q \in Q$ and $\sigma \in \Sigma$, otherwise the automaton is nondeterministic. The automaton is *complete* if $|\{(q, \sigma, q') \in T \mid q' \in Q\}| \geq 1$ for every $q \in Q$ and $\sigma \in \Sigma$. For deterministic and complete automata, the transitions are usually given as a function $\delta \colon Q \times \Sigma \to Q$. The automaton in the example is deterministic but not complete.

Definition 2.3. An automaton $\mathcal{A}$ *accepts* an infinite word α if:

- there is a run r of $\mathcal{A}$ on α, and
- r is *accepting*: $r \in Acc$.

The *language recognized by* $\mathcal{A}$ is defined as follows: $\mathcal{L}(\mathcal{A}) = \{\alpha \in \Sigma^\omega \mid \mathcal{A} \text{ accepts } \alpha\}$

We say that two automata are *equivalent* if they have the same language.

2.3 The Büchi Acceptance Condition

The Büchi acceptance condition is given as a set of *accepting states* F. A run of a Büchi automaton is *accepting* if some state from this set occurs infinitely often. Formally, for an infinite word α over Σ we use $\mathrm{Inf}(\alpha) = \{\sigma \in \Sigma \mid \forall m \in \mathbb{N}.\, \exists n \in \mathbb{N}.\, n \geq m \text{ and } \alpha(n) = \sigma\}$ to denote the set of all letters of Σ that occur infinitely often in α, called the *infinity set* of α.

Definition 2.4. The *Büchi condition* $\textsc{büchi}(F)$ on a set of states $F \subseteq Q$ is the set

$$\textsc{büchi}(F) = \{\alpha \in Q^\omega \mid \mathrm{Inf}(\alpha) \cap F \neq \varnothing\}.$$

An automaton $\mathcal{A} = (\Sigma, Q, I, T, Acc)$ with $Acc = \textsc{büchi}(F)$ is called a *Büchi automaton*. The set F is called the *set of accepting states*.

Example 2.2. Let $\mathcal{A}$ be the automaton from Example 2.1 with Büchi acceptance condition $\textsc{büchi}(\{D\})$. The language of the automaton consists of a single word:

$$\mathcal{L}(\mathcal{A}) = \{aabbaabbaabb\ldots\}$$

As a first observation about Büchi automata, we establish that for every Büchi automaton $\mathcal{A}$ one can construct an equivalent complete Büchi automaton $\mathcal{A}'$:

Construction 2.1. For a Büchi automaton $\mathcal{A} = (\Sigma, Q, I, T, \textsc{büchi}(F))$, we define the complete Büchi automaton $\mathcal{A}' = (\Sigma, Q', I', T', \textsc{büchi}(F'))$ using:

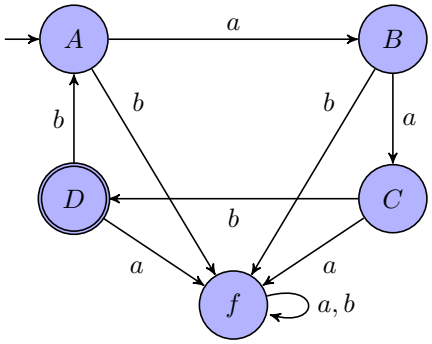
- $Q' = Q \cup \{q_f\}$ where q_f is a fresh state
- $I' = I$
- $T' = T \cup \{(q, \sigma, q_f) \mid \nexists q' \,.\, (q, \sigma, q') \in T\} \cup \{(q_f, \sigma, q_f) \mid \sigma \in \Sigma\}$
- $F' = F$

We prove the correctness of the construction by the following theorem.

Theorem 2.1. *For every Büchi automaton $\mathcal{A}$, there is a complete Büchi automaton $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.*

Proof. The runs of $\mathcal{A}'$ are a superset of those of $\mathcal{A}$, because we have not removed any transitions during the construction of $\mathcal{A}'$. Furthermore, on any infinite input word the accepting runs of $\mathcal{A}$ and $\mathcal{A}'$ are the same, because any run that reaches the new state q_f stays in q_f forever, and, since $q_f \notin F'$, such a run is not accepting. □

Example 2.3. Completing the automaton from the previous example we obtain the following Büchi automaton (the accepting state D is shown with double lines):



A complete deterministic automaton may be viewed as a total function from Σ^ω to Q^ω. A complete (possibly nondeterministic) automaton produces at least one run for every input word.

3 Büchi's Characterization Theorem

Our next goal is to characterize the languages that can be recognized by Büchi automata. For languages over finite words, Kleene's theorem states that the set of languages over finite words that can be defined by regular expressions is exactly the set of languages that can be recognized by automata over finite words. We quickly review this result, then define ω-regular expressions, and finally prove the corresponding theorem for ω-regular languages: Büchi's characterization theorem.

3.1 Kleene's Theorem

We give a quick recap of regular languages and Kleene's theorem. This is not meant as a full introduction to finite automata theory; if you want to brush up some more on this, we recommend the textbook "*Introduction to Automata Theory, Languages, and Computation*" by Hopcroft, Motwani, and Ullman.

Regular expressions consist of constants and operator symbols that denote languages of finite words and operations over these languages, respectively.

Definition 3.1. *Regular expressions* are defined as follows:

- The constants ε and $\varnothing$ are regular expressions.
  $\mathcal{L}(\varepsilon) = \{\varepsilon\}, \mathcal{L}(\varnothing) = \varnothing$.
- If $a \in \Sigma$ is a symbol, then a is a regular expression.
  $\mathcal{L}(a) = \{a\}$.
- If E and F are regular expressions, then $E + F$ is a regular expression:
  $\mathcal{L}(E + F) = \mathcal{L}(E) \cup \mathcal{L}(F)$.
- If E and F are regular expressions, then $E \cdot F$ is a regular expression:
  $\mathcal{L}(E \cdot F) = \{uv \mid u \in \mathcal{L}(E), v \in \mathcal{L}(F)\}$.
- If E is a regular expression, then E^* is a regular expression.
  $\mathcal{L}(E^*) = \{u_1 u_2 \dots u_n \mid n \in \mathbb{N}, u_i \in \mathcal{L}(E) \text{ for all } 0 \leq i \leq n\}$.

A language over finite words is *regular* if it is definable by a regular expression. Alternatively, regular languages can be defined as the languages recognized by automata over finite words.

Definition 3.2. An *automaton on finite words* $\mathcal{A}$ is a tuple (Σ, Q, I, T, F), where Σ is an input alphabet, Q is a nonempty finite set of states, $I \subseteq Q$ is a set of initial states, $\Delta \subseteq Q \times \Sigma \times Q$ is a set of transitions, and $F \subseteq Q$ are a set of final states.

An automaton $\mathcal{A}$ accepts a finite word $w \in \Sigma^*$ if there is a finite sequence of states $q(0)q(1)\dots q(|w|)$ such that $q(0) \in I$ and $(q(i), w(i), q(i+1)) \in \Delta$ for all $i < |w|$ and with $q(|w|) \in F$. The set of all words accepted by $\mathcal{A}$ is called the *language* of $\mathcal{A}$, denoted by $\mathcal{L}(\mathcal{A})$. The equivalence of regular expressions and finite automata is known as Kleene's theorem.

Theorem 3.1 (Kleene's Theorem)**.** *A language is regular iff it is recognized by some finite word automaton.*

3.2 ω-Regular Languages

ω-regular expressions denote languages over infinite words. In addition to language union on languages over infinite words, we have two operations that convert languages over finite words into languages over infinite words: the *infinite concatenation* R^ω of words of a regular language R and the *concatenation* $R{\cdot}U$ of a regular language R and an ω-regular language U.

Definition 3.3. The *ω-regular expressions* are defined as follows.

- If E is a regular expression where $\varepsilon \notin \mathcal{L}(E)$,
  then E^ω is an ω-regular expression.
  $\mathcal{L}(E^\omega) = \mathcal{L}(E)^\omega$
  where $L^\omega = \{w_0 w_1 \ldots \mid w_i \in L, |w_i| > 0 \text{ for all } i \in \mathbb{N}\}$ for $L \subseteq \Sigma^*$.
- If E is a regular expression and W is an ω-regular expression,
  then $E \cdot W$ is an ω-regular expression.
  $\mathcal{L}(E \cdot W) = \mathcal{L}(E) \cdot \mathcal{L}(W)$
  where $L_1 \cdot L_2 = \{w\alpha \mid w \in L_1, \alpha \in L_2\}$ for $L_1 \subseteq \Sigma^*, L_2 \subseteq \Sigma^\omega$.
- If W_1 and W_2 are ω-regular expressions,
  then $W_1 + W_2$ is an ω-regular expression.
  $\mathcal{L}(W_1 + W_2) = \mathcal{L}(W_1) \cup \mathcal{L}(W_2)$.

A language over infinite words is *ω-regular* if it is definable by an ω-regular expression.

3.3 Closure Properties of the Büchi-Recognizable Languages

Büchi's characterization theorem states that the ω-regular languages are exactly the languages recognized by Büchi automata. In the following we will refer to the languages recognized by Büchi automata simply as *Büchi-recognizable languages*. To prepare for the proof of Büchi's theorem, we establish several *closure properties* of the Büchi-recognizable languages. First off, the Büchi-recognizable languages are closed under language union, as demonstrated by the following simple construction.

Construction 3.1. Let L_1 and L_2 be ω-languages recognized by the Büchi automata $\mathcal{A}_1 = (\Sigma, Q_1, I_1, T_1, \textsc{büchi}(F_1))$ and $\mathcal{A}_2 = (\Sigma, Q_2, I_2, T_2, \textsc{büchi}(F_2))$, respectively. We construct $\mathcal{A}_\cup = (\Sigma, Q_\cup, I_\cup, T_\cup, \textsc{büchi}(F_\cup))$ with $\mathcal{L}(\mathcal{A}_\cup) = L_1 \cup L_2$ as follows:

- $Q_\cup = Q_1 \cup Q_2$ (w.l.o.g. we assume $Q_1 \cap Q_2 = \varnothing$)
- $I_\cup = I_1 \cup I_2$
- $T_\cup = T_1 \cup T_2$
- $F_\cup = F_1 \cup F_2$

The correctness of this construction is proven by the following theorem.

Theorem 3.2. *If L_1 and L_2 are Büchi-recognizable, then so is $L_1 \cup L_2$.*

Proof. We prove that the Büchi automaton $\mathcal{A}_\cup$ built by Construction 3.1 from the Büchi automata $\mathcal{A}_1$ and $\mathcal{A}_2$ indeed recognizes the union of the languages of the two automata.

$\mathcal{L}(\mathcal{A}_\cup) \subseteq \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$: For $\alpha \in \mathcal{L}(A_\cup)$, we have an accepting run $r = r(0)r(1)r(2)\ldots$ of $\mathcal{A}_\cup$ on α. If $r(0) \in I_1$, then r is an accepting run of $\mathcal{A}_1$, otherwise $r(0) \in I_2$ and r is an accepting run of $\mathcal{A}_2$.

$\mathcal{L}(\mathcal{A}_\cup) \supseteq \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$: For $i \in \{1, 2\}$ and $\alpha \in \mathcal{L}(\mathcal{A}_i)$, there is an accepting run r of $\mathcal{A}_i$. The run r is also an accepting run of $\mathcal{A}_\cup$. □

As we will see later, the Büchi-recognizable languages are closed under complement. Together with the closure under union, the closure under complement implies the closure under intersection (because the intersection of two sets is the complement of the union of the complements of the two sets). Since the proof of the closure under complement is more complicated, we postpone this for now, however, and instead give a direct construction for the closure under intersection.

Construction 3.2. Let L_1 and L_2 be ω-languages recognized by the Büchi automata $\mathcal{A}_1 = (\Sigma, Q_1, I_1, T_1, \textsc{büchi}(F_1))$ and $\mathcal{A}_2 = (\Sigma, Q_2, I_2, T_2, \textsc{büchi}(F_2))$, respectively. We construct $\mathcal{A}_\cap = (\Sigma, Q_\cap, I_\cap, T_\cap, \textsc{büchi}(F_\cap))$ with $\mathcal{L}(\mathcal{A}_\cap) = L_1 \cap L_2$ as follows:

- $Q_\cap = Q_1 \times Q_2 \times \{1, 2\}$
- $I_\cap = I_1 \times I_2 \times \{1\}$
- $T_\cap = \{((q_1, q_2, i), \sigma, (q_1', q_2', j)) \mid (q_1, \sigma, q_1') \in T_1,\ (q_2, \sigma, q_2') \in T_2,\ i, j \in \{1, 2\},$
  $q_i \notin F_i \ \rightarrow\ i = j \quad \wedge \quad q_i \in F_i \ \rightarrow\ i \neq j\}$
- $F_\cap = \{(q_1, q_2, 2) \mid q_1 \in Q_1,\ q_2 \in F_2\}$

The automaton for the intersection is essentially the product of the two automata. The states of the new automaton contain the states of the two automata plus a third component, which is used to combine the acceptance conditions of the two automata. The product automaton alternates between waiting for a visit to the accepting states of the first automaton (in which case the third component is 1) and waiting for a visit to the accepting states of the second automaton (in which case the third component is 2). The accepting states of the product are those states where the third component is 2 *and* where the second automaton visits an accepting state, i.e., where both automata have seen an accepting state since the last visit to an accepting state of the product automaton.

Theorem 3.3. *If L_1 and L_2 are Büchi-recognizable, then so is $L_1 \cap L_2$.*

Proof. We prove that the Büchi automaton $\mathcal{A}_\cap$ built by Construction 3.2 from the Büchi automata $\mathcal{A}_1$ and $\mathcal{A}_2$ indeed recognizes the intersection of the languages of the two automata.

$r' = (q_1^0, q_2^0, t^0)(q_1^1, q_2^1, t^1)\ldots$ is a run of $\mathcal{A}_\cap$ on an input word α iff $r_1 = q_1^0 q_1^1 \ldots$ is a run of $\mathcal{A}_1$ on α and $r_2 = q_2^0 q_2^1 \ldots$ is a run of $\mathcal{A}_2$ on α. The run r' is accepting iff r_1 is accepting and r_2 is accepting. □

Next, the concatenation of a regular language and a Büchi-recognizable language is again Büchi-recognizable.

Construction 3.3. Let $\mathcal{A}_1 = (\Sigma, Q_1, I_1, T_1, F_1)$ be an automaton over finite words that recognizes the language L_1, and let $\mathcal{A}_2 = (\Sigma, Q_2, I_2, T_2, \textsc{büchi}(F_2))$ be a Büchi automaton over the same alphabet that recognizes L_2. We construct a Büchi automaton $\mathcal{A}' = (\Sigma, Q', I', T', \textsc{büchi}(F_2))$ with $\mathcal{L}(\mathcal{A}') = L_1 \cdot L_2$ as follows:

- $Q' = Q_1 \cup Q_2$ (w.l.o.g. we assume $Q_1 \cap Q_2 = \varnothing$)
- $I' = \begin{cases} I_1 & \text{if } I_1 \cap F_1 = \varnothing \\ I_1 \cup I_2 & \text{otherwise} \end{cases}$
- $T' = T_1 \cup T_2 \cup \{(q, \sigma, q') \mid (q, \sigma, f) \in T_1, f \in F_1, q' \in I_2\}$

The correctness of this construction is proven by the following theorem.

Theorem 3.4. *If L_1 is a regular language and L_2 is Büchi-recognizable, then $L_1 \cdot L_2$ is Büchi-recognizable.*

Proof. We prove that the Büchi automaton $\mathcal{A}'$ built in Construction 3.3 from the automaton on finite words $\mathcal{A}_1$ and the Büchi automaton $\mathcal{A}_2$ indeed recognizes the concatenation of the languages of the two automata.

$\mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}(\mathcal{A}_1) \cdot \mathcal{L}(\mathcal{A}_2)$: For $\alpha \in \mathcal{L}(\mathcal{A}')$, let $r = r(0)r(1)r(2)\dots$ be an accepting run of $\mathcal{A}'$ on α. If $r(0) \in I_1$, then there is an $n \in \mathbb{N}$ such that $(r(n), \alpha(n), r(n+1)) \in Q_1 \times \Sigma \times I_2$ and therefore, there is a a final state $f \in F_1$ such that $r(0)r(1)\dots r(n)f$ is an accepting run of $\mathcal{A}_1$ on $\alpha(0)\alpha(1)\cdots\alpha(n)$ and $r(n+1)r(n+2)\dots$ is an accepting run of $\mathcal{A}_2$ on $\alpha(n+1)\alpha(n+2)\dots$.
If $r(0) \in I_2$ then $I_1 \cap F_1 \neq \varnothing$ and therefore, $\epsilon \in \mathcal{L}(\mathcal{A}_1)$, $\alpha \in \mathcal{L}(\mathcal{A}_2)$.

$\mathcal{L}(\mathcal{A}') \supseteq \mathcal{L}(\mathcal{A}_1) \cdot \mathcal{L}(\mathcal{A}_2)$: For $w \in \mathcal{L}(\mathcal{A}_1)$, let $r = r(0)r(1)\cdots r(n)$ be an accepting run of $\mathcal{A}_1$ on w. For $\alpha \in \mathcal{L}(\mathcal{A}_2)$, let $s = s(0)s(1)\dots$ be an accepting run of $\mathcal{A}_2$ on α. Then, $r(n) \in F_1$ and, by construction, $r(0)r(1)\dots r(n-1)s(0)s(1)\dots$ is an accepting run of $\mathcal{A}'$ on $w\alpha$. □

Finally, we show that the infinite concatenation of words of a regular language forms a Büchi-recognizable language. We construct a Büchi automaton for this language from an automaton over finite words in two steps. The first construction modifies a given automaton over finite words into an equivalent automaton with a single initial state that has no incoming transitions.

Construction 3.4. Let $\mathcal{A} = (\Sigma, Q, I, T, F)$ be an automaton over finite words. We assume that $\varepsilon \notin \mathcal{L}(\mathcal{A})$. We construct an automaton $\mathcal{A}' = (\Sigma, Q', I', T', F)$ over finite words such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$ and $\mathcal{A}'$ has a single initial state that has no incoming transitions.

- $Q' = Q \cup \{q_f\}$ where q_f is a fresh state
- $I' = \{q_f\}$
- $T' = T \cup \{(q_f, \sigma, q') \mid (q, \sigma, q') \in T \text{ for some } q \in I\}$

The second construction builds the Büchi automaton that recognizes the infinite concatenations of words from the regular language by adding a loop to the modified automaton.

Construction 3.5. Let $\mathcal{A}$ be an automaton over finite words, for which we assume that $\varepsilon \notin \mathcal{L}(\mathcal{A})$. We construct a Büchi automaton $\mathcal{A}'' = (\Sigma, Q'', I'', T'', \textsc{büchi}(F''))$ such that $\mathcal{L}(\mathcal{A}'') = \mathcal{L}(\mathcal{A})^\omega$. Let $\mathcal{A}' = (\Sigma, Q', I', T', F')$ be the automaton of Construction 3.4. Then $\mathcal{A}''$ is defined as follows:

- $Q'' = Q'$
- $I'' = I'$
- $T'' = T' \cup \{(q, \sigma, q_f) \mid (q, \sigma, q') \in T' \text{ and } q' \in F'\}$
- $F'' = I'$

The correctness of this construction is proven by the following theorem.

Theorem 3.5. *If L is a regular language such that $\epsilon \notin L$, then L^ω is Büchi-recognizable.*

Proof. Construction 3.4 does not affect the language of $\mathcal{A}$. We show that the Büchi automaton $\mathcal{A}''$ built in Construction 3.5 from the resulting automaton $\mathcal{A}'$ on finite words indeed recognizes $\mathcal{L}(\mathcal{A}')^\omega$.

$\mathcal{L}(\mathcal{A}'') \subseteq \mathcal{L}(\mathcal{A}')^\omega$: Assume that $\alpha \in \mathcal{L}(\mathcal{A}'')$ and that $r(0)r(1)r(2)\ldots$ is an accepting run of $\mathcal{A}''$ on α. Hence, we have that $r(i) = q_f \in F'' = I'$ for infinitely many indices i: $i_0, i_1, i_2, \ldots$. This provides a sequence of runs of $\mathcal{A}'$:

- run $r(0)r(1)\ldots r(i_0-1)q$ on $w_0 = \alpha(0)\alpha(1)\ldots\alpha(i_0-1)$ for some $q \in F'$
- run $r(i_0)r(i_0+1)\ldots r(i_1-1)q$ on $w_1 = \alpha(i_0)\alpha(i_0+1)\ldots\alpha(i_1-1)$ for some $q \in F'$
- and so forth.

We thus have that $w_k \in \mathcal{L}(\mathcal{A}')$ for every $k \geq 0$. Hence, $\alpha \in \mathcal{L}(\mathcal{A}')^\omega$.

$\mathcal{L}(\mathcal{A}'') \supseteq \mathcal{L}(\mathcal{A}')^\omega$: Assume that $\alpha = w_0w_1w_2\ldots \in \Sigma^\omega$ such that $w_k \in \mathcal{L}(\mathcal{A}')$ for all $k \geq 0$. For each k, we choose an accepting run $r_k(0)r_k(1)r_k(2)\ldots r_k(n_k)$ of $\mathcal{A}'$ on w_k. Hence, $r_k(0) \in I'$ and $r_k(n_k) \in F'$ for all $k > 1$. Thus,

$$r_0(0)\ldots r_0(n_0-1)r_1(0)\ldots r_1(n_1-1)r_2(0)\ldots r_2(n_2-1)\ldots$$

is an accepting run of $\mathcal{A}''$ on α. Hence, $\alpha \in \mathcal{L}(\mathcal{A}'')$. □

3.4 Büchi's Characterization Theorem

We are now ready to prove Büchi's Characterization Theorem, a result from 1962.

Theorem 3.6 (Büchi's Characterization Theorem)**.** *An ω-language is Büchi-recognizable iff it is ω-regular.*

Proof. "$\Leftarrow$" follows from the closure properties of the Büchi-recognizable languages established by Theorems 3.2, 3.4, and 3.5.

"$\Rightarrow$": Given a Büchi automaton $\mathcal{A}$, we consider for each pair $q, q' \in Q$ the regular language

$$W_{q,q'} = \{w \in \Sigma^* \mid \text{finite-word automaton } (\Sigma, Q, \{q\}, T, \{q'\}) \text{ accepts } w\ \}\ .$$

We claim that $\mathcal{L}(\mathcal{A}) = \bigcup_{q\in I, q'\in F} W_{q,q'} \cdot (W_{q',q'} \setminus \{\epsilon\})^\omega$. The claim is proven as follows.

$\mathcal{L}(\mathcal{A}) \subseteq \bigcup_{q\in I, q'\in F} W_{q,q'} \cdot (W_{q',q'} \setminus \{\epsilon\})^\omega$: Let $\alpha \in \mathcal{L}(\mathcal{A})$. Then there is an accepting run r of $\mathcal{A}$ on α, which begins at some $q = r(0) \in I$ and visits some $q' \in F$ infinitely often:

$$r : q \xrightarrow{w_0} q' \xrightarrow{w_1} q' \xrightarrow{w_2} q' \xrightarrow{w_3} \ \ldots,$$

where $w_i \in \Sigma^*$ for all $i \geq 0$, $|w_i| > 0$ for all $i > 0$ and $\alpha = w_0w_1w_2\ldots$. The notation $q_0 \xrightarrow{w} q_{k+1}$ for some finite word $w = w(0)w(1)\ldots w(k)$ means that there exist states $q_1, \ldots q_k \in Q$ s.t. $(q_i, w(i), q_{i+1}) \in T$ for all $0 \leq i \leq k$. Since $w_0 \in W_{q,q'}$ and $w_k \in W_{q',q'}$ for $k > 0$, we have that $\alpha \in W_{q,q'} \cdot {W_{q',q'}}^\omega$ for some $q \in I, q' \in F$.

$\mathcal{L}(\mathcal{A}) \supseteq \bigcup_{q\in I, q'\in F} W_{q,q'} \cdot (W_{q',q'} \setminus \{\epsilon\})^\omega$: Let $\alpha = w_0w_1w_2\ldots$ with $w_0 \in W_{q,q'}$ and $w_k \in W_{q',q'}$ for some $q \in I, q' \in F$ and for all $k > 0$. Then the run

$$r : q \xrightarrow{w_0} q' \xrightarrow{w_1} q' \xrightarrow{w_2} q' \xrightarrow{w_3} \ldots$$

exists and is accepting because $q' \in F$. It follows that $\alpha \in \mathcal{L}(\mathcal{A})$. □

4 Deterministic Büchi Automata

One of the most important constructions in the theory of automata over finite words is the Rabin-Scott *powerset construction*, which converts a nondeterministic automaton over finite words into a deterministic automaton that recognizes the same language. The powerset construction establishes that while nondeterminism makes automata over finite words more *concise* (the powerset construction produces an exponential number of states), nondeterminism does not make automata over finite words more *expressive*. The situation is different for Büchi automata: even though the language $L = (a+b)^*b^\omega$ is clearly Büchi-recognizable, there is, as the following theorem shows, no deterministic Büchi automaton that recognizes L.

Theorem 4.1. *The language $L = (a+b)^*b^\omega$ is not recognizable by a deterministic Büchi automaton.*

Proof. Assume, by way of contradiction, that L is recognizable by the deterministic Büchi automaton $\mathcal{A} = (\Sigma, Q, I, T, \textsc{büchi}(F))$. Since $\alpha_0 = b^\omega$ is in L, there is a unique run

$$r_0 = r_0(0)r_0(1)r_0(2)\dots$$

of $\mathcal{A}$ on α_0 with $r_0(n_0) \in F$ for some $n_0 \in \mathbb{N}$. Similarly, $\alpha_1 = b^{n_0}ab^\omega$ in L and there is a unique run

$$r_1 = r_0(0)r_0(1)r_0(2)\dots r_0(n_0)r_1(n_0+1)r_1(n_0+2)\dots$$

of $\mathcal{A}$ on α_1 with $r_1(n_1) \in F$ for some $n_1 > n_0$. Because $\mathcal{A}$ is deterministic, r_0 and r_1 are identical up to position n_0.

By repeating this argument infinitely often, we obtain a word $\alpha = b^{n_0}ab^{n_1}ab^{n_2}a\dots$ and a run r with infinitely many visits to F. Hence, α is accepted by $\mathcal{A}$. However, α is not an element of L. This contradicts $L = \mathcal{L}(\mathcal{A})$. □

What languages are recognizable by deterministic Büchi automata? A straightforward characterization can be given in terms of the *limit operator*.

Definition 4.1 (Limit)**.** The *limit* $\overrightarrow{W}$ of a language $W \subseteq \Sigma^*$ over finite words is the following language over infinite words:

$$\overrightarrow{W} = \{\alpha \in \Sigma^\omega \mid \text{there exist infinitely many } n \in \mathbb{N} \text{ s.t. } \alpha[0,n] \in W\}.$$

Theorem 4.2. *An ω-language $L \subseteq \Sigma^\omega$ is recognizable by a deterministic Büchi automaton iff there is a regular language $W \subseteq \Sigma^*$ s.t. $L = \overrightarrow{W}$.*

Proof. We claim that the languages of a deterministic Büchi automaton $\mathcal{A}_B$ and of a deterministic automaton over finite words $\mathcal{A}_F$, where the automata $\mathcal{A}_B = (\Sigma, Q, I, T, \textsc{büchi}(F))$ and $\mathcal{A}_F = (\Sigma, Q, I, T, F)$, constructed from the same components, are related as follows:

$$\mathcal{L}(\mathcal{A}_B) = \overrightarrow{\mathcal{L}(\mathcal{A}_F)}.$$

Since every regular language is recognized by a deterministic automaton over finite words, the theorem follows. To prove the claim, we consider an infinite word $\alpha \in \Sigma^\omega$.

$\alpha \in \mathcal{L}(\mathcal{A}_B)$

iff for the unique run r of $\mathcal{A}_B$ on $\alpha, \mathrm{Inf}(r) \cap F \neq \varnothing$

iff $\alpha[0,n] \in \mathcal{L}(\mathcal{A}_F)$ for infinitely many $n \in \mathbb{N}$

iff $\alpha \in \overrightarrow{\mathcal{L}(\mathcal{A}_F)}$ □

Next on our agenda is the *complementation* of languages. For regular languages, this is very simple: one translates a given complete deterministic automaton $\mathcal{A}$ that recognizes some language $L \subseteq \Sigma^*$ into an automaton $\mathcal{A}'$ that recognizes the complement $\Sigma^* \setminus L$ by complementing the set of final states, i.e., $F' = F \setminus F$. For deterministic Büchi automata, the construction is slightly more complicated, because it introduces nondeterminism.

Construction 4.1. Let $\mathcal{A} = (\Sigma, Q, I, T, \textsc{büchi}(F))$ be a complete deterministic Büchi automaton. We construct a Büchi automaton $\mathcal{A}' = (\Sigma, Q', I', T', \textsc{büchi}(F'))$ with $\mathcal{L}(\mathcal{A}') = \Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$ as follows:

- $Q' = (Q \times \{0\}) \cup ((Q \setminus F) \times \{1\})$
- $I' = I \times \{0\}$
- $T' = \{((q,0), \sigma, (q',i)) \mid (q, \sigma, q') \in T,\ i \in \{0,1\},\ (q',i) \in Q'\} \cup$
  $\{((q,1), \sigma, (q',1)) \mid (q, \sigma, q') \in T, q' \in Q \setminus F\}$
- $F' = (Q \setminus F) \times \{1\}$

The construction uses two copies of the given automaton $\mathcal{A}$, where all accepting states are eliminated from the second copy. The switch from the first copy (identified by the flag 0) to the second copy (identified by the flag 1) happens nondeterministically. The automaton accepts if the run ends up in the second copy, which means that, on the unique run of $\mathcal{A}$ on the input word, the accepting states of $\mathcal{A}$ are only visited finitely often. Note that the resulting automaton is nondeterministic. This is, in general, unavoidable, because there are languages, such as $L = (b^*a)^\omega$ where the language itself is recognizable by a deterministic Buchi automaton, while its complement $\Sigma^\omega \setminus L$ can only be recognized by a nondeterministic Büchi automaton.

Theorem 4.3. *For every deterministic Büchi automaton $\mathcal{A}$, there exists a Büchi automaton $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}') = \Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$.*

Proof. Let $\mathcal{A}'$ be constructed from the given deterministic Büchi automaton $\mathcal{A}$ (which we assume, w.l.o.g., to be complete) by Construction 4.1. We prove that $\mathcal{L}(\mathcal{A}') = \Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$.

$\mathcal{L}(\mathcal{A}') \subseteq \Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$: For every word $\alpha \in \mathcal{L}(\mathcal{A}')$ we have an accepting run

$$r' \colon (q_0,0)(q_1,0)\ldots(q_j,0)(q'_0,1)(q'_1,1)\ldots$$

on $\mathcal{A}'$. Hence, $r \colon q_0q_1q_2\ldots q_jq'_0q'_1\ldots$ is the unique run of $\mathcal{A}$ on α. Since $q'_0, q'_1, \ldots \in Q \setminus F$, we have that $\mathrm{Inf}(r) \subseteq Q \setminus F$. Hence, r is not accepting and $\alpha \in \Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$.

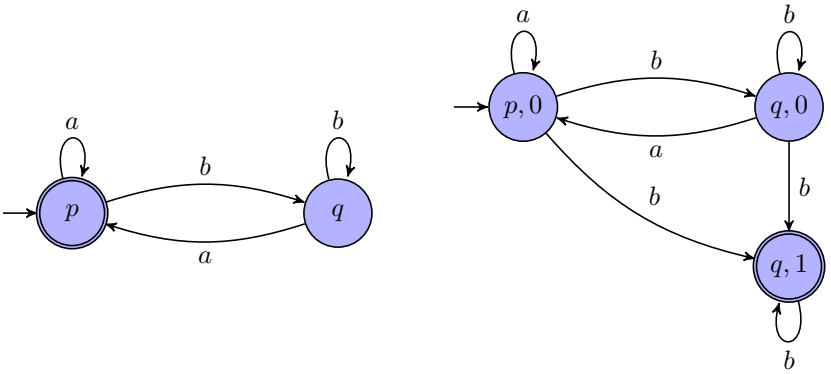
$\mathcal{L}(\mathcal{A}') \supseteq \Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$: Let $\alpha \notin \mathcal{L}(\mathcal{A})$ be some word that is not in the language of $\mathcal{A}$. Since $\mathcal{A}$ is complete and deterministic, there exists a unique run $r \colon q_0q_1q_2\ldots$ of $\mathcal{A}$ on α and

$\mathrm{Inf}(r) \cap F = \varnothing$. Thus, there exists a $k \in \mathbb{N}$ such that $q_j \notin F$ for all $j > k$. This gives us the run

$$r' \colon (q_0, 0)(q_1, 0) \dots (q_k, 0)(q_{k+1}, 1)(q_{k+2}, 1) \dots$$

of $\mathcal{A}'$ on α with $\mathrm{Inf}(r') \subseteq ((Q \setminus F) \times \{1\}) = F'$. Hence, r' is accepting and therefore $\alpha \in \mathcal{L}(\mathcal{A}')$. □

Example 4.1. We use Construction 4.1 to complement the deterministic Büchi automaton $\mathcal{A}$ with language $(b^*a)^\omega$. The result is the nondeterministic Büchi automaton $\mathcal{A}'$ with language $(a + b)^*b^\omega$.



Note that, since not all ω-regular languages can be recognized by deterministic Büchi automata, Construction 4.1 does not provide us with a complementation construction for all ω-regular languages. Such a general construction will be the subject of the following section.

5 Complementation of Büchi Automata

Complementation is an important operation in verification. In particular we can, as discussed in the introduction, reduce the question whether the language of an automaton $\mathcal{A}_1$, representing the system, is contained in the language of an automaton $\mathcal{A}_2$, representing the specification, to the language emptiness problem of an automaton that recognizes the intersection of the language of $\mathcal{A}_1$ with the complement of the language of $\mathcal{A}_2$. While the closure of the Büchi-recognizable languages under complementation has been known since the 1960s, improvements to the complementation construction were made until around 2009. The construction discussed in the following is due to Kupferman and Vardi.

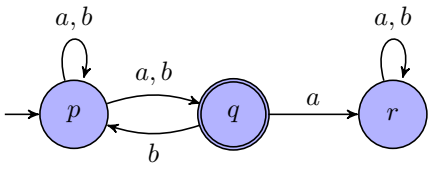
In order to determine whether a given word is in the complement of the language of a nondeterministic automaton, we must check whether *all* runs of the automaton are rejecting. We begin by representing the set of all runs as an infinite directed acyclic graph (DAG):

Definition 5.1. Let $\mathcal{A} = (\Sigma, Q, I, T, \textsc{Büchi}(F))$ be a nondeterministic Büchi automaton. The *run DAG* of $\mathcal{A}$ on a word $\alpha \in \Sigma^\omega$ is the directed acyclic graph $G = (V, E)$ where

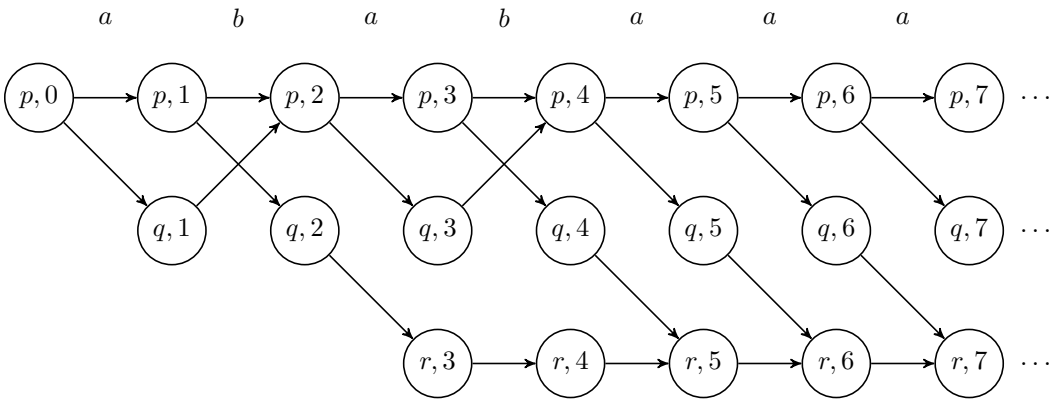
- $V = \bigcup_{i \geq 0} (Q_i \times \{i\})$ where $Q_0 = I$ and $Q_{i+1} = \bigcup_{\substack{q \in Q_i, \\ (q, \alpha(i), q') \in T}} \{q'\}$
- $E = \{((q, i), (q', i+1)) \mid i \geq 0,\ (q, \alpha(i), q') \in T\}$

A path in a run DAG is *accepting* iff it visits $F \times \mathbb{N}$ infinitely often. The automaton accepts a word $\alpha \in \Sigma^\omega$ iff some path in the run DAG on α is accepting.

Example 5.1. Consider the following nondeterministic Büchi automaton $\mathcal{A}$. The language of $\mathcal{A}$ consists of all infinite words over $\{a, b\}$ with infinitely many bs, i.e., $(a^*b)^\omega$:



The run DAG of $\mathcal{A}$ on $\alpha = ababaaa\ldots$ is shown below. Note that starting with level 1, every level of the graph includes the accepting state q, indicating that for every number $n \geq 1$, there exists a run that visits q in the nth position. However, since α is not in the language of $\mathcal{A}$, there is no run that visits q infinitely often.



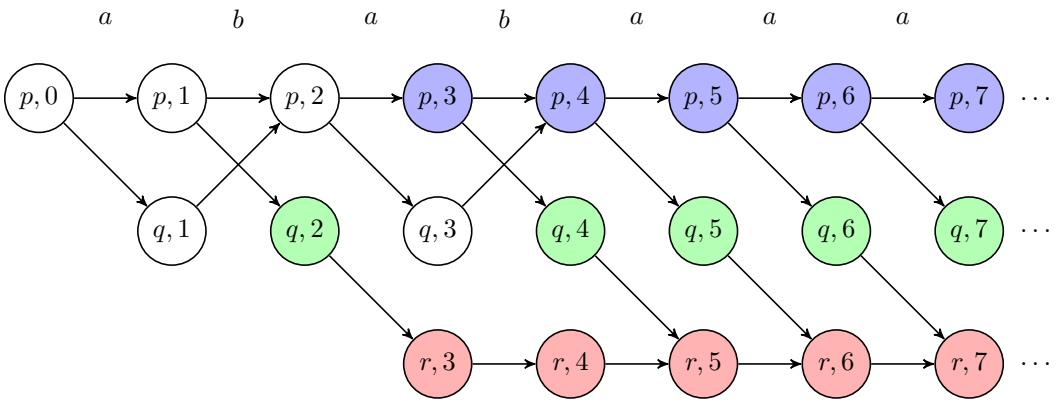
To prove that none of the paths in a given run DAG is accepting, we use an annotation of the vertices of the run DAG with natural numbers, called a *ranking*. The numbers assigned by the ranking come from a finite set, whose size is derived from the number of states of the automaton. We require that vertices that are labeled with odd numbers do not contain accepting states and that the rank never increases along paths in the run DAG. If furthermore all paths eventually stay in some odd rank, then we call the ranking *odd*.

Definition 5.2. A *ranking* for a run DAG G is a function $f\colon V \to \{0, \ldots, 2|Q|\}$ such that

- for all $(q, i) \in V$ where $f(q, i)$ is odd, we have that $q \notin F$, and
- for all $((q, i), (q', i')) \in E$, we have that $f(q', i') \leq f(q, i)$.

A ranking is *odd* iff for all paths $(q_0, i_0)(q_1, i_1)(q_2, i_2)\ldots$ in G, there is a position $n \in \mathbb{N}$ such that $f(q_n, i_n)$ is odd and, for all $j \geq 0$, $f(q_{n+j}, i_{n+j}) = f(q_n, i_n)$.

Example 5.2. Continuing our example, we annotate the run DAG with an odd ranking. The ranks are displayed with the following colors: rank 1 - rank 2 - rank 3 - rank 4.



Clearly, the existence of an odd ranking implies that there is no accepting run.

Lemma 5.1. *If there exists an odd ranking for the run DAG G of a nondeterministic Büchi automaton $\mathcal{A}$ on a word $\alpha \in \Sigma^\omega$, then $\mathcal{A}$ does not accept α.*

Proof. Consider that by the definition of an odd ranking, every path eventually gets trapped in some odd rank, and if $f(q, i)$ is odd, then $q \notin F$. We conclude that every path of G visits F only finitely often. □

The more complicated argument is to show that if there is no accepting run, then there exists an odd ranking. We begin by introducing some helpful terminology. Let G' be a subgraph of G. We call a vertex (q, i)

- *safe* in G', if for all vertices (q', i') reachable from (q, i), we have that $q' \notin F$, and
- *endangered* in G', if only finitely many vertices are reachable from (q, i).

We now define a sequence $G_0 \supseteq G_1 \supseteq G_2 \supseteq \ldots$ of subgraphs of G inductively as follows:

- $G_0 = G$

- $G_{2j+1} = G_{2j} \setminus \{(q,i) \mid (q,i) \text{ is endangered in } G_{2i}\}$
- $G_{2j+2} = G_{2j+1} \setminus \{(q,i) \mid (q,i) \text{ is safe in } G_{2i+1}\}$

Example 5.3. Consider again the run DAG from our previous examples with the coloring from Example 5.2. G_0 is the entire graph. There are no endangered vertices in G_0, hence $G_1 = G_0$. The red vertices are safe in G_1; hence, G_2 consists of all vertices except the red vertices. In G_2, the green vertices are endangered; hence, G_3 consists of all vertices except the red and green vertices. In G_3, the blue vertices are safe; leaving only the non-colored vertices for G_4. G_4 is finite, hence all remaining vertices are endangered. G_5 and all further subgraphs thus are empty.

In the example, G_5 and all further subgraphs are empty. In general, if $\mathcal{A}$ has n states, then G_{2n+1} is empty. We establish this fact with the inductive argument of the following lemma:

Lemma 5.2. *If $\mathcal{A}$ does not accept α, then the following holds: For every $n \leq 2|Q|$ there exists an $\ell_n \in \mathbb{N}$ such that for all $i \geq \ell_n$ we have at most $|Q| - n$ vertices of the form $(\_, i)$ in G_{2n}.*

Proof. We prove the lemma by induction on $n \in \mathbb{N}$. The case of $n = 0$ is straightforward, as every subgraph of G has at most $|Q|$ vertices of the from $(\_, i)$. So, let n be greater than 0.

First assume that $G_{2(n-1)}$ is finite. Then every vertex in $G_{2(n-1)}$ is endangered and correspondingly, both G_{2n-1} and G_n are empty.

Next, assume that $G_{2(n-1)}$ is infinite. Then there must exist a safe vertex (q, m) in G_{2n-1}, as otherwise, we can construct a path within G containing infinitely many visits to F. We choose $\ell_n = m$ and claim that for all $j \geq \ell_n$, there are at most $|Q| - n$ many vertices of the form $(\_, j)$ in G_{2n}. Our claim is proven as follows:

Since $(q, \ell_n) \in G_{2n-1}$, we know that it is not endangered in $G_{2(n-1)}$. Hence, there are infinitely many vertices reachable from (q, ℓ_n) in $G_{2(n-1)}$. It follows by König's Lemma, that there exists an infinite path $p = (q, \ell_n)(q_1, \ell_n + 1)(q_0, \ell_n + 2) \dots$ in $G_{2(n-1)}$. Given that the path is infinite, clearly no vertex of p is endangered in $G_{2(n-1)}$. Thus, p is also a path of G_{2n-1}. Since (q, ℓ_n) is safe in G_{2n-1}, all vertices on p are safe in G_{2n-1} as well. Hence, by the construction of G_{2n}, we can conclude that none of the vertices of p are in G_{2n}. Thus, for all $j \geq \ell_n$, the number of vertices of the form $(\_, j)$ in G_{2n} is strictly smaller than in $G_{2(n-1)}$, concluding the proof. □

With these preparations, we are now ready to show that the existence of an odd ranking is not only sufficient for the absence of an accepting run, but also necessary.

Lemma 5.3. *If $\mathcal{A}$ does not accept α, then there exists an odd ranking for the run DAG of $\mathcal{A}$ on α.*

Proof. We define the function f by $f(q,i) = 2n$ if the vertex (q,i) is endangered in G_{2n} and by $f(q,i) = 2n+1$ if (q,i) is safe in G_{2n+1}. We claim that f is an odd ranking:

By Lemma 5.2, G_j is empty for $j > 2|Q|$. Hence, $f\colon V \to \{0, \dots, 2|Q|\}$. Furthermore:

- For all successors (q', i') of a vertex (q, i) it holds that $f(q', i') \leq f(q, i)$:

  Let $j = f(q, i)$. Then if j is even, the vertex (q, i) is endangered in G_j. Hence, either (q', i') is not in G_j, and therefore $f(q', i') \leq j$, or (q', i') is in G_j, and thus endangered,

i.e., $f(q', i') = j$. Analogously, if j is odd, then the vertex (q, i) is safe in G_j such that either (q', i') is not in G_j, and therefore $f(q, i) < j$, or (q', i') is in G_j, and thus safe, i.e. $f(q, i) = j$.

- f is odd:

  For every path $(q_0, i_0)(q_1, i_1)(q_2, , i_2)\dots$ in G there exists an $n \in \mathbb{N}$ such that for all $j \geq n$ we have that $f(q_j, i_j) = f(q_n, i_n)$. Now, suppose that $k = f(q_n, i_n)$ is even. Then, (q_n, i_n) is endangered in G_k. Furthermore, all (q_j, i_j) are in G_k, since $f(q_j, i_j) = k$ for all $j \in \mathbb{N}$. However, this contradicts that (q_n, i_n) is endangered in G_k such that k must be odd. □

In our complementation construction, we build an automaton that constructs the run DAG level by level and nondeterministically "guesses" the odd ranking. A *level ranking* is a function $g\colon Q \to \{0, \dots, 2|Q|\} \cup \{\bot\}$ such that if $g(q)$ is odd, then $q \notin F$. The special element $\bot$ indicates that the state is not present in the level of the run DAG.

Let $\mathcal{R}$ be the set of all level rankings. We say that a level ranking g' *covers* a level ranking g if for all $q, q' \in Q$, where $g(q) \neq \bot$ and $(q, \sigma, q') \in T$, it holds that $\bot \neq g'(q') \leq g(q)$.

Construction 5.1. For a given Büchi automaton $\mathcal{A} = (\Sigma, Q, I, T, \text{BÜCHI}(F))$ we construct a Büchi automaton $\mathcal{A}' = (\Sigma, Q', I', T', \text{BÜCHI}(F'))$ with $\mathcal{L}(\mathcal{A}') = \Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$ as follows.

- $Q' = \mathcal{R} \times 2^Q$
- $I' = \{(g_0, \varnothing) \mid g_0 \in \mathcal{R},\ g_0(q) = \bot \text{ iff } q \notin I\}$
- $T' = \{((g, \varnothing), \sigma, (g', P')) \mid g' \text{ covers } g,\ P' = \{q' \in Q \mid g'(q') \text{ is even}\}\} \cup$
  $\{((g, P), \sigma, (g', P')) \mid P \neq \varnothing,\ g' \text{ covers } g,$
  $P' = \{q' \in Q \mid (q, \sigma, q') \in T,\ q \in P, g'(q') \text{ is even}\}\}$
- $F' = \mathcal{R} \times \{\varnothing\}$

Intuitively, the first component of each state identifies the level ranking, and the second component tracks the states whose corresponding vertices in the run DAG have even ranks. Paths that traverse such vertices should eventually reach a vertex with an odd rank and the acceptance condition ensures that all paths visit a vertex with odd rank infinitely often.

Theorem 5.1. *For every Büchi automaton $\mathcal{A}$ there exists a Büchi automaton $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}') = \Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$.*

Proof. $\mathcal{L}(\mathcal{A}') \subseteq \Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$: Let $\alpha \in \mathcal{L}(\mathcal{A}')$ and let $r' = (g_0, P_0)(g_1, P_1)\dots$ be an accepting run of $\mathcal{A}'$ on α. Further, let $G = (V, E)$ be the run DAG of $\mathcal{A}$ on α. Then the function $f\colon (q, i) \to g_i(q)$ for $q \in Q_i$ and $i \in \mathbb{N}$ is a ranking for G, because if $g_i(s)$ is odd then $s \notin F$ and for every $((q, i), (q', i+1)) \in E$ it holds that $g_{i+1}(q') \leq g_i(q)$.

We claim that f is an odd ranking for G. By way of contradiction assume that it is not. Then there is a path $(q_0, i_0)(q_1, i_1)(q_2, i_2)\dots$ in G such that for infinitely many $n \in \mathbb{N}$ the rank $f(q_n, i_n)$ is even. Hence, there exists an index $m \in \mathbb{N}$, such that $f(q_m, i_m)$ is even and for all $j \geq m$ we have that $f(q_j, i_j) = f(q_n, i_n)$. Now, since r' is accepting, it follows that $P_k = \varnothing$ for infinitely many k. Let k' be the smallest such index greater or equal than n. However, then also $P_j \neq \varnothing$ for all $j > k'$, yielding the desired contradiction.

We conclude that $\alpha \notin \mathcal{L}(\mathcal{A})$, as there is an odd ranking.

$\Sigma^\omega \setminus \mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$: Let $\alpha \notin \mathcal{L}(\mathcal{A})$ and let $G = (V, E)$ be the run DAG of $\mathcal{A}$ on α. Then there exists an odd ranking f on G and there is a run $r' = (g_0, P_0)(g_1, P_1)\dots$ of $\mathcal{A}'$ on α, where

- $g_i(q) = \begin{cases} f(q,i) & \text{if } q \in Q_i \\ \bot & \text{otherwise,} \end{cases}$
- $P_0 = \varnothing$, and
- $P_{i+1} = \begin{cases} \{q \in Q \mid g_{i+1}(q) \text{ is even}\} & \text{if } P_i = \varnothing \\ \{q' \in Q \mid \exists q \in Q_i \cap P_i.\, ((q,i),(q',i+1)) \in E,\, g_{i+1}(q') \text{ is even}\} & \text{otherwise.} \end{cases}$

The run r' is accepting, as otherwise there would be an index n such that $P_j \neq \varnothing$ for all $j \geq n$, yielding a path in G that visits only finitely often an odd rank. Hence, we conclude that $\alpha \in \mathcal{L}(\mathcal{A}')$. □

6 McNaughton's Theorem

We already established that while the languages that can be recognized with nondeterministic Büchi automata are exactly the ω-regular languages, the languages that can be recognized with *deterministic* Büchi automata are a strictly smaller set. We now repair this deficiency with a more expressive acceptance condition, the *Muller condition*. McNaughton's theorem states that the set of languages recognizable by *deterministic* Muller automata are again exactly the ω-regular languages. We will see later that it is very useful to have a deterministic automaton for a given ω-language, for example in synthesis, where we construct the game between the system and the environment from a deterministic automaton that recognizes the winning plays for the system player. Since the complementation of deterministic Muller automata is a very simple operation, McNaughton's theorem also provides an alternative proof for the result of the previous section that the ω-regular languages are closed under complementation.

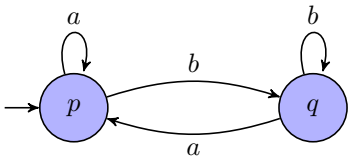
6.1 The Muller Acceptance Condition

Definition 6.1. The *Muller condition* $\textsc{muller}(\mathcal{F})$ on as set of sets of states $\mathcal{F} \subseteq 2^Q$ is the set

$$\textsc{muller}(\mathcal{F}) = \{\alpha \in Q^\omega \mid \mathrm{Inf}(\alpha) \in \mathcal{F}\}$$

An automaton $\mathcal{A} = (\Sigma, Q, I, T, Acc)$ with $Acc = \textsc{muller}(\mathcal{F})$ is called a *Muller automaton*. The set $\mathcal{F}$ is called the set of *accepting subsets* (or the *table*) of $\mathcal{A}$.

Example 6.1. Consider the deterministic automaton over the alphabet $\Sigma = \{a, b\}$ shown below. For the table $\mathcal{F} = \{\{q\}\}$ we obtain the Muller automaton $\mathcal{A}$ recognizing the language $\mathcal{L}(\mathcal{A}) = (a + b)^* b^\omega$, for $\mathcal{F}' = \{\{q\}, \{p, q\}\}$ we obtain the Muller automaton $\mathcal{A}'$ recognizing $\mathcal{L}(\mathcal{A}') = (a^* b)^\omega$.



As a first exercise, we translate Büchi automata into Muller automata:

Construction 6.1. For a (deterministic) Büchi automaton $\mathcal{A} = (\Sigma, Q, I, T, \textsc{büchi}(F))$ we define the (deterministic) Muller automaton $\mathcal{A}' = (\Sigma, Q, I, T, \textsc{muller}(\mathcal{F}))$ using

$$\mathcal{F} = \{S \subseteq Q \mid S \cap F \neq \varnothing\}$$

Since the construction does not modify the transitions, the Muller automaton is again deterministic if the Büchi automaton is deterministic. It is straightforward to see that the automata recognize the same language.

Theorem 6.1. *For every (deterministic) Büchi automaton $\mathcal{A}$, there is a (deterministic) Muller automaton $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.*

Proof. The automaton $\mathcal{A}'$ of Construction 6.1 complies with our requirements, since

$$\textsc{büchi}(F) = \{\alpha \in Q^\omega \mid \mathrm{Inf}(\alpha) \cap F \neq \varnothing\} = \{\alpha \in Q^\omega \mid \mathrm{Inf}(\alpha) \in \mathcal{F}\} = \textsc{muller}(\mathcal{F}) \qquad \square$$

A slightly more difficult construction is to translate the Muller automaton back into a Büchi automaton.

Construction 6.2. Let $\mathcal{A} = (\Sigma, Q, I, T, \text{MULLER}(\{F_1, \ldots, F_n\}))$ be a Muller automaton and $<$ some arbitrary total order on Q. We construct the Büchi automaton $\mathcal{A}' = (\Sigma, Q', I', T', \text{BÜCHI}(F'))$ with $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$ as follows:

- $Q' = Q \cup \bigcup_{i=1}^{n} (\{i\} \times F_i \times F_i)$
- $I' = I$
- $T' = T \cup \{(q, \sigma, (i, q', q')) \mid 1 \leq i \leq n,\ (q, \sigma, q') \in T,\ q' \in F_i\}$
  $\cup \{((i, q, p), \sigma, (i, q', p')) \mid 1 \leq i \leq n,\ (q, \sigma, q') \in T,$
  $$p' = \begin{cases} p & \text{if } q \neq p \\ \min(F_i) & \text{if } q = p = \max(F_i) \\ \min(F_i \setminus \{r \mid r \leq p\}) & \text{if } q = p < \max(F_i), \end{cases}$$
  $q, p, q' \in F_i\}$
- $F' = \bigcup_{i=1}^{n} (\{i\} \times \{\min(F_i)\} \times \{\min(F_i)\})$

A run of the Büchi automaton first simply simulates (while in states Q) the Muller automaton and then "guesses" the accepting subset of the Muller automaton. The purpose of the order $<$ on the states is that we can "step" through the states of the accepting subset in order to make sure that all states in the accepting subset actually occur infinitely often. In states $\{i\} \times F_i \times F_i$, the first component records the index i of the accepting subset, the second component the currently visited state of the Muller automaton, and the third component the "next" state (according to the order on the states) we need to see in order to make progress towards accepting the input word. The Büchi automaton accepts if we step through the states of the accepting subset infinitely often. Recall that we used a similar trick in the construction of the Büchi automaton for the intersection of two Büchi-recognizable languages in Construction 3.2.

Theorem 6.2. *For every Muller automaton $\mathcal{A}$ there is a Buchi automaton $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.*

Proof. We show that the Büchi automaton $\mathcal{A}'$ of Construction 6.2 and $\mathcal{A}$ are equivalent.

$\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$: Let $\alpha \in \mathcal{L}(\mathcal{A})$ and $r = q_0 q_1 q_2 \ldots$ be an accepting run of $\mathcal{A}$ on α. As r is accepting, we have that $\text{Inf}(r) \in \mathcal{F}$, i.e., there is some $1 \leq i \leq n$ such that $\text{Inf}(r) = F_i$. Let m the first position such that $q_j \in \text{Inf}(r)$ for all $j \geq m$ and consider some run

$$r' = q_0 q_1 \ldots q_{m-1} (i, q_m, p_0)(i, q_{m+1}, p_1)(i, q_{m+2}, p_2) \ldots$$

of $\mathcal{A}'$ on α, which nondeterministically switches to (i, q_m, p_0) at position m. For the sake of contradiction assume that r is not accepting. Then there is a position $k \geq 0$ such that $p_j = p_k$ for all $j \geq k$. Then also $q_{m+j} \neq p_j$ for all $j \geq k$. However, this contradicts that $p_k \in F_i$.

$\mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}(\mathcal{A})$: Let $\alpha \in \mathcal{L}(\mathcal{A}')$ and

$$r' = q_0 q_1 \ldots q_{m-1} (i, q_m, p_0)(i, q_{m+1}, p_1)(i, q_{m+2}, p_2) \ldots$$

be some accepting run of $\mathcal{A}'$ on α, which has to switch to some (i, q_m, p_0) at some position m, as otherwise it would not be accepting. By construction $q_j \in F_i$ for all $j \geq m$ and for each

$p \in F_i$ there are infinitely many positions k such that $q_k = p_k = p$. Thus, $\mathrm{Inf}(pr_2(r)) = F_i$, yielding an accepting run $r' = q_0 q_1 q_2 \dots$ of $\mathcal{A}$ on α. □

We now show that deterministic Muller automata are closed, like nondeterministic Büchi automata, under the Boolean operations (complementation, union, and intersection).

Construction 6.3. For a deterministic Muller automaton $\mathcal{A} = (\Sigma, Q, I, T, \textsc{muller}(\mathcal{F}))$ we construct the deterministic Muller automaton $\mathcal{A}^C = (\Sigma, Q, I, T, \textsc{muller}(2^Q \setminus \mathcal{F}))$ with $\mathcal{L}(\mathcal{A}^C) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$.

We use the function pr_n for $n \in \mathbb{N}$ to project to the $(n-1)$-th component of a tuple of arbitrary length. If the tuple has no such component, pr_n is undefined. The first component is accessed using pr_0, and for sets we have that $pr_n(S) = \bigcup_{s \in S} pr_n(s)$.

Construction 6.4. For Muller automata $\mathcal{A}_1 = (\Sigma, Q_1, I_1, T_1, \textsc{muller}(\mathcal{F}_1))$ and $\mathcal{A}_2 = (\Sigma, Q_2, I_2, T_2, \textsc{muller}(\mathcal{F}_2))$ over the same alphabet Σ, we construct the Muller automaton $\mathcal{A}_\cap = (\Sigma, Q_1 \times Q_2, I_1 \times I_2, T_\cap, \textsc{muller}(\mathcal{F}_\cap))$ with $\mathcal{L}(\mathcal{A}_\cap) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$ and where $\mathcal{A}_\cap$ is deterministic if $\mathcal{A}_1$ and $\mathcal{A}_2$ are deterministic, as follows:

- $T_\cap = \{((q_1, q_2), \sigma, (q_1', q_2')) \mid (q_1, \sigma, q_1') \in T_1, (q_2, \sigma, q_2') \in T_2\}$
- $\mathcal{F}_\cap = \{P \subseteq Q_1 \times Q_2 \mid pr_0(P) \in \mathcal{F}_1,\ pr_1(P) \in \mathcal{F}_2\}$

Theorem 6.3. *The languages recognizable by deterministic Muller automata are closed under Boolean operations (complementation, union, intersection).*

Proof. Deterministic Muller automata are closed under complementation: For a deterministic Muller automaton $\mathcal{A}$, the automaton $\mathcal{A}'$ of Construction 6.3 recognizes the complement language, because any set $F \notin \mathcal{F}$ has to be in the complement, i.e., $F \in 2^Q \setminus \mathcal{F}$.

Deterministic Muller automata are closed under intersection: For deterministic Muller automata $\mathcal{A}_1$ and $\mathcal{A}_2$, the automaton $\mathcal{A}_\cap$ of Construction 6.4 recognizes the intersection. Let $r^1 = q_0^1 q_1^1 \dots$ and $r^2 = q_0^2 q_1^2 \dots$ be accepting runs of $\mathcal{A}_1$ and $\mathcal{A}_2$ on some α. Then $r = (r_0^1, r_0^2)(r_1^1, r_1^2) \dots$ is an accepting run of $\mathcal{A}_\cap$ on α and vice versa.

Deterministic Muller automata are closed under union: They are closed under complement and intersection, which suffices by De Morgan's law. □

Theorem 6.4. *A language L is recognizable by a deterministic Muller automaton if and only if L is a Boolean combination of languages $\overrightarrow{W}$ where $W \subseteq \Sigma^*$ is regular.*

Proof. "⇐": If W is regular, then $\overrightarrow{W}$ is recognizable by a deterministic Büchi automaton. Hence, $\overrightarrow{W}$ is recognizable by a deterministic Muller automaton. Thus, the boolean combination $\mathcal{L}$ is recognizable by a deterministic Muller automaton.

"⇒": A deterministic Muller automaton $\mathcal{A}$ accepts some word α with a unique run r if for some $F \in \mathcal{F}$ we have that $\mathrm{Inf}(r) = F$. Thus, there is some $F \in \mathcal{F}$ such that for all $q \in F$ we have that $\alpha \in \overrightarrow{W_q}$ and for all $q \notin F$ we have that $\alpha \notin \overrightarrow{W_q}$, where $W_q = \mathcal{L}(\mathcal{A}_q)$ for the finite-word automaton $\mathcal{A}_q = (\Sigma, Q, I, T, \{q\})$. Hence,

$$\alpha \in \bigcup_{F \in \mathcal{F}} \left(\bigcap_{q \in F} \overrightarrow{W_q} \cap \bigcap_{q \notin F} (\Sigma^\omega \setminus \overrightarrow{W_q}) \right).$$ □

6.2 From Nondeterministic to Semi-Deterministic Automata

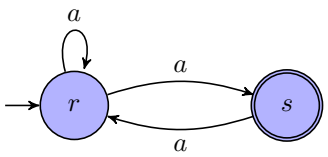
We prove McNaughton's theorem in two steps. In this subsection, we translate nondeterministic Büchi automata into semi-deterministic Büchi automata. In the next subsection, we continue from semi-deterministic Büchi automata to deterministic Muller automata.

A *semi-deterministic* automaton is a (possibly nondeterministic) automaton where all accepting runs ultimately end up in a subset of the states from which all transitions are deterministic.

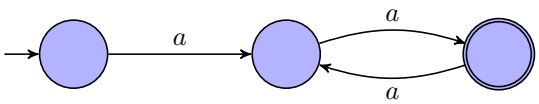
Definition 6.2. A Büchi automaton $\mathcal{A} = (\Sigma, Q, I, T, \textsc{büchi}(F))$ is *semi-deterministic* if $Q = N \uplus D$ is a partition of Q such that $F \subseteq D$, $pr_2(T \cap (D \times \Sigma \times Q)) \subseteq D$, and $(\Sigma, D, \{d\}, T \cap (D \times \Sigma \times D), \textsc{büchi}(F))$ is deterministic for every $d \in D$.

The translation from nondeterministic to semi-deterministic Büchi automata is based on a subset construction where we collect two sets of states: the states that are reachable on the given input word and the states that are reachable on some path trough an accepting state. A state of the semi-deterministic automaton is accepting if the two sets become equal; when this happens, the second set is reinitialized with the subset of accepting states that appear in the first component.

Example 6.2. Consider the following nondeterministic Büchi automaton:



The subset construction results in the following deterministic automaton:

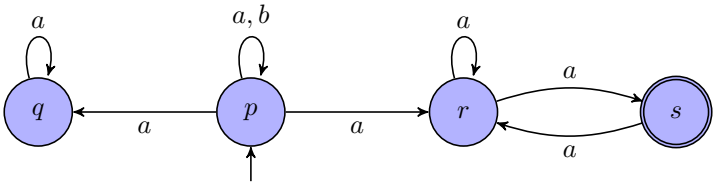


$(\{r\}, \varnothing)$ $(\{r, s\}, \{s\})$ $(\{r, s\}, \{r, s\})$

The subset construction produces a deterministic automaton that accepts a subset of the words accepted by the original automaton. If the two sets are equal infinitely often, we can construct a run of the original automaton that goes through accepting states infinitely often: intuitively, we can go "backwards" from each position where the two sets have become equal and select a path segment for the original automaton where an accepting state is visited (in the proof below we give a more precise argument using König's lemma).

There is no general guarantee that the set of reachable states from some position of an accepting run and the set of states reachable on a path through some accepting state are the same. This is illustrated by the following example.

Example 6.3. Consider the following automaton:



Let the input word be a^ω. From the initial position of some run, which starts in the initial state p, all states are reachable, but only r, and s are reachable on paths through s.

Ultimately, however, every accepting run must reach (and remain in) positions where the set of reachable states and the set of states reachable on a path through some accepting state are the same. This is because the set of reachable states can only become smaller finitely often; hence, at some point, the set of reachable states will remain the same from all subsequent positions, including those (future) positions of the accepting run where the run visits an accepting state.

In our semi-deterministic automaton, we therefore start by simulating the given nondeterministic automaton. At any point we allow a nondeterministic transition into the (from then on) deterministic subset construction.

Construction 6.5. For a Büchi automaton $\mathcal{A} = (\Sigma, Q, I, T, \text{BÜCHI}(F))$, we construct the semi-deterministic Büchi automaton $\mathcal{A}' = (\Sigma, Q', I', T', \text{BÜCHI}(F'))$ with $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$ as follows:

- $Q' = Q \uplus (2^Q \times 2^Q)$
- $I' = I$
- $T' = T \cup \{(q, \sigma, (\{q'\}, \varnothing)) \mid (q, \sigma, q') \in T\}$
  $\cup \{((L_1, L_2), \sigma, (L_1', L_2')) \mid L_1 \neq L_2,$
  $L_1' = pr_2(T \cap L_1 \times \{\sigma\} \times Q),$
  $L_2' = pr_2(T \cap L_1 \times \{\sigma\} \times F) \cup pr_2(T \cap L_2 \times \{\sigma\} \times Q)\}$
  $\cup \{((L_1, L_2), \sigma, (L_1', L_2')) \mid L_1 = L_2,$
  $L_1' = pr_2(T \cap L_1 \times \{\sigma\} \times Q),$
  $L_2' = pr_2(T \cap L_1 \times \{\sigma\} \times F)\}$
- $F' = \{(L, L) \mid L \neq \varnothing\}$

Lemma 6.1. *For every Büchi automaton $\mathcal{A}$ there exists a semi-deterministic Büchi automaton $\mathcal{A}'$ with $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.*

Proof. We show that the Büchi automaton $\mathcal{A}'$ of Construction 6.5 and $\mathcal{A}$ are equivalent.

$\mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}(\mathcal{A})$: Let $\alpha \in \mathcal{L}(\mathcal{A}')$ and let $r' = q_0 q_1 \dots q_{n-1}(L_n, L_n')(L_{n+1}, L_{n+1}') \dots$ be an accepting run of $\mathcal{A}'$ on α. Since r' is accepting, there is an infinite sequence $i_0 i_1 \dots$ of indices such that $i_0 = n$, and, for all $j \geq 1$, $L_{i_j} = L_{i_j}'$ and $L_{i_j} \neq \varnothing$. For every $j \geq 1$, and every $q' \in L_{i_j}$ there exists a state $q \in L_{i_{j-1}}$ and a sequence $q = q_{i_{j-1}}, q_{i_{j-1}+1}, \dots, q_{i_j} = q'$ such that $(q_k, \alpha(k), q_{k+1}) \in T$ for all $k \in \{i_{j-1}, \dots, i_j - 1\}$ and $q_k \in F$ for some $k \in \{i_{j-1} + 1, \dots, i_j\}$. We use the following notation: $\mathit{predecessor}(q', i_j) := q$, $\mathit{run}(q', i_0) = q_0 q_1 \dots q_{n-1} q'$ for $L_{i_0} = \{q'\}$, and $\mathit{run}(q', i_j) = (q_{i_{j-1}+1})(q_{i_{j-1}+2}) \dots q_{i_j}$, for $j \geq 1$.

Now consider the $\left(\bigcup_{j \in \mathbb{N}} L_{i_j} \times \{j\}\right)$-labeled tree where the root is labeled with $(q', 0)$ for $L_{i_0} = \{q'\}$, and the parent of each node with a label (q', j) is labeled with $(\mathit{predecessor}(q', i_j), j - 1)$. The tree is infinite and finite-branching, and, hence, by König's Lemma, has an infinite branch $(q_{i_0}, i_0), (q_{i_1}, i_1), \dots$, corresponding to an accepting run of $\mathcal{A}$:

$$\mathit{run}(q_{i_0}, i_0) \cdot \mathit{run}(q_{i_1}, i_1) \cdot \mathit{run}(q_{i_2}, i_2) \cdot \dots$$

$\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$: Let $\alpha \in \mathcal{L}(\mathcal{A})$ and let $r = q_0, q_1, \dots$ be an accepting run of $\mathcal{A}$ on α. Let $i \in \mathbb{N}$ be an index s.t. $q_i \in F$ and for all $j \geq i$ there exists a $k > j$, such that

$$\{q \in Q \mid q_i \xrightarrow{\alpha[i,k]} q\} = \{q \in Q \mid q_j \xrightarrow{\alpha[j,k]} q\}.$$

The index i exists: "$\supseteq$" holds for all i, because there is a path through q_j. Assume, by way of contradiction, that for all $i \in \mathbb{N}$, there is a $j \geq i$ s.t for all $k > j$ "$\supsetneq$" holds. Then there exists an i' s.t. $\{q \in Q \mid q_{i'} \xrightarrow{\alpha[i',k]} q\} = \varnothing$ for all $k > i'$. Contradiction.
We define a run r' of $\mathcal{A}'$:

$$r' = q_0 \dots q_{i-1}(\{q_i\}, \varnothing)(L_1, L'_1)(L_2, L'_2)\dots$$

where L_j and L'_j are determined by the definition of $\mathcal{A}'$. To prove that r' is accepting, assume otherwise, and let $m \in \mathbb{N}$ be an index such that $L_n \neq L'_n$ for all $n \geq m$.

Then, let $j > m$ be some index with $q_j \in F$; hence $q_j \in L'_j$. There exists a $k > j$ such that $L'_{k+1} = \{q \in Q \mid q_j \xrightarrow{\alpha[j,k]} q\} = \{q \in Q \mid q_i \xrightarrow{\alpha[i,k]} q\} = L_{k+1}$. Contradiction. □

6.3 From Semi-Deterministic Büchi to Deterministic Muller

From the semi-deterministic Büchi automaton we now build a deterministic Muller automaton. The idea of the construction is to continuously simulate, in the deterministic automaton, the nondeterministic part of the semi-deterministic automaton and to "attempt" a transition into the deterministic part whenever possible. In the state of the deterministic automaton we maintain an "array" of states that correspond to these attempts. Along each run of the automaton, there may of course be infinitely many such attempts; we only need a finite array, however, because we do not need to keep track of two different attempts to enter the deterministic part, if they both reach the same state (in this case, we simply track the attempt that entered the deterministic part earlier). We use an array of size $2m$, where m is the number of states of the deterministic part. The factor two allows us to leave a position of the array empty ("$\sqcup$") if an attempt is not continued. This is necessary to distinguish a situation where a previously started attempt failed and, at the same time, a new attempt enters the deterministic part, from the situation where the same attempt ran continuously. The deterministic automaton accepts if there is at least one attempt that runs forever after some point and reaches an accepting state infinitely often.

Construction 6.6. Let $\mathcal{A} = (\Sigma, Q, I, T, \text{BÜCHI}(F))$ be a complete semi-deterministic Büchi automaton with $Q = N \uplus D$ and $m = |D|$, and let $<$ be some arbitrary order on D. We construct the deterministic Muller automaton $\mathcal{A}' = (\Sigma, Q', \{q'_0\}, I', T', \text{MULLER}(\mathcal{F}))$ with $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$ as follows:

- $Q' = 2^N \times (\{0, \dots, 2m\} \to (D \cup \{\sqcup\}))$
- $q'_0 = (N \cap I, (d_1, d_2, \dots, d_n, \sqcup, \dots, \sqcup))$,
  where $d_i < d_{i+1}, \{d_1, \dots, d_n\} = D \cap I\}$.
- $T' = \{((N_1, f_1), \sigma, (N_2, f_2)) \mid$
  $N_2 = pr_2(T \cap N_1 \times \{\sigma\} \times N),\ D' = pr_2(T \cap N_1 \times \{\sigma\} \times D)$
  $g_1(n) = \begin{cases} \sqcup & \text{if } f_1(n) = \sqcup \\ q & \text{if } f_1(n) \xrightarrow{\sigma} q \end{cases}$
  g_2: insort the elements of D' in the empty slots of g_1 (using $<$)
  f_2: delete every recurrence, leaving an *empty* ($\sqcup$) slot
- $\mathcal{F} = \{F' \subseteq Q' \mid \exists i \in 1, \dots, 2m$ s.t.
  $f(i) \neq \sqcup$ for all $(N', f) \in F'$ and
  $f(i) \in F$ for some $(N', f) \in F'\}$.

Lemma 6.2. *For every semi-deterministic Büchi automaton $\mathcal{A}$ there exists a deterministic Muller automaton $\mathcal{A}'$ with $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$.*

Proof. We show that the Büchi automaton $\mathcal{A}'$ of Construction 6.6 and $\mathcal{A}$ are equivalent.

$\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$: Let $\alpha \in \mathcal{L}(\mathcal{A})$, and let $r = n_0 \dots n_{j-1} d_j d_{j+1} d_{j+2} \dots$ be an accepting run of $\mathcal{A}$ on α where $n_k \in N$ for $k < j$ and $d_k \in D$ for $k \geq j$. Let $r' = (N_0, f_0)(N_1, f_1) \dots$ be the run of $\mathcal{A}'$ on α.

We have that $n_k \in N_k$ for all $k < j$, and for all $k \geq j$, $d_k = f_k(i)$ for some $i \leq 2m$. The index i such that $d_k = f_k(i)$ may change during the run, but will never increase, and will, therefore, eventually stabilize. For this stable i, we have that $f(i) \neq \llcorner\!\lrcorner$ for all $(N', f) \in \mathrm{Inf}(r')$ and $f(i) \in F$ for some $(N', f) \in \mathrm{Inf}(r')$. Hence, $\mathrm{Inf}(r') \in \mathcal{F}$.

$\mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}(\mathcal{A})$: Let $\alpha \in \mathcal{L}(\mathcal{A}')$, and let $r' = (N_0, f_0)(N_1, f_1) \dots$ be the accepting run of $\mathcal{A}'$ on α. We pick an index $i \in \{0, \dots 2m\}$ and an accepting subset $F' \in \mathcal{F}$ s.t. $f(i) \neq \llcorner\!\lrcorner$ for all $(N', f) \in F'$ and $f(i) \in F$ for some $(N', f) \in F'$. We furthermore pick an index $j \in \mathbb{N}$ such that $f_n(i) \neq \llcorner\!\lrcorner$ for all $n > j$. There is a run

$$r = q_0 q_1 \dots q_j f_{j+1}(i) f_{j+2}(i) f_{j+3}(i) \dots$$

of $\mathcal{A}$ on α. The run is accepting. □

Combining Lemmas 6.1 and 6.2, we obtain McNaughton's theorem.

Theorem 6.5 (McNaughton's Theorem (1966))**.** *Every Büchi-recognizable language is recognizable by a deterministic Muller automaton.*

6.4 Safra's Construction

Our two-step proof of McNaughton's theorem provides a construction of a deterministic Muller automaton with a doubly exponential number of states (in the number of states of the nondeterministic Büchi automaton). The second exponent can be avoided by a smart combination of the two steps due to Shmuel Safra. In Safra's construction, the states of the Muller automaton are finite trees (known as *Safra trees*), which satisfy the following conditions:

- Each node of the tree is labeled with a set of states of the Büchi automaton, called the *macrostate* of the node.
- The macrostates of siblings are disjoint.
- The union of the macrostates of a set of sister nodes is a proper subset of the macrostate of their parent.
- Each node has a unique name in $\{1, \dots, 2|Q|\}$, where Q are the states of the Büchi automaton. The root is named 1.
- Each node may be marked with an `!`.

In Construction 6.6, the states of the Muller automaton consist of two components (N', f), the set $N' \subseteq N$ of reachable states (where N are states of the nondeterministic part from Construction 6.5, i.e., the states of the original Büchi automaton) and the mapping $f : \{0, \dots, 2m\} \to (D \cup \{\llcorner\!\lrcorner\})$ assigning states from the set D of the deterministic part to certain numbers. Safra's construction implements Construction 6.6 in the sense that N' is the macrostate of the root, and the states from D are also represented as macrostates of

nodes of the tree, which have numbers as unique names. (The precise numbering in Safra's construction differs slightly from the numbering in Construction 6.6).

In Construction 6.5, the states D of the deterministic part consist of pairs (L_1, L_2) of sets of states, where L_1 tracks the set of states that are reachable from the state in which the deterministic part was entered, and L_2 tracks the states that are reachable on some path through an accepting state. In Safra's construction, set L_1 appears as the macrostate of a node, set L_2 is distributed over the children of the node: each child tracks the states reached through a visit to accepting states at a particular position of the word. The situation that L_1 and L_2 become equal, i.e., an accepting state of the semi-deterministic automaton of Construction 6.5 is reached, is identified in Safra's construction by marking the node with the macrostate L_1 with an `!`. As in Construction 6.6, the accepting subsets of the Muller automaton identify particular numbers (here "names") that are present in all states of the accepting subset (and thus come from the same run) and that correspond to an accepting state from D in at least one state.

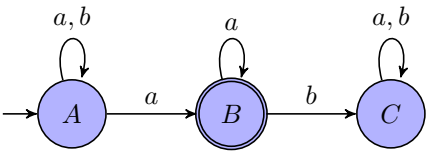
Safra's construction thus cleverly distributes the states of D over the tree, while still being able to recognize an accepting situation. The final trick of the construction is to ensure that the tree remains small, i.e., that the small selection $\{1, \ldots, 2|Q|\}$ of names for the nodes is actually enough. This is ensured through two clean-up operations in the tree, called the horizontal merge and the vertical merge. The *horizontal merge* removes any state from a macrostate (and the macrostate of all descendants) if the state appears in the macrostate of an older sibling. This corresponds to the deletion of recurrences in Construction 6.6. The *vertical merge* removes the descendants of a node when the macrostate becomes equal to the union of the macrostates of the children. In this case, an accepting state of the semi-deterministic automaton has been reached; if this continues to happen infinitely often, the run is accepting, and it is not necessary to analyze the states of the macrostate in more detail.

Construction 6.7. For a Büchi automaton $\mathcal{A} = (\Sigma, Q, I, T, \text{BÜCHI}(F))$, we construct the deterministic Muller automaton $\mathcal{A}' = (\Sigma, Q', \{q_0'\}, I', T', \text{MULLER}(\mathcal{F}))$ with $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$ as follows:

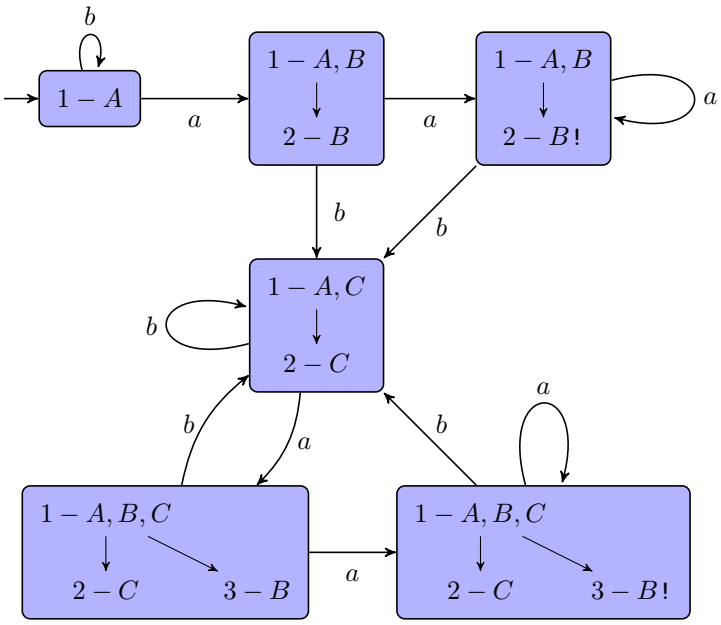
- Q' is the set of Safra trees.
- q_0' is the Safra tree with a single (unmarked) node with macrostate I and name 1.
- T' consists of the transitions that carry out the following steps:
  1. All marks `!` are removed.
  2. For every node with macrostate M, a new child with new macrostate $M' = pr_2(T \cap M \times \{\sigma\} \times F)$, is created. The new nodes get fresh names (in a predefined fashion).
  3. The macrostate M of old nodes is updated to $M' = pr_2(T \cap M \times \{\sigma\} \times Q)$.
  4. (*horizontal merge*): For every node with macrostate M and $q \in M$, remove q from the macrostate of all younger siblings and their descendants.
  5. All nodes with empty macrostate are removed.
  6. (*vertical merge*): For every node n whose macrostate equals the union of the macrostates of its children:
     (a) all its descendants are removed, and
     (b) the node n is marked with an `!`.
- $\mathcal{F} = \{F' \subseteq Q' \mid \exists i \in \{1, \ldots, 2|Q|\}$ s.t. a node named i is in all Safra trees in F', and the node named i is marked with an `!` in some Safra tree in $F'\}$.

The correctness proof of Safra's construction follows the ideas of the proofs of Lemma 6.1 and Lemma 6.2. We therefore skip this proof here and instead conclude with a small example.

Example 6.4. Consider the following nondeterministic Büchi automaton over $\Sigma = \{a, b\}$:



Safra's construction results in the following Muller automaton:



The accepting subsets consist of all subsets that include the state where node 2 is marked and possibly other states where node 2 is present, and all subsets that include the state where node 3 is marked and possibly some other states where node 3 is present.

7 Logics over Infinite Sequences

7.1 Linear-time Temporal Logic (LTL)

Linear-time temporal logic (LTL) is a popular logic for the specification of reactive systems. For a given set AP of *atomic propositions*, formulas of the logic define sets of infinite words over the alphabet 2^{AP}. Typically, the atomic propositions refer to the interface of a system or a component, such as (a boolean representation of) the input and output variables, and the words defined by the formula correspond to executions of the system that are considered correct (cf. Section 1.1). The syntax of LTL is constructed from atomic propositions, the boolean connectives of propositional logic, and the temporal operators $\bigcirc$ and $\mathcal{U}$. In practice, $\mathcal{U}$ is often replaced by the derived temporal operators $\Diamond$ and $\Box$.

- *Propositional logic*
  - $p \in AP$ — atomic proposition
  - $\neg\varphi$ and $\varphi \wedge \psi$ — negation and conjunction
- *Temporal operators*
  - $\bigcirc\varphi$ — next state fulfills φ
  - $\varphi\,\mathcal{U}\,\psi$ — φ holds until a ψ-state is reached
- *Derived operators*
  - $\Diamond\varphi \equiv \mathit{true}\,\mathcal{U}\,\varphi$ — "some time in the future"
  - $\Box\varphi \equiv \neg\Diamond\neg\varphi$ — "from now on forever"

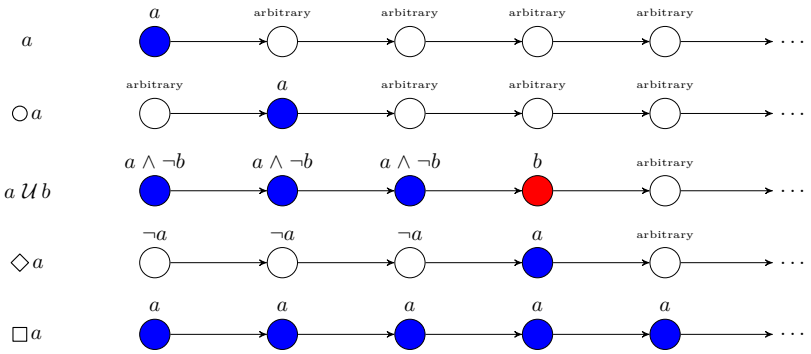
An LTL formula φ over AP defines the following language over the alphabet 2^{AP}:

$$\mathcal{L}(\varphi) \;=\; \Big\{\alpha \in \big(2^{AP}\big)^{\omega} \mid \alpha \models \varphi\Big\},$$

where $\models$ is the smallest relation satisfying:

$$\begin{array}{lcll}
\alpha & \models & p & \text{iff} \quad p \in \alpha(0) \quad (\text{i.e., } \alpha(0) \models p) \\
\alpha & \models & \varphi_1 \wedge \varphi_2 & \text{iff} \quad \alpha \models \varphi_1 \text{ and } \alpha \models \varphi_2 \\
\alpha & \models & \neg\varphi & \text{iff} \quad \alpha \not\models \varphi \\
\alpha & \models & \bigcirc\varphi & \text{iff} \quad \alpha[1,\infty] = \alpha(1)\alpha(2)\alpha(3)\ldots \models \varphi \\
\alpha & \models & \varphi_1\,\mathcal{U}\,\varphi_2 & \text{iff} \quad \exists j \geq 0.\ \alpha[j,\infty] \models \varphi_2 \ \text{ and } \ \alpha[i,\infty] \models \varphi_1 \text{ for all } 0 \leq i < j
\end{array}$$

The following picture illustrates the semantics of the temporal operators.



Typical examples of properties expressible in LTL are "infinitely often p" ($\Box\Diamond p$) and "finitely often p" ($\Diamond\Box\neg p$). Not all ω-regular language can, however, be defined in LTL. For example, the language $(\varnothing\varnothing)^*\{p\}^\omega$, i.e., "an arbitrary but even sequence of $\varnothing$ symbols, followed by an infinite sequence of p symbols", cannot be defined in LTL. This language is an example of a *counting* language. LTL can only define non-counting languages.

Definition 7.1. A language $L \subseteq \Sigma^\omega$ is *non-counting* iff

$$\exists n_0 \in \mathbb{N}.\ \forall n \geq n_0.\ \forall v, w \in \Sigma^*, \alpha \in \Sigma^\omega.\ vw^n\alpha \in L \ \Leftrightarrow\ vw^{n+1}\alpha \in L.$$

Example 7.1. The language $L = (\varnothing\varnothing)^*\{p\}^\omega$ is counting. For every $\varnothing^n\{p\}^\omega \in L$, but $\varnothing^{n+1}\{p\}^\omega \notin L$.

Theorem 7.1. *For every LTL-formula φ, $\mathcal{L}(\varphi)$ is non-counting.*

Proof. We prove the theorem by structural induction on φ:

Case $\varphi = p$:

We choose $n_0 = 1$.

Case $\varphi = \varphi_1 \wedge \varphi_2$:

By induction hypothesis, φ_1 defines non-counting language with threshold $n_0' \in \mathbb{N}$, φ_2 with n_0''. We choose $n_0 = \max\{n_0', n_0''\}$.

Case $\varphi = \neg\varphi_1$:

We choose $n_0 = n_0'$, where n_0' is the threshold of $\mathcal{L}(\varphi_1)$.

Case $\varphi = \bigcirc\varphi_1$:

We choose $n_0 = n_0' + 1$, where n_0' is the threshold of $\mathcal{L}(\varphi_1)$. We show for $n \geq n_0$ that $vw^n\alpha \models \bigcirc\varphi$ if and only if $vw^{n+1}\alpha \models \bigcirc\varphi$:

Case $v \neq \varepsilon$: Thus, $v = av'$ for some $a \in \Sigma, v' \in \Sigma^*$. We have that

$$\begin{aligned} & av'w^n\alpha \models \bigcirc\varphi \\ \text{iff } & v'w^n\alpha \models \varphi \\ \text{iff } & v'w^{n+1}\alpha \models \varphi \qquad \text{(induction hypothesis)} \\ \text{iff } & av'w^{n+1}\alpha \models \bigcirc\varphi. \end{aligned}$$

Case $v = \varepsilon$: Thus, $w = aw'$ for some $a \in \Sigma, w' \in \Sigma^*$. It follows that

$$\begin{aligned} & (aw')^n\alpha \models \bigcirc\varphi \\ \text{iff } & (aw')(aw')^{n-1}\alpha \models \bigcirc\varphi \\ \text{iff } & w'(aw')^{n-1}\alpha \models \varphi \\ \text{iff } & w'(aw')^n\alpha \models \varphi \qquad \text{(induction hypothesis)} \\ \text{iff } & (aw')^{n+1}\alpha \models \bigcirc\varphi. \end{aligned}$$

Case $\varphi = \varphi_1 \,\mathcal{U}\, \varphi_2$:

We choose $n_0 = \max\{n_0', n_0''\} + 1$, where n_0' is the threshold of $\mathcal{L}(\varphi_1)$ and n_0'' is the threshold of $\mathcal{L}(\varphi_2)$. We show that for $n \geq n_0$ it holds that $vw^n\alpha \models \varphi_1 \,\mathcal{U}\, \varphi_2$ if and only if $vw^{n+1}\alpha \models \varphi_1 \,\mathcal{U}\, \varphi_2$.

"$\Rightarrow$": By the semantics of $\mathcal{U}$, we have by $vw^n\alpha \models \varphi_1 \,\mathcal{U}\, \varphi_2$, that there is a j such that $vw^n\alpha[j, \infty] \models \varphi_2$ and for all $i < j$, $vw^n\alpha[i, \infty] \models \varphi_1$. Let j be the least such index.

Case $j \leq |v|$:

By induction hypothesis, $vw^{n+1}\alpha[j,\infty] \models \varphi_2$ and for $i < j$, $vw^{n+1}\alpha[i,\infty] \models \varphi_1$.

Case $j > |v|$:

We have that $vw^{n+1}\alpha[j+|w|,\infty] \models \varphi_2$, because $vw^{n+1}\alpha$ has the same suffix from position $j+|w|$ as $vw^n\alpha$ from position j. Analogously, for each position $|v|+|w| \leq i < j+|w|$, we have that $vw^{n+1}\alpha[i,\infty] \models \varphi_1$. Thus, by induction hypothesis, for all $i < |v|+|w|$ with $i < j$ we get that $vww^n\alpha[i,\infty] \models \varphi_1$, since $vww^{n-1}\alpha[i,\infty] \models \varphi_1$.

"$\Leftarrow$" By the semantics of $\mathcal{U}$, we have that $vw^{n+1}\alpha \models \varphi_1 \,\mathcal{U}\, \varphi_2$ implies that there is a j such that $vw^{n+1}\alpha[j,\infty] \models \varphi_2$ and $\forall i < j.\ vw^{n+1}\alpha[i,\infty] \models \varphi_1$.

Case $j \leq |v|+|w|$:

By induction hypothesis, $vww^{n-1}\alpha[j,\infty] \models \varphi_2$ and for $i < j$, $vww^{n-1}\alpha[i,\infty] \models \varphi_1$.

Case $j > |v|+|w|$:

Since $vw^n\alpha$ has the same suffix from position $j-|w|$ as $vw^{n+1}\alpha$ from position j, we have that $vw^n\alpha[j-|w|,\infty] \models \varphi_2$. Analogously, for all positions $|v|+|w| \leq i < j$ it holds that $vw^n\alpha[i,\infty] \models \varphi_1$. By induction hypothesis, for all $i < |v|+|w|$. $vww^{n-1}\alpha[i,\infty] \models \varphi_1$, because $vww^n\alpha[i,\infty] \models \varphi_1$. □

7.2 Quantified Propositional Temporal Logic (QPTL)

Theorem 7.1 shows that LTL cannot express all ω-regular languages. We repair this deficiency by adding quantification over propositions. QPTL formulas over a set AP of atomic propositions are generated by the following grammar, where $p \in AP$:

$$\psi \quad ::= \quad p \;\mid\; \neg\psi \;\mid\; \psi \wedge \psi \;\mid\; \bigcirc\psi \;\mid\; \Diamond\psi \;\mid\; \exists p.\, \psi$$

The QPTL connectives have the same semantics as in LTL, except for propositional quantification:

$$\alpha \models \exists p.\psi \quad \text{iff} \quad \text{there exists } \alpha' \in (2^{AP})^\omega \text{ such that } \alpha =_{AP \setminus \{p\}} \alpha' \text{ and } \alpha' \models \psi\ ,$$

where $\alpha =_P \alpha'$ for some $P \subseteq AP$ iff, for all $i \in \mathbb{N}$, $\alpha(i) \cap P = \alpha'(i) \cap P$.

An occurrence of an atomic proposition is called *free* if it is not in the scope of a quantifier, and *bound* otherwise. The set of free atomic propositions $AP' \subseteq AP$ consists of those atomic propositions that have a free occurrence. The language of a QPTL formula φ is a language over the alphabet $2^{AP'}$: $\mathcal{L}(\varphi) \;=\; \left\{\alpha \in \left(2^{AP'}\right)^\omega \mid \alpha \models \varphi\right\}$.

Example 7.2. The language $L = (\varnothing\varnothing)^*\{p\}^\omega$ from Example 7.1 is QPTL-definable:
$\exists q.\ (q \wedge \Box(q \leftrightarrow \neg\bigcirc q) \wedge \Box(p \rightarrow \bigcirc p) \wedge \Box(\bigcirc p \rightarrow (p \vee \neg q))) \wedge \Diamond p$

In fact, it is exactly the ω-regular languages that can be expressed in QPTL. The following theorem translates a given Büchi automaton with alphabet 2^{AP} into a QPTL formula that defines the language of the automaton. We postpone the proof that, vice versa, there is a Büchi automaton for every QPTL formula, to a little bit later.

Theorem 7.2. *For every Büchi automaton $\mathcal{A}$ over $\Sigma = 2^{AP}$ there exists a QPTL formula $\varphi_\mathcal{A}$ such that $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{A})$.*

Proof. For a Büchi automaton $\mathcal{A} = (2^{AP}, Q, I, T, \textsc{büchi}(F))$, we define a QPTL formula $\varphi_\mathcal{A}$ with $\mathcal{L}(\varphi_\mathcal{A}) = \mathcal{L}(\mathcal{A})$. Let $Q = \{q_1, q_2, \ldots, q_n\}$; we introduce an auxiliary proposition at_q for each state $q \in Q$. Then $\varphi_\mathcal{A}$ is defined as follows:

$$\varphi_{\mathcal{A}} \;:=\; \exists at_{q_1}, \ldots, at_{q_n}. \qquad \bigvee_{q \in I} at_q$$
$$\wedge \quad \Box\left(\bigvee_{(q,A,q') \in T} at_q \wedge \bigcirc at_{q'} \wedge \left(\bigwedge_{p \in A} p \right) \wedge \left(\bigwedge_{p \in AP \setminus A} \neg p \right) \right)$$
$$\wedge \quad \Box\left(\bigwedge_{i=1}^{n} \bigwedge_{j \neq i} \neg(at_{q_i} \wedge at_{q_j}) \right)$$
$$\wedge \quad \Box\Diamond \bigvee_{q \in F} at_q$$

□

7.3 Monadic Second-Order Logic of One Successor (S1S)

Temporal logics like LTL and QPTL refer to the positions of the input word implicitly through the temporal operators. We now introduce a logic that allows us to manipulate positions directly, though variables that store positions (*first-order variables*) or sets of positions (*second-order variables*) and through a *successor* operation S. The logic is called *Monadic Second-Order Logic of One Successor*, where *monadic* means that the second-order quantification is restricted to unary relations, i.e., sets, and *one successor* means that we only have a single successor operation. Later in the course, we will study monadic second-order logics of two or more successors, which allow us to describe trees rather than words.

Let $V_1 = \{x, y, \ldots\}$ be a set of first-order variables and $V_2 = \{X, Y, \ldots\}$ a set of second-order variables. Then the *terms* of S1S are defined by the following grammar:

$$t ::= 0 \mid x \mid S(t)$$

The *formulas* of S1S are defined by the following grammar:

$$\varphi ::= t \in X \mid t = t \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x.\varphi \mid \exists X.\varphi \;,$$

where $x \in V_1$ is a first-order variable and $X \in V_2$ is a second-order variable. Additionally, we allow the usual boolean connectives and introduce the following abbreviations:

- $\forall X.\ \varphi \;:=\; \neg\exists X.\ \neg\varphi$
- $\forall x.\ \varphi \;:=\; \neg\exists x.\ \neg\varphi$
- $x \notin Y \;:=\; \neg(x \in Y)$
- $x \neq y \;:=\; \neg(x = y)$

The semantics of an S1S formula is given relative to a valuation of the variables. A *first-order valuation* is a function $\sigma_1 : V_1 \to \mathbb{N}$ that assigns to each first-order variable a natural number. A *second-order valuation* is a function $\sigma_2 : V_2 \to 2^{\mathbb{N}}$ that assigns to each second-order variable a set of natural numbers. The value of a *term* is then defined as follows:

- $[0]_{\sigma_1} = 0$
- $[x]_{\sigma_1} = \sigma_1(x)$
- $[S(t)]_{\sigma_1} = [t]_{\sigma_1} + 1$

We again distinguish free and bound occurrences of a variable, and identify the subsets $V_1' \subseteq V_1$ and $V_2' \subseteq V_2$ of free first-order and free second-order variables, respectively. An S1S *formula* φ defines the following language over the alphabet $2^{V_1' \cup V_2'}$:

$$\mathcal{L}(\varphi) = \{\alpha_{\sigma_1,\sigma_2} \in (2^{V_1' \cup V_2'})^\omega \mid \sigma_1, \sigma_2 \models \varphi\},$$

where $x \in \alpha_{\sigma_1,\sigma_2}(j)$ iff $j = \sigma_1(x)$, and $X \in \alpha_{\sigma_1,\sigma_2}(j)$ iff $j \in \sigma_2(X)$, and $\models$ is the smallest relation that satisfies the following:

- $\sigma_1, \sigma_2 \models t \in X$ iff $[t]_{\sigma_1} \in \sigma_2(X)$
- $\sigma_1, \sigma_2 \models t_1 = t_2$ iff $[t_1]_{\sigma_1} = [t_2]_{\sigma_1}$
- $\sigma_1, \sigma_2 \models \neg\psi$ iff $\sigma_1, \sigma_2 \not\models \psi$
- $\sigma_1, \sigma_2 \models \psi_0 \vee \psi_1$ iff $\sigma_1, \sigma_2 \models \psi_0$ or $\sigma_1, \sigma_2 \models \psi_1$
- $\sigma_1, \sigma_2 \models \exists x.\ \varphi$ iff there is an $a \in \mathbb{N}$ s.t.

$$\sigma_1'(y) = \begin{cases} \sigma_1(y) \text{ if } y \neq x \\ a \text{ otherwise} \end{cases}$$

  and $\sigma_1', \sigma_2 \models \varphi$.
- $\sigma_1, \sigma_2 \models \exists X.\ \varphi$ iff there is an $A \subseteq \mathbb{N}$ s.t.

$$\sigma_2'(Y) = \begin{cases} \sigma_2(Y) \text{ if } Y \neq X \\ A \text{ otherwise} \end{cases}$$

  and $\sigma_1, \sigma_2' \models \varphi$

Example 7.3. We give a few example formulas that are often useful to define more complicated properties.

$$\begin{array}{lcl} X \subseteq Y & :\equiv & \forall z.\ (z \in X \to z \in Y); \\ X = Y & :\equiv & X \subseteq Y \wedge Y \subseteq X; \\ \mathit{Suff}(X) & :\equiv & \forall y.\ (y \in X \to S(y) \in X); \\ x \leq y & :\equiv & \forall Z.\ (x \in Z \wedge \mathit{Suff}(Z)) \to y \in Z; \\ \mathit{Fin}(X) & :\equiv & \exists Y.\ (X \subseteq Y \wedge (\exists z.\ z \notin Y) \wedge (\forall z.\ (z \notin Y \to S(z) \notin Y))); \end{array}$$

We already showed, in Theorem 7.2, that every Büchi-recognizable language is QPTL-definable. We now complete a full circle by showing that every QPTL-definable language is S1S-definable, and that every S1S-definable language is Büchi-recognizable. Hence, QPTL, S1S, and Büchi automata are equally expressive.

Theorem 7.3. *Every QPTL-definable language is S1S-definable.*

Proof. For every QPTL-formula φ over AP and every S1S-term t, we define a S1S formula $T(\varphi, t)$ with $V_2 = AP$ such that, for all $\alpha \in (2^{AP})^\omega$,

$$\alpha[[t]_{\sigma_1}, \infty] \models_{\text{QPTL}} \varphi \qquad \text{iff} \qquad \sigma_1, \sigma_2 \models_{\text{S1S}} T(\varphi, t),$$

where $\sigma_2 : P \mapsto \{i \in \mathbb{N} \mid P \in \alpha(i)\}$.

- $T(P, t) = t \in P$, for $P \in AP$
- $T(\neg\varphi, t) = \neg T(\varphi, t)$
- $T(\varphi \wedge \psi, t) = T(\varphi, t) \wedge T(\psi, t)$
- $T(\bigcirc\varphi, t) = T(\varphi, S(t))$
- $T(\Diamond\psi, t) = \exists x.(x \geq t \wedge T(\psi, x))$
- $T(\exists P.\ \varphi, t) = \exists P.\ T(\varphi, t)$

The language of φ is then defined by the S1S formula $T(\varphi, 0)$. □

To prepare for the proof that every S1S-definable language is Büchi-recognizable, we show in the following lemma that we can focus on a restricted sublogic, called $\mathrm{S1S}_0$, which is defined by the following grammar:

$$\varphi ::= \quad 0 \in X \mid x \in Y \mid x = 0 \mid x = y \mid x = S(y) \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x.\ \varphi \mid \exists X.\ \varphi$$

In $\mathrm{S1S}_0$, we only allow variables and 0 in membership tests, and variables, 0, and a single application of the successor operation in equalities. A formula that has more complex terms in such expressions can be simplified by introducing additional variables.

Lemma 7.1. *For every S1S formula φ there is an S1S$_0$ formula φ' such that $\mathcal{L}(\varphi) = \mathcal{L}(\varphi')$.*

Proof. We rewrite a given S1S formula φ into the $\mathrm{S1S}_0$ formula φ' using the following rewrite rules:

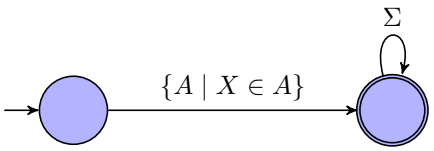
$$\begin{array}{rcl@{\qquad}rcl}
 & & S(t) \in X & \mapsto & \exists y.\ y = S(t) \wedge y \in X \\
0 = x & \mapsto & x = 0 & 0 = 0 & \mapsto & \exists Y.\ 0 \in Y \vee 0 \notin Y \\
S(t) = 0 & \mapsto & 0 = S(t) & 0 = S(t) & \mapsto & \exists y.\ y = S(t) \wedge y = 0 \\
S(t) = x & \mapsto & x = S(t) & t = S(0) & \mapsto & \exists x.\ x = 0 \wedge t = S(x) \\
S(t) = S(t') & \mapsto & t = t' & t = S(S(t')) & \mapsto & \exists y.\ y = S(t') \wedge t = S(y)
\end{array}$$

□

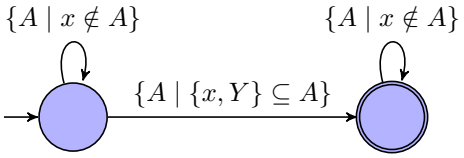
Theorem 7.4. *Every S1S-definable language is Büchi-recognizable.*

Proof. Let φ be an S1S-formula. We construct a Büchi automaton $\mathcal{A}$ with $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{A})$. We begin by translating φ into an equivalent $\mathrm{S1S}_0$ according to Lemma 7.1 and by renaming bound variables to obtain unique variables. The Büchi automaton is then constructed inductively as follows.

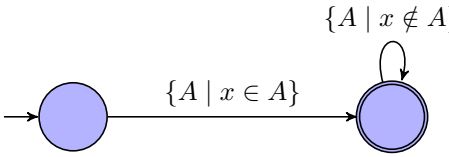
- $0 \in X$:



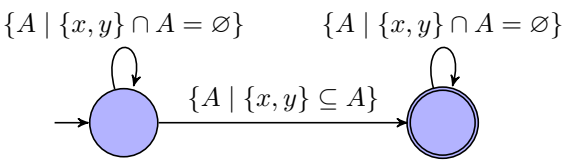
- $x \in Y$:



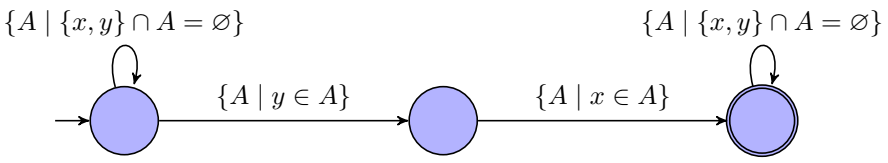
- $x = 0$:



- $x = y$:



- $x = S(y)$:

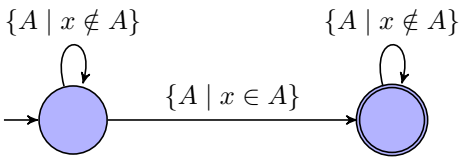


- $\varphi \vee \psi$: let $\mathcal{A}_\varphi$ and $\mathcal{A}_\psi$ be the automata constructed for φ and ψ, respectively. We obtain the automaton for $\varphi \vee \psi$ by constructing the automaton that recognizes the union of $\mathcal{L}(\mathcal{A}_\varphi)$ and $\mathcal{L}(\mathcal{A}_\psi)$.

- $\neg\varphi$: let $\mathcal{A}_\varphi$ be the automaton constructed for φ. We obtain the automaton for $\neg\varphi$ by constructing the automaton that recognizes the complement of $\mathcal{L}(\mathcal{A}_\varphi)$.

- $\exists X.\ \varphi$: let $\mathcal{A}_\varphi$ be the automaton constructed for φ. We obtain the automaton for $\exists X.\ \varphi$ by eliminating X from the input alphabet, i.e., we replace each transition (q, A, q') by $(a, A \setminus \{X\}, q')$.

- $\exists x.\ \varphi$: as for $\exists X.\ \varphi$, except that before the elimination of x, we intersect with the language of the following automaton $\mathcal{A}_x$, which ensures that x appears exactly once:



As the final step, we intersect with $\mathcal{A}_x$ for each free first-order variable x. □

7.4 Weak Monadic Second-Order Logic of One Successor (WS1S)

Weak monadic second-order logic of one successor (WS1S) differs from S1S in that the second-order variables refer to *finite* rather than general, possibly infinite sets. WS1S is often used as a coding language for decidable verification problems, such as parametric hardware or software with pointers. The MONA tool[1] is an efficient BDD-based implementation of a decision procedure for WS1S. The syntax of WS1S is the same as for S1S; the semantics is the same except for quantification:

- $\sigma_1, \sigma_2 \models \exists X.\ \varphi$ iff there is a **finite** $A \subseteq \mathbb{N}$ s.t.

$$\sigma_2'(Y) = \begin{cases} \sigma_2(Y) & \text{if } Y \neq X \\ A & \text{otherwise} \end{cases}$$

  and $\sigma_1, \sigma_2' \models \varphi$.

The following theorem shows that the restriction to finite sets does not affect the expressiveness of the logic.

Theorem 7.5. *A language is WS1S-definable iff it is S1S-definable.*

[1] `http://www.brics.dk/mona`

Proof. "⇒": We use formula $\mathrm{Fin}(X)$ from Example 7.3 to express that a set X is finite. A WS1S formula is translated into an equivalent S1S formula by replacing each quantified subformula $\exists X \varphi$ with $\exists X.\ \mathrm{Fin}(X) \wedge \varphi$.

"⇐": Let φ be an S1S-formula, let V_1 and V_2 be the free first-order and second-order variables of φ, respectively. By Theorem 7.4, there is a Büchi automaton $\mathcal{A}_\varphi$ with $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$, and, by McNaughton's Theorem, a deterministic Muller automaton $\mathcal{A}'_\varphi$ with $\mathcal{L}(\mathcal{A}'_\varphi) = \mathcal{L}(\mathcal{A}_\varphi)$. By the characterization of deterministic Muller languages (Theorem 6.4), $\mathcal{L}(\mathcal{A}')$ is a boolean combination of languages $\overrightarrow{W}$, where W is a regular language. For each such regular language W, recognizable by a finite-word automaton $\mathcal{A} = (\Sigma, Q, I, T, F)$, where $Q = \{q_1, q_2, \ldots, q_n\}$, we define a WS1S formula $\psi_W(y)$ over $V_1' = V_1$ and $V_2' = V_2 \cup \{At_{q_1}, \ldots, At_{q_n}\}$ that defines the words whose prefix up to position y is in W:

$$
\begin{aligned}
\psi_W(y) \;:=\; & \exists At_{q_1}, \ldots, At_{q_n}.\ \bigvee_{q \in I} (0 \in At_q) \\
\wedge\ & \forall x.\ x < y \rightarrow \Bigg(\bigvee_{(q_i, A, q_j) \in T} \Big(x \in At_{q_i} \wedge S(x) \in At_{q_j} \wedge \bigwedge_{P \in A \cap V_2} x \in P \wedge \bigwedge_{P \in V_2 \setminus A} x \notin P \\
& \qquad \wedge \bigwedge_{p \in A \cap V_1} x = p \wedge \bigwedge_{p \in V_1 \setminus A} x \neq p \Big) \Bigg) \\
\wedge\ & \forall x.\ x \leq y \rightarrow \Bigg(\bigwedge_{i \neq j} \neg \Big(x \in At_{q_i} \wedge x \in At_{q_j} \Big) \Bigg) \\
\wedge\ & \bigvee_{q_i \in F} y \in At_{q_i}
\end{aligned}
$$

The WS1S formula $\varphi_{\overrightarrow{W}} :\ = \forall x.\ \exists y.\ (x < y \wedge \psi_W(y))$ then defines the words in $\overrightarrow{W}$. Hence, $\mathcal{L}(\varphi)$ is WS1S-definable. □

8 Alternating Büchi Automata

Logics are often significantly more concise than automata. For example, in the translation from S1S to Büchi automata in the proof of Theorem 7.4, each negation increases the size of the Büchi automaton exponentially, resulting in a non-elementary number of states. The blow-up when translating LTL formulas is less dramatic, but still exponential. In this section, we show that the conciseness of the logic and the automata can be brought closer together when the automata are equipped with both nondeterministic and universal choices.

8.1 Alternating Automata

A *nondeterministic* choice requires that the suffix of an input word is accepted by *some* successor state. Dually, a *universal* choice requires that the suffix of an input word is accepted by *all* successor states. In an *alternating automaton*, we allow for both types of choices by defining, for each state and input letter, a positive Boolean formula over the successor states: disjunction expresses nondeterministic choice, conjunction universal choice.

Definition 8.1. The *positive Boolean formulas* over a set X, denoted $\mathbb{B}^+(X)$, are the formulas built from elements of X, conjunction $\wedge$, disjunction $\vee$, *true* and *false*.

A set $Y \subseteq X$ *satisfies* a formula $\varphi \in \mathbb{B}^+(X)$, denoted $Y \models \varphi$, iff the truth assignment that assigns *true* to the members of Y and *false* to the members of $X \setminus Y$ satisfies φ.

Definition 8.2. An *alternating automaton over infinite words* $\mathcal{A}$ is a tuple $\mathcal{A} = (\Sigma, Q, q_0, \delta, Acc)$, where:

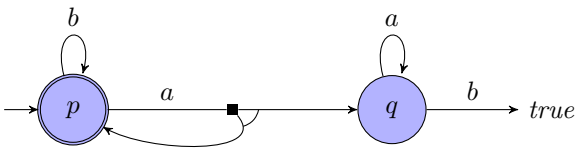
- Q is a finite set of states,
- $q_0 \in Q$ is the initial state,
- $\delta \colon Q \times \Sigma \to \mathbb{B}^+(Q)$ is the *transition function*, and
- $Acc \subseteq Q^\omega$ is an accepting condition.

For alternating automata, runs generalize from sequences to trees. A tree $\mathcal{T}$ over a set of *directions* D is a prefix-closed subset of D^*. The empty sequence ε is called the *root*. The children of a node $n \in \mathcal{T}$ are the nodes $\text{children}(n) = \{n \cdot d \in \mathcal{T} \mid d \in D\}$. A Σ-labeled tree is a pair $(\mathcal{T}, \ell)$, where $\ell \colon \mathcal{T} \to \Sigma$ is the labeling function.

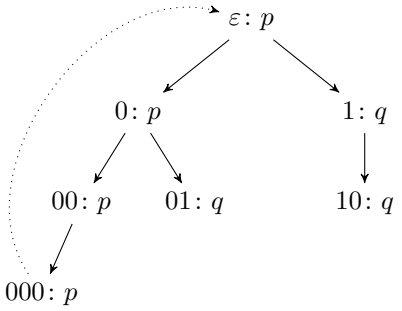
Definition 8.3. A *run* of an alternating automaton on a word $\alpha \in \Sigma^\omega$ is a Q-labeled *tree* $(\mathcal{T}, r)$ with the following properties:

- $r(\varepsilon) = q_0$ and
- for all $n \in \mathcal{T}$, if $r(n) = q$, then $\{r(n') \mid n' \in \text{children}(n)\}$ satisfies $\delta(q, \alpha(|n|))$.

Example 8.1. The following alternating Büchi automaton recognizes the language $L = (\{a, b\}^* b)^\omega$. The transition function is given as follows: $\delta(p, a) = p \wedge q$, $\delta(p, b) = p$, $\delta(q, a) = q$, and $\delta(q, b) = \mathit{true}$. We depict universal choice by connecting the edges with a small arc.



On the input word $\alpha = (aab)^\omega$, our automaton produces the following run. Note that, in general, an alternating automaton may also have more than one run on a particular word, or no run at all. We use a dotted line to indicate that the subtree repeats infinitely often.



To determine if a run of an alternating automaton is accepting, we apply the acceptance condition to all infinite branches of the run tree. A *branch* of a tree $\mathcal{T}$ is a maximal sequence of words $n_0 n_1 n_2 \ldots$ such that $n_0 = \varepsilon$ and n_{i+1} is a child of n_i for $i \geq 0$.

Definition 8.4. A run $(\mathcal{T}, r)$ is *accepting* iff, for every infinite branch $n_0 n_1 n_2 \ldots$,

$$r(n_0) r(n_1) r(n_2) \ldots \in Acc.$$

8.2 From LTL to Alternating Automata

It is usually much simpler to translate a logical formula into an alternating automaton than into a nondeterministic automaton. We illustrate this with the translation of LTL formulas into equivalent alternating Büchi automata. The states are simply the subformulas of the given formula and their negations (this set is called the *closure* of the formula). The transition function is derived from the *expansition laws* of the logic. For example, an Until formula $\psi_1 \mathcal{U} \psi_2$ holds if ψ_2 holds *or* if ψ_1 holds *and* the entire formula holds in the next step. The boolean formula produced by the transition function from the state $\psi_1 \mathcal{U} \psi_2$ is therefore a *disjunction* between the transition function for ψ_2 and a *conjunction* between the transition function for ψ_1 and the state $\psi_1 \mathcal{U} \psi_2$.

Construction 8.1. Let φ be an LTL formula. We construct the alternating Büchi automaton $\mathcal{A}_\varphi = (\Sigma, Q, \varphi, \delta, \textsc{büchi}(F))$ using:

- $Q = \text{closure}(\varphi) := \{\psi, \neg\psi \mid \psi \text{ is subformula of } \varphi\}$
- $\delta(p, a) = \begin{cases} \mathit{true} & \text{if } p \in a \\ \mathit{false} & \text{if } p \notin a \end{cases}$
- $\delta(\mathit{true}, a) = \mathit{true}$
- $\delta(\mathit{false}, a) = \mathit{false}$
- $\delta(\neg\psi, a) = \overline{\delta(\psi, a)}$
- $\delta(\psi_1 \wedge \psi_2, a) = \delta(\psi_1, a) \wedge \delta(\psi_2, a)$
- $\delta(\psi_1 \vee \psi_2, a) = \delta(\psi_1, a) \vee \delta(\psi_2, a)$
- $\delta(\bigcirc \psi, a) = \psi$
- $\delta(\psi_1 \mathcal{U} \psi_2, a) = \delta(\psi_2, a) \vee (\delta(\psi_1, a) \wedge \psi_1 \mathcal{U} \psi_2)$
- $F = \{\neg(\psi_1 \mathcal{U} \psi_2) \in \text{closure}(\varphi)\}$

where we define $\overline{\cdot}$ for $\psi, \psi_1, \psi_2 \in Q$ via

- $\overline{\psi} = \neg\psi$
- $\overline{\neg\psi} = \psi$
- $\overline{\psi_1 \wedge \psi_2} = \overline{\psi_1} \vee \overline{\psi_2}$
- $\overline{\psi_1 \vee \psi_2} = \overline{\psi_1} \wedge \overline{\psi_2}$
- $\overline{\mathit{true}} = \mathit{false}$
- $\overline{\mathit{false}} = \mathit{true}$

Theorem 8.1. *For every LTL formula φ, there is an alternating Büchi automaton $\mathcal{A}_\varphi$ with $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$.*

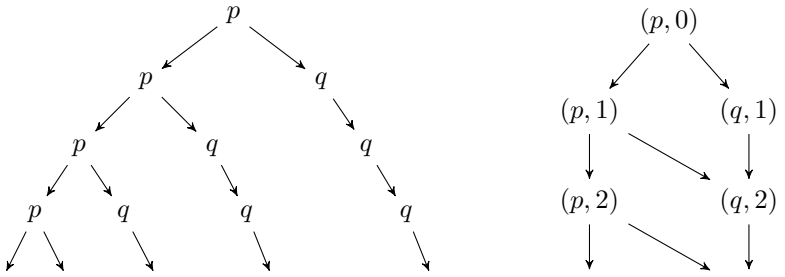
Proof. For a subformula ψ of φ let $\mathcal{A}_\varphi^\psi$ be the automaton $\mathcal{A}_\varphi$ from Construction 8.1 with initial state ψ. We prove, by structural induction on ψ, that $\mathcal{L}(\mathcal{A}_\varphi^\psi) = \mathcal{L}(\psi)$. □

8.3 Translating Alternating to Nondeterministic Automata

The translation from alternating to nondeterministic automata is based on a representation of runs as directed acyclic graphs (DAGs). The idea is similar to the DAG representation we used in the complementation construction for nondeterministic Büchi automata in Section 5. There the DAG was used to represent the set of all runs of the nondeterministic automaton. The complement automaton would then "guess" the DAG level-by-level.

Here, the DAG is used to represent the branches of a (single) run of the alternating automaton. The idea is illustrated in the following example.

Example 8.2. The following is a run tree of an automaton with two states p and q and its representation as a DAG.



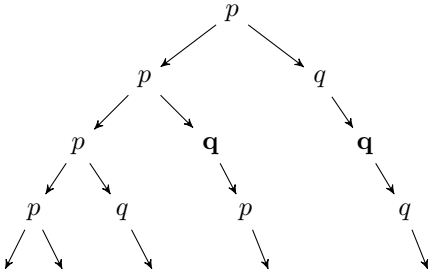
Definition 8.5. A *run DAG* of an alternating Büchi automaton $\mathcal{A}$ on an infinite word α is a DAG (V, E), with $V \subseteq Q \times \mathbb{N}$ and $(q_0, 0) \in V$, where

- $E \subseteq \bigcup_{i \in \mathbb{N}} (Q \times \{i\}) \times (Q \times \{i+1\})$
- $\forall (q, i) \in V \;.\; \exists Y \subseteq Q$ s.t.
  $Y \models \delta(q, \alpha(i))$, $Y \times \{i+1\} \subseteq V$ and $\{(q, i)\} \times (Y \times \{i+1\}) \subseteq E$.

A run DAG is *accepting* if every infinite path has infinitely many visits to $F \times \mathbb{N}$.

Our construction of the nondeterministic automaton will be based on run DAGs rather than trees. It is important to note, however, that not every run tree can be represented as a DAG. This is illustrated by the following example:

Example 8.3. The following run tree cannot be represented as a DAG, because there are two nodes on the third level that are both labeled with **q** that differ, however, in the labeling of their children.



We call two nodes $x_1, x_2 \in \mathcal{T}$ in a run tree $(\mathcal{T}, r)$ *similar* if $|x_1| = |x_2|$ and $r(x_1) = r(x_2)$. Run trees where the subtrees starting in similar nodes have the same labels are called *memoryless*. Memoryless run trees can be represented as DAGs.

Definition 8.6. A run tree $(\mathcal{T}, r)$ is *memoryless* if for all similar nodes x_1 and x_2 and for all $y \in D^*$ we have that $(x_1 \cdot y \in \mathcal{T}$ iff $x_2 \cdot y \in \mathcal{T})$ and $r(x_1 \cdot y) = r(x_2 \cdot y)$.

The following theorem shows that whenever there is an accepting run tree, there is also an accepting run tree that is memoryless. Hence, the existence of a memoryless run tree, or, equivalently, the existence of an accepting run DAG, is a necessary (and sufficient) criterion for the acceptance of some word.

Theorem 8.2. *If an alternating Büchi automaton $\mathcal{A}$ accepts a word α, then there exists a memoryless accepting run of $\mathcal{A}$ on α.*

Proof. Let $(\mathcal{T}, r)$ be an accepting run tree on α with directions D. We construct a memoryless run tree $(\mathcal{T}', r')$ by copying from $(\mathcal{T}, r)$. Inuitively, we pick, whenever there are multiple occurrences of the same state in $(\mathcal{T}, r)$, the occurrence where the last visit to the accepting states was the longest time ago. Formally, let $\gamma \colon \mathcal{T} \to \mathbb{N}$ be a function that measures the number of steps since the last visit to F:

- $\gamma(\varepsilon) = 0$
- $\gamma(n \cdot d) = \begin{cases} \gamma(n) + 1 & \text{if } r(n) \notin F \\ 0 & \text{otherwise} \end{cases}$

Based on γ, we define a mapping $\Delta \colon Q \times \mathbb{N} \to \mathcal{T}$ that assigns to each state and level a unique tree node:

$$\begin{aligned} \Delta(q, n) \quad = \quad & \text{the leftmost } y \in \mathcal{T} \text{ with } |y| = n \text{ s.t. } r(y) = q \\ & \text{and } \forall z \in \mathcal{T}.\ |z| = n \wedge r(z) = q \Rightarrow \gamma(z) \leq \gamma(y) \end{aligned}$$

We now construct $(\mathcal{T}', r')$ by copying from the nodes in $(\mathcal{T}, r)$ indicated by Δ:

- $\varepsilon \in \mathcal{T}'$ and $r'(\varepsilon) = r(\varepsilon)$
- for $n \in \mathcal{T}'$ and $d \in D$, we have that $n \cdot d \in \mathcal{T}'$ if and only if $\Delta(r'(n), |n|) \cdot d \in \mathcal{T}$ and $r'(n \cdot d) = r(\Delta(r'(n), |n|) \cdot d)$

It is easy to see that $(\mathcal{T}', r')$ is a run of $\mathcal{A}$ on α: The root is labeled by the initial state: $r'(\varepsilon) = r(\varepsilon) = q_0$. For $n \in \mathcal{T}'$, let $q_n = \Delta(r'(n), |n|)$. Then, the set $\{r(q_n \cdot d) \mid d \in D, q_n \cdot d \in \mathcal{T}\}$ satisfies $\delta(r(q_n), \alpha(|q_n|))$ and therefore $\{r'(n \cdot d) \mid d \in D, n \cdot d \in \mathcal{T}'\} \models \delta(r'(n), \alpha(|n|))$.

To prove that $(\mathcal{T}', r')$ is accepting we first show that, for every $n \in \mathcal{T}'$, it holds that $\gamma(n) \leq \gamma(\Delta(r'(n), |n|))$. This is shown by induction on the length of n:

- for $n = \varepsilon$ we have that $\gamma(n) = 0$
- for $n = n' \cdot d$ (where $d \in D$) we have:
  - if $r(n') \in F$, then $\gamma(n) = 0$
  - if $r(n') \notin F$, then

$$\begin{aligned} \gamma(\Delta(r'(n' \cdot d), |n' \cdot d|)) &\overset{\text{Def. } \Delta}{\geq} \gamma(\Delta(r'(n'), |n'|) \cdot d) \overset{\text{Def. } \gamma}{=} 1 + \gamma(\Delta(r'(n'), |n'|)) \\ &\overset{\text{IH}}{\geq} 1 + \gamma(n') \overset{\text{Def. } \gamma}{=} \gamma(n' \cdot d) \end{aligned}$$

Suppose now that $(\mathcal{T}', r')$ is not accepting. Then there is an infinite branch $n_0, n_1, n_2, \ldots$ in $\mathcal{T}'$ and $\exists k \in \mathbb{N}$ such that $\forall j \geq k.\ r'(n_j) \notin F$. Let $m_i = \Delta(r'(n_i), |n_i|)$ for $i \geq k$. We have,

$$\begin{array}{ccccc} \gamma(n_k) & < & \gamma(n_{k+1}) & < & \ldots \\ /\!\!\bigwedge & & /\!\!\bigwedge & & \\ \gamma(m_k) & < & \gamma(m_{k+1}) & < & \ldots \end{array}$$

So, for any $j \geq k$ it holds that $\gamma(m_k) \geq j - k$. Since $\mathcal{T}$ is finitely branching, there must be a branch with an infinite suffix of non-F labeled positions. This contradicts our assumption that $(\mathcal{T}, r)$ is accepting. □

Corollary 8.1. *A word α is accepted by an alternating Büchi automaton $\mathcal{A}$ if and only if $\mathcal{A}$ has an accepting run DAG on α.*

We are now ready to translate an alternating Büchi automaton into an equivalent nondeterministic Büchi automaton. The construction is due to Miyano and Hayashi (1984).

Construction 8.2. For an alternating Büchi automaton $\mathcal{A} = (\Sigma, Q, q_0, \delta, \text{BÜCHI}(F))$, we construct a nondeterministic Büchi automaton $\mathcal{A} = (\Sigma, Q', I', T', \text{BÜCHI}(F'))$ with $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$ as follows:

- $Q' = 2^Q \times 2^Q$
- $I' = \{(\{q_0\}, \varnothing)\}$
- $T' = \{((X, \varnothing), \sigma, (X', X' \setminus F)) \mid X' \models \bigwedge_{q \in X} \delta(q, \sigma)\} \cup$
  $\{((X, W), \sigma, (X', W' \setminus F)) \mid W \neq \varnothing, W' \subseteq X',$
  $X' \models \bigwedge_{q \in X} \delta(q, \sigma),\ W' \models \bigwedge_{q \in W} \delta(q, \sigma)\}$
- $F' = \{(X, \varnothing) \mid X \subseteq Q\}$

Theorem 8.3 (Miyano and Hayashi, 1984)**.** *For every alternating Büchi automaton $\mathcal{A}$, there exists a nondeterministic Büchi automaton $\mathcal{A}$ with $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.*

Proof. "$\mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}(\mathcal{A})$": Let $\alpha \in \mathcal{L}(\mathcal{A}')$ with an accepting run

$$r' = (X_0, W_0)(X_1, W_1)(X_2, W_2)\ldots$$

where $W_0 = \varnothing$ and $X_0 = \{s_0\}$. We construct the run DAG (V, E) for $\mathcal{A}$ on α:

- $V = \{(x, i) \mid i \in \mathbb{N},\ x \in X_i\}$
- $E = \{((x, i), (x', i+1)) \mid i \in \mathbb{N},\ x \in X_i \setminus W_i,\ x' \in X_i\} \cup$
  $\{((x, i), (x', i+1)) \mid i \in \mathbb{N},\ x \in W_i,\ x' \in X_{i+1} \cap (F \cup W_{i+1})\}$

First, we show that (V, E) is a run DAG: $(q_0, 0) \in V$ and for every $(x, i) \in V$: if $x \in X_i \setminus W_i$, $X_{i+1} \models \delta(x, \alpha(i))$; if $x \in W_i$, $X_{i+1} \cap (F \cup W_{i+1}) \models \delta(x, \alpha(i))$. The run DAG is accepting, because every path through the run DAG visits F infinitely often (otherwise $W_i = \varnothing$ only for finitely many i).

"$\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$": Let $\alpha \in \mathcal{L}(A)$ and (V, E) be an accepting run DAG of $\mathcal{A}$ on α. We construct a run

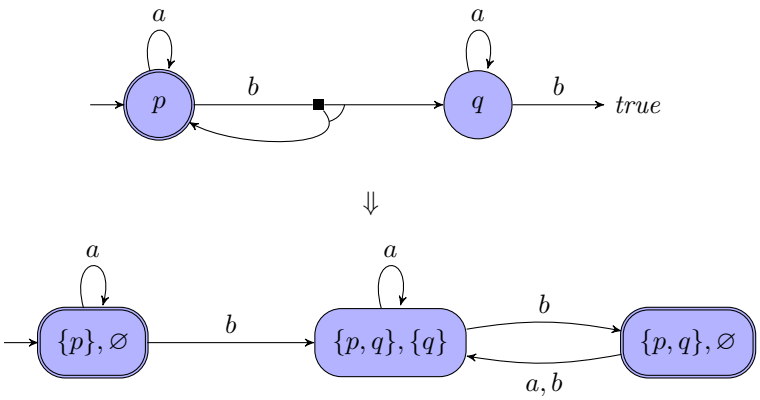
$$r' = (X_0, W_0)(X_1, W_1)(X_2, W_2)\ldots$$

on $\mathcal{A}'$ as follows:

- $X_0 = \{q_0\}$ and $W_0 = \varnothing$
- for $i > 0$, let $X_i = \{(x', i) \in V \mid ((x, i-1), (x', i)) \in E,\ (x, i-1) \in X_{i-1}\}$ and
  - if $W_i = \varnothing$ then $W_{i+1} = X_{i+1} \setminus F$,
  - otherwise, let $W_{i+1} = \{x' \in Q \setminus F \mid \exists (x, i) \in V, ((x, i), (x', i+1)) \in E, x \in W_i\}$.

Clearly, r' is a run: it starts with $(\{q_0\}, \varnothing)$ and obeys T', since for $x \in X_i \smallsetminus W_i$, we have that $X_{i+1} \models \delta(x, \alpha(i))$. Furthermore, for $x \in W_i$, $X_{i+1} \cap (F \cup W_{i+1})$ satisfies $\delta(x, \alpha(i))$. The run r' is accepting, because otherwise there is a path in (V, E) that rejects. □

Example 8.4. We translate the following alternating automaton into an equivalent nondeterministic automaton:



Corollary 8.2. *A language is ω-regular if and only if it is recognizable by an alternating Büchi automaton.*

9 Infinite Games

We now introduce *infinite two-player games on finite graphs*. Infinite games are useful to solve the synthesis problem, where we are interested in finding a strategy that guarantees that a given specification is satisfied (cf. Section 1.2). As we will see, games also play a fundamental role in automata theory, in particular for automata over infinite trees.

9.1 Basic Definitions

The game is played on a graph, called the *arena*. The vertices of the graph are called *positions* and are partitioned into the positions of Player 0 and the positions of Player 1. A play of the game starts in some initial position, there, as well as in all subsequent positions, the player who owns the position chooses the edge on which the play is continued. The winner is determined by a *winning* condition, which, like the acceptance condition of an automaton on infinite words is a subset of the infinite words over the positions. Player 0 wins if the play is an element of the winning condition.

Definition 9.1. A *game arena* is a tuple $\mathcal{A} = (V, V_0, V_1, E)$, where

- V_0 and $V_1 = V \setminus V_0$ are disjoint sets of positions, called the positions of Player 0 and Player 1,
- $E \subseteq V \times V$ is a set of edges such that every position $v \in V$ has at least one outgoing edge $(v, v') \in E$.

Definition 9.2. A *play* is an infinite sequence $\rho \in V^\omega$ such that

$$\forall n \in \mathbb{N}.\ (\rho(n), \rho(n+1)) \in E.$$

We say a play ρ *starts* in a position v iff $v = \rho(0)$. We denote the set of all possible plays on $\mathcal{A}$ with $\mathrm{Plays}(\mathcal{A})$ and the set of all possible plays starting in position v with $\mathrm{Plays}(\mathcal{A}, v)$.

Definition 9.3. A *game* $\mathcal{G} = (\mathcal{A}, \mathrm{Win})$ consists of an arena $\mathcal{A}$ and a *winning condition* $\mathrm{Win} \subseteq V^\omega$. We call a play ρ *winning for Player 0* iff $\rho \in \mathrm{Win}$ and *winning for Player 1* otherwise.

A *strategy* fixes the decisions of a player based on the prefix of the play seen so far. We call such a prefix the *history* of the play. A history that ends in a position of Player i is an element of V^*V_i. A strategy for Player i is a function $\sigma : V^*V_i \to V$ that selects for each such history a successor position.

Definition 9.4. A *strategy* for Player i is a function $\sigma : V^*V_i \to V$ such that $(v, v') \in E$ whenever $\sigma(wv) = v'$ for some $w \in V^*, v \in V_i$.

In the following, we will use σ and τ to denote strategies for some Player i and the opponent Player $(1 - i)$, respectively.

Definition 9.5. A play ρ is *consistent with* a strategy σ iff

$$\forall n \in \mathbb{N}\ .\ \text{if } \rho(n) \in V_i \text{ then } \rho(n+1) = \sigma(\rho[n]).$$

We denote the set of all plays that begin in some position v and are consistent with strategy σ with $\text{Plays}(\mathcal{A}, \sigma, v)$. Note that the strategies σ and τ of the two players together uniquely identify a specific play: $|\text{Plays}(\mathcal{A}, \sigma, v) \cap \text{Plays}(\mathcal{A}, \tau, v)| = 1$.

Our definition of a strategy is very general in the sense that the decisions are based on the entire history of the play. Intuitively, this means that the players have infinite memory. It often suffices to work with simpler strategies, such as *memoryless* strategies. Memoryless strategies are often also called *positional*.

Definition 9.6. A strategy σ for Player i is *memoryless* iff $\sigma(wv) = \sigma(v)$ for all $w \in V^*, v \in V_i$.

In a slight abuse of notation, memoryless strategies are often given directly as a function $\sigma : V_i \to V$ that maps the positions owned by Player i to their successor positions. Next, we characterize *winning* strategies:

Definition 9.7. A strategy σ for Player i is *winning* from a position v if all plays that start in v and that are consistent with σ are winning for Player i.

Note that this definition refers to a specific position v in which we start the play. The set of all positions where the player has a winning strategy is called the *winning region*.

Definition 9.8. The *winning region* $W_i(\mathcal{G})$ of Player i in a game $\mathcal{G}$ is defined as the set of positions $v \in V$ for which there exists a strategy for Player i that is winning from v.

Note that the strategies for different positions in the winning region may be different. If a strategy σ is winning from all positions of the winning region, we call σ a *uniform winning strategy*.

It is easy to see that no position can be in the winning regions of both players. Suppose that there exists a position v and strategies σ and τ that are winning from v for Player 0 and 1, respectively. Then the unique play that is consistent with σ and τ would need to be both in Win, because σ is winning, and not in Win, because τ is winning.

A more difficult question is whether all positions are in some winning region, i.e., whether the winning regions form a partition of V. This property is called the *determinacy* of a game:

Definition 9.9. A game $\mathcal{G}$ is *determined* if $V = W_0(\mathcal{G}) \cup W_1(\mathcal{G})$.

If the winning strategies are in fact memoryless, we say that the game is *memoryless* (also: *positionally*) determined.

Definition 9.10. A game is *memoryless determined* if for every position $v \in V$, there exists a memoryless strategy that is winning for some player from position v.

9.2 Reachability Games

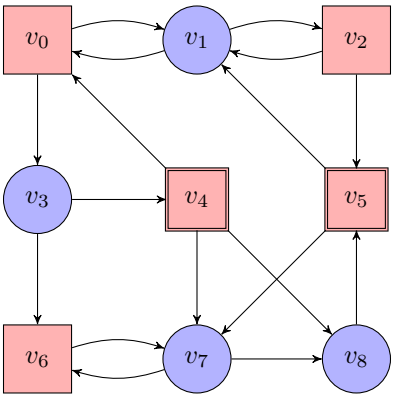
We will now analyze infinite games for various types of winning conditions. We start with the simple *reachability condition*. The reachability condition is given as a set R of positions called the *reachability set*. The reachability condition is satisfied if the play reaches some position in R. Formally, for an infinite word α over Σ, we use $\text{Occ}(\alpha) := \{\sigma \in \Sigma \mid \exists n \in \mathbb{N}.\ \alpha(n) = \sigma\}$ to denote the set of all letters occurring in α.

Definition 9.11. The *reachability condition* $\textsc{reach}(R)$ on a set of positions $R \subseteq V$ is the set

$$\textsc{reach}(R) = \{\rho \in V^\omega \mid \mathrm{Occ}(\rho) \cap R \neq \varnothing\}.$$

A game $\mathcal{G} = (\mathcal{A}, \mathrm{Win})$ with $\mathrm{Win} = \textsc{reach}(R)$ is called a *reachability game* with reachability set R.

Example 9.1. Consider the following reachability game $\mathcal{G}$ with $R = \{v_4, v_5\}$. We depict positions of Player 0 as circles and positions of Player 1 as rectangles. Positions in R are depicted with double lines.



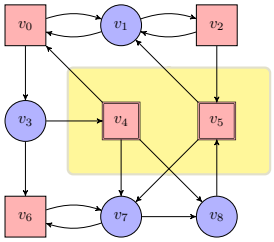
The winning region for Player 0 is $W_0(\mathcal{G}) = \{v_3, v_4, v_5, v_6, v_7, v_8\}$, as the following uniform winning strategy σ shows: $\sigma(v_1) = v_2,\ \sigma(v_3) = v_4,\ \sigma(v_7) = v_8,\ \sigma(v_8) = v_5$.

Reachability games can be solved with a simple fixed point construction called the *attractor construction*. The attractor construction computes the winning region for Player 0 by starting with the reachability set and then iteratively adding all positions owned by Player 0 that have an edge into the winning region, and all positions owned by Player 1 where all edges lead into the winning region. This process is repeated until no more positions can be added. In the following, we give a slightly more general definition of the attractor construction that can be applied also to Player 1. We do this in preparation for the constructions for other winning conditions, which will use the attractor construction as a subroutine.

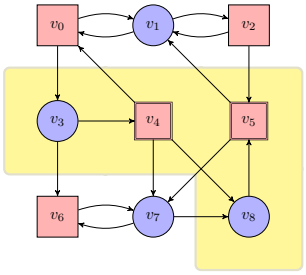
Construction 9.1. Let an arena $\mathcal{A} = (V, V_0, V_1, E)$ be given. The *attractor construction* on $\mathcal{A}$ is defined for each Player i, for all $n \in \mathbb{N}$ and $R \subseteq V$ as follows.

$$\begin{aligned}
\mathit{CPre}_i(R) &= \{v \in V_i \mid \exists v' \in V.\, (v, v') \in E \wedge v' \in R\} \\
&\quad \cup \{v \in V_{1-i} \mid \forall v' \in V.\, (v, v') \in E \to v' \in R\} \\
\\
\mathit{Attr}_i^0(R) &= R \\
\mathit{Attr}_i^{n+1}(R) &= \mathit{Attr}_i^n(R) \cup \mathit{CPre}_i(\mathit{Attr}_i^n(R)) \\
\\
\mathit{Attr}_i(R) &= \bigcup_{n \in \mathbb{N}} \mathit{Attr}_i^n(R)
\end{aligned}$$

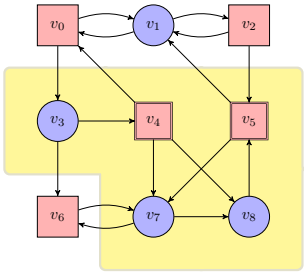
Example 9.2. Consider again the reachability game $\mathcal{G}$ from Example 9.1. Using the attractor construction, we solve the game as follows:



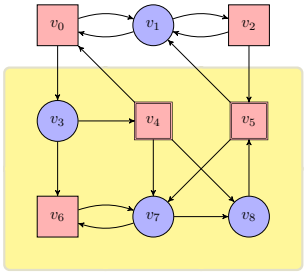
$Attr_0^0(\{v_4, v_5\}) = \{v_4, v_5\}$



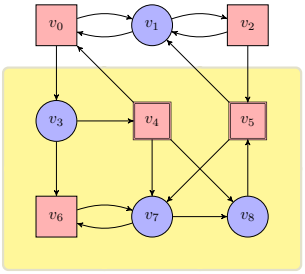
$Attr_0^1(\{v_4, v_5\}) = \{v_4, v_5\} \cup \{v_3, v_8\}$



$Attr_0^2(\{v_4, v_5\}) = \{v_3, v_4, v_5, v_8\} \cup \{v_3, v_7, v_8\}$



$Attr_0^3(\{v_4, v_5\}) = \{v_3, v_4, v_5, v_7, v_8\} \cup \{v_3, v_6, v_7, v_8\}$



$Attr_0^4(\{v_4, v_5\}) = Attr_0(\{v_4, v_5\}) = \{v_3, v_4, v_5, v_6, v_7, v_8\}$

In general, the attractor construction solves a game with winning condition $\textsc{reach}(R)$ as follows: $W_0(\mathcal{G}) = Attr_0(R)$, $W_1(\mathcal{G}) = V \setminus W_0(\mathcal{G})$. We can furthermore give a uniform memoryless winning strategy. These results are summarized in the following theorem.

Theorem 9.1. *Reachability games are memoryless determined. It holds that $W_0(\mathcal{G}) = Attr_0(R)$, $W_1(\mathcal{G}) = V \setminus W_0(\mathcal{G})$. Both players have a uniform winning strategy.*

Proof. We show for all positions $v \in V$ that

- if $v \in Attr_0(R)$, then $v \in W_0(\mathcal{G})$, with the following uniform memoryless strategy σ:

  We fix an arbitrary total ordering on V. For $v \in (Attr_0(R) \setminus R) \cap V_0$, let $n = \min\{n \in \mathbb{N} \mid v \in Attr_0^n(R)\}$. Then, let $\sigma(v)$ be the smallest $v' \in Attr_0^{n-1}(R)$ with $(v, v') \in E$. For every other position $v \in V_0 \setminus (Attr_0(R) \setminus R)$, let $\sigma(v)$ be the smallest $v' \in V$ with $(v, v') \in E$. We show, by induction on $n \in \mathbb{N}$, that any play that starts in $v \in Attr_0^n(R)$ and is consistent with σ reaches R within at most n steps.

- if $v \in V \setminus Attr_0(R)$, then $v \in W_1(\mathcal{G})$ with the following uniform memoryless strategy τ:

  We again fix an arbitrary total ordering on V. For $v \in V_1 \setminus Attr_0(R)$ let $\tau(v)$ be the smallest $v' \in V \setminus Attr_0(R)$ such that $(v, v') \in E$. Such a successor v' always exists, because otherwise $v \in Attr_0(R)$. For every other position $v \in V_1 \cap Attr_0(R)$, let $\tau(v)$ be the smallest $v' \in V$ with $(v, v') \in E$. Now let ρ be an arbitrary play that is consistent with τ. We show, by induction on n, that $\rho(n) \notin Attr_0(R)$ and, hence, $\rho(n) \notin R$, for all $n \in \mathbb{N}$. □

9.3 Büchi Games

In a Büchi game, the goal of Player 0 is to visit some accepting position infinitely often. The attractor construction allows us to check whether there is a strategy which enforces at least one visit to an accepting position. Winning a Büchi game is more difficult. Reaching an accepting state at least once is indeed a necessary precondition, but we also have to ensure that from this position we can enforce a second visit to some accepting state, then a third, and so forth. The *recurrence construction* computes the largest subset of the accepting states from which Player 0 can enforce infinitely many subsequent visits to the subset.

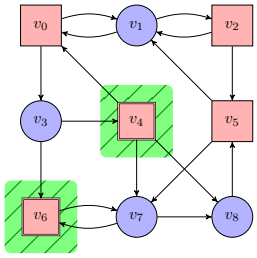
Construction 9.2. Let an arena $\mathcal{A} = (V_0, V_1, E)$ with $V = V_0 \cup V_1$ be given. The *recurrence construction* on $\mathcal{A}$ is defined for all $n \in \mathbb{N}$ and $F \subseteq V$ as:

$$\begin{aligned} \mathit{Recur}^0(F) &= F \\ W_1^n(F) &= V \setminus \mathit{Attr}_0(\mathit{Recur}^n(F)) \\ \mathit{Recur}^{n+1}(F) &= \mathit{Recur}^n(F) \setminus \mathit{CPre}_1(W_1^n(F)) \\ \mathit{Recur}(F) &= \bigcap_{n \in \mathbb{N}} \mathit{Recur}^n(F) \end{aligned}$$

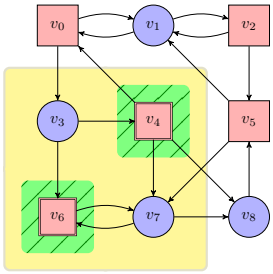
The set $\mathit{Recur}^n(F)$ contains the subset of F from which Player 0 can enforce at least n *further* (i.e., a total of at least $n+1$) visits to F. The set $W_1^n(F)$ contains those positions in V from which Player 1 can enforce that there are *at most* n visits to F. The set $\mathit{Recur}(F)$ contains the subset of F from which Player 0 can enforce infinitely many visits to F. The recurrence construction solves a game with winning condition $\textsc{büchi}(F)$ as follows:

$$W_0(\mathcal{G}) = \mathit{Attr}_0(\mathit{Recur}(F)),\ W_1(\mathcal{G}) = V \setminus W_0(\mathcal{G})$$

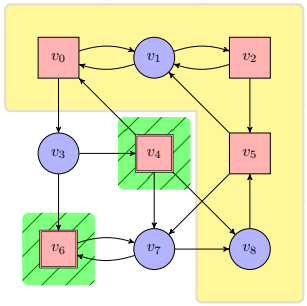
Example 9.3. We solve the game $\mathcal{G} = (\mathcal{A}, \textsc{büchi}(v_4, v_6))$ with arena $\mathcal{A}$ from Example 9.1:



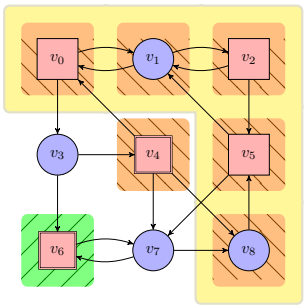
$\mathit{Recur}^0(\{v_4, v_6\}) = \{v_4, v_6\}$



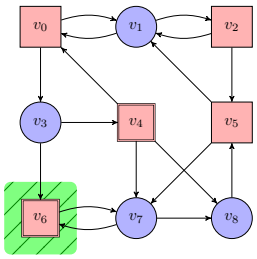
$\mathit{Attr}_0(\{v_4, v_6\}) = \{v_3, v_4, v_6, v_7\}$



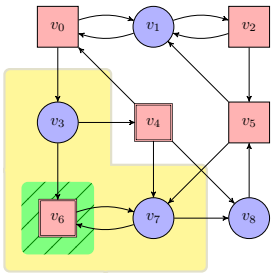
$W_1^0(\{v_4, v_6\}) = \{v_0, v_1, v_2, v_5, v_8\}$



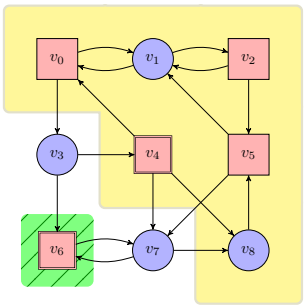
$\mathit{CPre}_1(\{v_0, v_1, v_2, v_5, v_8\}) = \{v_0, v_1, v_2, v_4, v_5, v_8\}$



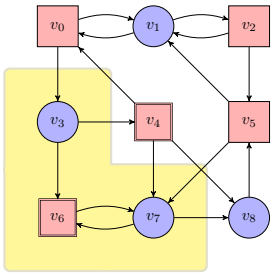
$\mathit{Recur}^1(\{v_4, v_6\}) = \{v_6\}$



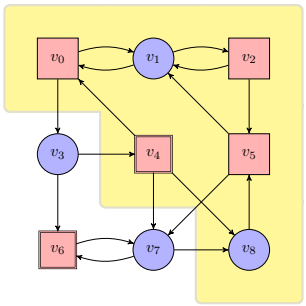
$\mathit{Attr}_0(\{v_6\}) = \{v_3, v_6, v_7\}$



$W_1^1(\{v_4, v_6\}) = \{v_0, v_1, v_2, v_4, v_5, v_8\}$



$W_0(\mathcal{G}) = \{v_3, v_6, v_7\}$



$W_1(\mathcal{G}) = \{v_0, v_1, v_2, v_4, v_5, v_8\}$

Theorem 9.2. *Büchi games are memoryless determined. It holds that* $W_0(\mathcal{G}) = Attr_0(Recur(F))$, $W_1(\mathcal{G}) = V \setminus W_0(\mathcal{G})$. *Both players have a uniform winning strategy.*

Proof. We show for all positions $v \in V$ that

- if $v \in Attr_0(Recur(F))$, then $v \in W_0(\mathcal{G})$, with the following uniform memoryless strategy σ:

  We fix some arbitrary total ordering on V. For $v \in (Attr_0(Recur(F)) \setminus Recur(F)) \cap V_0$, we follow the attractor strategy from the proof of Theorem 9.1. For $v \in Recur(F) \cap V_0$, we choose the smallest $v' \in V$ with $(v, v') \in E$ and $v' \in Attr_0(Recur(F))$. Such a successor must exist, because otherwise $v \in CPre_1(W_1^n(F))$ for some $n \in \mathbb{N}$, and hence $v \notin Recur(F)$. Every play that is consistent with σ visits $Recur(F) \subseteq F$ infinitely often. Hence, σ is winning for Player 0.

- if $v \in V \setminus Attr_0(Recur(F))$, then $v \in W_1(\mathcal{G})$ with the following uniform memoryless strategy τ:

  We again fix an arbitrary total ordering on V. We define the memoryless strategies τ such that, for each $n \in \mathbb{N}$, if a play starts in $v \in W_1^n = V \setminus Attr_0(Recur^n(F))$ and is consistent with τ, there are at most n visits to F.

  - For $n = 0$ let $\tau(v)$ be the smallest $v' \in V$ such that $(v, v') \in E$ and $v' \in V \setminus Attr_0(F)$.
  - For $n > 0$ let $\tau(v)$ be the smallest $v' \in W_1^{n-1}(F)$ with $(v, v') \in E$ if $v \in CPre_1(W_1^{n-1}(F))$ and as the smallest v' in $W_1^n(F)$ with $(v, v') \in E$ otherwise. Such a v' always exists as otherwise $v \in Attr_0(Recur^n(F))$. □

9.4 Parity Games

Perhaps the most intriguing type of infinite games are *parity games*. Parity games play a key role in verification (in particular for μ-calculus model checking) and synthesis, and finding fast algorithms for parity games is an active research topic. Part of the allure may be that the complexity status of parity games is still open. Solving parity games is in NP, but it is unknown if it can be done in polynomial time. The known algorithms take exponential time.

Definition 9.12. The *parity condition* $\textsc{parity}(c)$ for a coloring function $c\colon Q \to \mathbb{N}$ is the set

$$\textsc{parity}(c) = \{\alpha \in V^\omega \mid \max\{c(q) \mid q \in \mathrm{Inf}(\alpha)\} \text{ is even}\}.$$

We first prove that parity games are memoryless determined, and then derive an algorithm for solving parity games. In the following theorem, we emphasize that determinacy holds also for (countably) infinite game arenas. This will be helpful when we use the determinacy to complement tree automata, because the acceptance game of a tree automaton refers to the infinite input tree and is therefore infinite.

Theorem 9.3. *Parity games are memoryless determined with uniform winning strateges for game arenas with a countable set of positions and a finite number of colors.*

Proof. Let $k = \max\{c(v) \mid v \in V\}$ be the highest color in the given parity game. We prove that parity games are memoryless determined by induction on k.

Case $k = 0$: If the highest color is 0, then all plays are winning. $W_0(\mathcal{G}) = V, W_1(\mathcal{G}) = \varnothing$. For the memoryless winning strategy σ, we fix an arbitrary total order on V and choose $\sigma(v) = \min\{v' \in V \mid (v, v') \in E\}$.

Case $k > 0$: If k is even, consider Player i, otherwise Player $(1-i)$. Let W_{1-i} be the set of positions where Player $(1-i)$ has a memoryless winning strategy. We show that Player i has a memoryless winning strategy σ from $V \setminus W_{1-\sigma}$. Consider the subgame $\mathcal{G}'$:

- $V'_0 = V_0 \setminus W_{1-i}$, $V'_1 = V_1 \setminus W_{1-i}$, $V' = V'_0 \cup V'_1$
- $E' = E \cap (V' \times V')$
- $c'(v) = c(v)$ for all $v \in V'$

Note that $\mathcal{G}'$ is still a game:

- for $v \in V'_i$, there is a $v' \in V \setminus W_{1-i}$ with $(v, v') \in E'$, otherwise v would be in W_{1-i}
- for $v \in V'_{1-i}$, we have that for all $v' \in V$ with $(v, v') \in E$, $v' \in V \setminus W_{1-i}$, hence there is a $v' \in V'$ with $(v, v') \in E$

Let $c'^{-1}(k) = \{v \in V' \mid c'(v) = k\}$ and let $Y = Attr'_i(c'^{-1}(k))$. (We denote with $Attr'$ the attractor on the subgame $\mathcal{G}'$.) Let σ_A be the corresponding attractor strategy on $\mathcal{G}'$ into $c'^{-1}(k)$, as defined in the proof of Theorem 9.1.

Now consider the subgame $\mathcal{G}''$:

- $V''_0 = V'_0 \setminus Y$, $V''_1 = V'_1 \setminus Y$, $V'' = V''_0 \cup V''_1$
- $E'' = E' \cap (V'' \times V'')$
- $c'' : V'' \to \{0, \ldots, k-1\}; c''(v) = c'(v)$ for all $v \in V''$

Note that $\mathcal{G}''$ is still a game, and that the maximal color in $\mathcal{G}''$ is at most $k-1$. We therefore know, by induction hypothesis, that $\mathcal{G}''$ is memoryless determined. It is also clear that W''_{1-i}, the set of positions in game $\mathcal{G}''$ where Player $(1-i)$ has a memoryless winning strategy, is empty, because W''_{1-i} is a subset of W_{1-i}: assume Player $(1-i)$ had a memoryless winning strategy from some position in V''. Then this strategy would win in $\mathcal{G}$, too, since Player i has no opportunity to leave $\mathcal{G}''$ other than to W_{1-i}. Hence, there is a uniform winning memoryless winning strategy σ_{IH} for player i from all positions in V''. We define the following uniform strategy σ for Player i in game $\mathcal{G}$:

$$\sigma(v) = \begin{cases} \sigma_{IH}(v) & \text{if } v \in V'' \\ \sigma_A(v) & \text{if } v \in Y \setminus c'^{-1}(k) \\ \text{min. successor in } V \setminus W_{1-i} & \text{if } v \in Y \cap c'^{-1}(k) \\ \text{min. successor in } V & \text{otherwise.} \end{cases}$$

The strategy σ is winning for Player 0 on $V \setminus W_{1-\sigma}$. Consider a play that is consistent with σ:

Case 1: Y is visited infinitely often. Thus, Player i wins, because the color k, which is the highest color, is visited.

Case 2: Eventually only positions in V'' are visited. Hence, since Player i follows σ_{IH}, Player i wins. □

The proof is non-constructive in the sense that we begin the argument by considering (rather than computing) the set W_{1-i} of positions where the opponent, Player $(1-i)$, has a memoryless winning strategy. McNaughton's algorithm, one of the classic algorithms for parity games over finite arenas, computes this set iteratively, with repeated recursive calls:

Construction 9.3. Let a finite parity game $\mathcal{G} = (\mathcal{A}, \text{PARITY}(c))$ be given. We compute the winning regions $W_0(\mathcal{G})$ and $W_1(\mathcal{G})$ as follows:

Function *McNaughton*$(\mathcal{G})$ =

1. $k :=$ highest color in $\mathcal{G}$
2. **if** $k = 0$ or $V = \varnothing$
   then return $(V, \varnothing)$
3. $i := k \bmod 2$
4. $W_{1-i} := \varnothing$
5. **repeat**
   (a) $\mathcal{G}' := \mathcal{G} \setminus \mathit{Attr}_i(c^{-1}(k))$
   (b) $(W'_0, W'_1) := \mathit{McNaughton}(\mathcal{G}')$
   (c) **if** $(W'_{1-i} = \varnothing)$ **then**
      i. $W_i := V \setminus W_{1-i}$
      ii. **return** (W_0, W_1)
   (d) $W_{1-i} := W_{1-i} \cup \mathit{Attr}_{1-i}(W'_{1-i})$
   (e) $\mathcal{G} := \mathcal{G} \setminus \mathit{Attr}_{1-i}(W'_{1-i})$

10 Rabin's Theorem

Infinite games allow us to reason very elegantly about infinite trees. A famous example of an argument that became significantly simpler with the introduction of game-theoretic ideas is the proof of Rabin's theorem. Rabin's theorem states that the satisfiability of monadic second-order logic with two successors (S2S) is decidable. Like in Section 7, where we showed that S1S formulas can be translated to automata, we will show that S2S formulas can be translated to automata, this time, in order to accommodate more than one successor function, to automata over infinite trees. The most difficult part of the proof of Rabin's theorem is to show that tree automata are closed under complement. While Rabin's original proof was purely combinatorial (and very difficult to understand), the game-theoretic argument is simply based on the determinacy of the acceptance game of the tree automaton: the acceptance of a tree by a tree automaton can be characterized by the existence of a winning strategy for Player 0, the *non*-acceptance, by the absence of such a strategy, or, by determinacy, by the existence of a winning strategy for Player 1. We can therefore complement the language of a given tree automaton by constructing a new automaton that verifies the existence of a winning strategy for Player 1.

We begin this section with a discussion of tree automata. The logic S2S and the translation to tree automata will be introduced later in the section.

10.1 Tree Automata

We consider tree automata over infinite *binary* trees. We use the notation for trees introduced in Section 8.1. The (full) binary tree is the language $\mathcal{T} = \{0,1\}^*$. For an alphabet Σ, $\mathcal{T}_\Sigma = \{(\mathcal{T}, t) \mid t : \mathcal{T} \to \Sigma\}$ is the set of all binary Σ-labeled trees.

Definition 10.1. An *automaton over infinite binary trees* $\mathcal{A}$ is a tuple (Σ, Q, q_0, T, Acc), where

- Σ is a finite alphabet,
- Q is a finite set of states,
- $q_0 \in Q$ is an initial state,
- $T \subseteq Q \times \Sigma \times Q \times Q$,
- $Acc \subseteq Q^\omega$ is the acceptance condition.

In the following, we will refer to automata over infinite binary trees simply as *tree automata*. Note that a transition of a tree automaton has two successor states, rather than a successor state as for word automata. The two states correspond to the two directions 0 and 1 of the input tree, the automaton may transition into different states for the different directions.

Definition 10.2. A *run* of a tree automaton $\mathcal{A}$ on an infinite Σ-labeled binary tree $(\mathcal{T}, t)$ is a Q-labeled binary tree $(\mathcal{T}, r)$ such that the following hold:

- $r(\varepsilon) = q_0$
- $(r(n), t(n), r(n0), r(n1)) \in T$ for all $n \in \{0,1\}^*$

Note that $n0$ and $n1$ are the children of node n in direction 0 and 1, respectively. The *accepting* runs are defined as for alternating automata in Section 8.1: we apply the acceptance condition to the branches of the run tree. (Note that a run of an alternating automaton may have finite and infinite branches. The acceptance condition is only checked on infinite branches. Here, all branches are infinite.)

Definition 10.3. A run $(\mathcal{T}, r)$ is *accepting* iff, for every infinite branch $n_0 n_1 n_2 \ldots$,

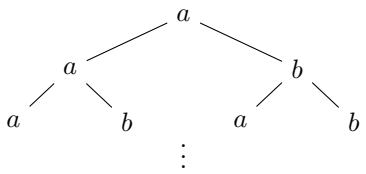
$$r(n_0)r(n_1)r(n_2)\ldots \in Acc.$$

The language of the tree automaton $\mathcal{A}$ consists of the set of accepted Σ-labeled trees.

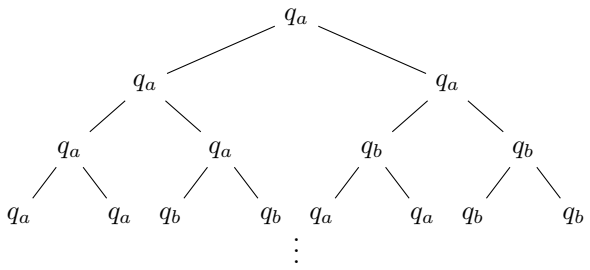
Example 10.1. The following Büchi tree automaton $\mathcal{A} = (\Sigma, Q, q_0, T, \text{BÜCHI}(F))$ accepts all $\{a, b\}$-labeled trees with infinitely many b's on each branch.

- $\Sigma = \{a, b\}$
- $Q = \{q_a, q_b\}; q_0 = q_a$
- $T = \{(q_a, a, q_a, q_a), (q_b, a, q_a, q_a), (q_a, b, q_b, q_b), (q_b, b, q_b, q_b)\}$
- $F = \{q_b\}$

Consider the input tree $(\mathcal{T}, t)$:



The following tree is a run of $\mathcal{A}$ on $(\mathcal{T}, t)$:

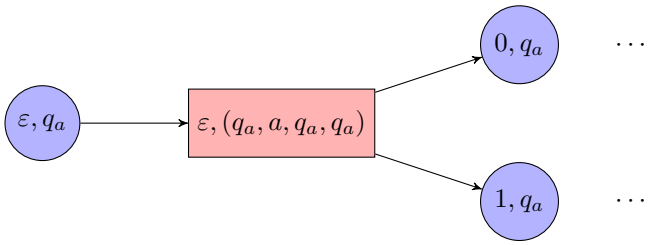


The acceptance mechanism of a tree automaton is also characterized via its *acceptance game*.

Definition 10.4. Let $\mathcal{A} = (\Sigma, Q, q_0, T, Acc)$ be a tree automaton and let $(\mathcal{T}, t)$ be a Σ-labeled binary tree. Then the *acceptance game* of $\mathcal{A}$ on $(\mathcal{T}, t)$ is the game $\mathcal{G}_{\mathcal{A},t} = (\mathcal{A}', \text{Win}')$ with the infinite game arena $\mathcal{A}' = (V', V_0', V_1', E')$ and the winning condition Win' defined as follows:

- $V_0' = \{(w, q) \mid w \in \{0,1\}^*,\ q \in Q\}$
- $V_1' = \{(w, \tau) \mid w \in \{0,1\}^*, \tau \in T\}$
- $E' = \{((w, q), (w, \tau)) \mid \tau = (q, t(w), q^0, q^1),\ \tau \in T\} \cup$
  $\{((w, \tau), (w', q')) \mid \tau = (q, \sigma, q^0, q^1)$ and
  $w' = w0$ and $q' = q^0$ or $w' = w1$ and $q' = q^1\}$
- $\text{Win}' = \{(w[0,0], q(0))(w[0,0], \tau(0))(w[0,1], q(1))(w[0,1], \tau(1))\ldots \mid$
  $q(0)q(1)\ldots \in Acc,\ w(0)w(1)\ldots \in \{0,1\}^\omega,\ \tau(0)\tau(1)\ldots \in T^\omega\}$

Example 10.2. The following is a part of the acceptance game of automaton $\mathcal{A}$ from Example 10.1.



Theorem 10.1. *A tree automaton $\mathcal{A} = (\Sigma, Q, q_0, T, Acc)$ accepts an input tree $(\mathcal{T}, t)$ if and only if Player 0 wins the acceptance game $\mathcal{G}_{\mathcal{A},t} = (\mathcal{A}', \mathrm{Win}')$ from position (ε, q_0).*

Proof. Given an accepting run $(\mathcal{T}, r)$ of $\mathcal{A}$, we construct a memoryless winning strategy $\sigma : V_0' \to V'$ for Player 0:

$$\sigma(w, q) = (w, (r(w), t(w), r(w0), r(w1))).$$

Conversely, given a winning strategy $\sigma : V'^* V_0' \to V'$, we construct an accepting run $(\mathcal{T}, r)$ where $r(\varepsilon) = q_0$ and for all $w \in \{0,1\}^*$

$$r(w0) = q \text{ for } \sigma(\Delta(w)) = (w, (\_, \_, q, \_)) \quad \text{and} \quad r(w1) = q \text{ for } \sigma(\Delta(w)) = (w, (\_, \_, \_, q)),$$

where $\Delta(\varepsilon) = (\varepsilon, q_0)$ and $\Delta(wd) = \Delta(w) \cdot \sigma(\Delta(w)) \cdot (wd, r(wd))$ for $d \in \{0,1\}$. □

The acceptance game can also be used to test if the language of a given tree automaton is non-empty. For this purpose, we first translate the given automaton into an automaton with singleton alphabet.

Construction 10.1. For a given tree automaton $\mathcal{A}$ over Σ-labeled trees we consider the following tree automaton $\mathcal{A}'$ over $\{1\}$-labeled trees, such that $\mathcal{L}(\mathcal{A}) = \varnothing$ iff $\mathcal{L}(\mathcal{A}') = \varnothing$:

- $Q' = Q$
- $q_0' = q_0$
- $T' = \{(q, 1, q', q'') \mid (q, \sigma, q', q'') \in T, \sigma \in \Sigma\}$
- $Acc' = Acc$

Because the subtrees of a $\{1\}$-labeled binary tree are the same from all nodes, we can simplify its acceptance game such that only finitely many positions are needed. We call this game the *emptiness game*.

Definition 10.5. Let $\mathcal{A} = (\Sigma, Q, q_0, T, Acc)$ be a tree automaton. The *emptiness game* of $\mathcal{A}$ is the game $\mathcal{G}_{\mathcal{A}} = (\mathcal{A}', \mathrm{Win}')$ with the finite arena $\mathcal{A}' = (Q \cup T, Q, T, E)$, where

$$E = \{(q, \tau) \mid \tau = (q, \sigma, q^0, q^1), \tau \in T\} \cup \{(\tau, q') \mid \tau = (\_, \sigma, q^0, q^1) \text{ and } (q' = q^0 \text{ or } q' = q^1)\}$$

and $\mathrm{Win}' = \{q(0)\tau(0)q(1)\tau(1)\ldots \mid q(0)q(1)\ldots \in Acc,\ \tau(0)\tau(1)\ldots \in T^\omega\}$.

Theorem 10.2. *The language of a tree automaton $\mathcal{A}$ is non-empty iff Player 0 wins the emptiness game $\mathcal{G}_\mathcal{A}$ from position q_0.*

Proof. The emptiness game corresponds to the acceptance game of the automaton from Construction 10.1 on the $\{1\}$-labeled binary tree. □

10.2 Complementation of Parity Tree Automata

We now prove that parity tree automata are closed under complementation. As discussed at the beginning of the section, our proof makes heavy use of the determinacy of parity games (with infinite game arenas) established in Theorem 9.3.

Theorem 10.3. *For every parity tree automaton $\mathcal{A}$ over Σ there is a parity tree automaton $\mathcal{A}'$ with $\mathcal{L}(\mathcal{A}') = \mathcal{T}_\Sigma \setminus \mathcal{L}(\mathcal{A}')$.*

Proof. Let $\mathcal{A} = (\Sigma, Q, q_0, T, \text{PARITY}(c))$. By Theorem 10.1, a tree $(\mathcal{T}, t)$ is accepted by $\mathcal{A}$ iff Player 0 has a winning strategy from position (ε, q_0) of the acceptance game $\mathcal{G}_{\mathcal{A},t}$. Since $\mathcal{A}$ is a parity tree automaton, $\mathcal{G}_{\mathcal{A},t}$ is a parity game and therefore, by Theorem 9.3, memoryless determined. Hence, $\mathcal{A}$ does *not* accept some tree t iff Player 1 has a winning memoryless strategy σ in $\mathcal{G}_{\mathcal{A},t}$ from (ε, q_0). The strategy

$$\sigma\colon \{0,1\}^* \times T \to \{0,1\}^* \times Q$$

can be represented as a function

$$\sigma'\colon \{0,1\}^* \times T \to \{0,1\}$$

where $\sigma(w, (q, \sigma, q^0, q^1)) = (wi, q^i)$ iff $\sigma'(w, (q, \sigma, q^0, q^1)) = i$. Yet another representation of the same strategy is

$$\sigma''\colon \{0,1\}^* \to (T \to \{0,1\}),$$

which can be understood as a labeling of $\mathcal{T}$ with finite "local strategies". Hence, $\mathcal{A}$ does *not* accept $(\mathcal{T}, t)$ iff

(1) there is a $(T \to \{0,1\})$-labeled tree $(\mathcal{T}, v)$ such that
(2) for all $i_0 i_1 i_2 \ldots \in \{0,1\}^\omega$
(3) for all $\tau_0 \tau_1 \ldots \in T^\omega$
(4) if

- for all j: $\tau_j = (q, a, q^0, q^1)$ implies $a = t(i_0 i_1 \ldots i_j)$ and
- $i_0 i_1 \ldots = v(\varepsilon)(\tau_0)v(i_0)(\tau_1)\ldots$

then
the generated state sequence $q(0)q(1)\ldots$ with $q(0) = q_0$, $(q(j), a, q^0, q^1) = \tau_j$, and $q(j+1) = q^{v(i_0,\ldots,i_{j-1})(\tau_j)}$ for all j violates $\text{PARITY}(c)$.

We now encode this condition as a tree automaton. Condition **(4)** is a property of words over alphabet

$$\Sigma_4 = \underbrace{(M \to \{0,1\})}_{v} \times \underbrace{\Sigma}_{t} \times \underbrace{T}_{\tau} \times \underbrace{\{0,1\}}_{i}$$

and can be checked by a parity word automaton $\mathcal{A}_4 = (\Sigma', Q_4, \{q_4\}, T_4, \text{PARITY}(c_4))$:

- $Q_4 = Q \cup \{\bot\}$
- $q_4 = q_0$
- $T_4 = \{(q, (f, a, (q, a, q^0, q^1), i), q^i) \mid$
$$q \in Q,\ f : T \to \{0,1\},\ (q, a, q^0, q^1) \in T,\ i = f(q, a, q^0, q^1)\} \cup$$
$$\{(q, (f, a, (q, a', q^0, q^1), i), \bot) \mid a \neq a' \text{ or } i \neq f(q, a', q^0, q^1)\} \cup \{(\bot, a, \bot) \mid a \in \Sigma'\}$$
- $c_4(q) = c(q) + 1$ for $q \in Q$ and $c_4(\bot) = 0$

Condition **(3)** is a property of words over $\Sigma_3 = (T \to \{0,1\}) \times \Sigma \times \{0,1\}$ which results from **(4)** by universal quantification (= complement → projection → complement). Hence, there is a deterministic parity word automaton $\mathcal{A}_3$ that checks **(3)**.

Condition **(2)** defines a property of Σ_2-labeled trees where $\Sigma_2 = (T \to \{0,1\}) \times \Sigma$. It can be checked by a tree automaton $\mathcal{A}_2 = (Q_2, q_2, T_2, \text{PARITY}(c_2))$, that simulates $\mathcal{A}_3$ along each path:

- $Q_2 = Q_3$
- $q_2 = q_3$
- $T_2 = \{(q, (f, a), q^0, q^1) \mid (q, (f, a, 0), q^0) \in T_3, (q, (f, a, 1), q^1) \in T_3\}$
- $c_2 = c_3$

Condition **(1)** is a property on Σ-labeled trees: We use nondeterminism to guess the $T \to \{0,1\}$ label: $\mathcal{A}_1 = (Q_1, q_1, T_1, \text{PARITY}(c_1))$, where

- $Q_1 = Q_2$
- $q_1 = q_2$
- $T_1 = \{(q, a, q^0, q^1) \mid \exists f : T \to \{0,1\}.\ (q, (f, a), q^0, q^1) \in T_2\}$
- $c_1 = c_2$ □

10.3 Monadic Second-Order Logic of Two Successors (S2S)

In Section 7.3 we introduced the monadic second-order logic of one successor to describe sets of infinite *words* through quantification over positions and sets of positions. In this section, we extend this concept to infinite *trees*, where we now quantify over nodes (and sets of nodes) in the tree instead of positions in the word. We begin with binary trees, leading to the *Monadic Second-Order Logic of Two Successors*. In contrast to S1S, we now have two successor operations, one to address the left child in of a node in the tree and one to address the right child.

The *terms* of S2S are defined by the following grammar:

$$t ::= \varepsilon \mid x \mid t0 \mid t1$$

The *formulas* of S2S are defined by the following grammar:

$$\varphi ::= t \in X \mid t_1 = t_2 \mid \neg\varphi \mid \varphi_0 \vee \varphi_1 \mid \exists x.\, \varphi \mid \exists X.\, \varphi$$

where $x \in V_1$ and $X \in V_2$ are first-order and second-order variables, respectively.

The semantics of S2S is again defined relative to a valuation of the variables. The *first-order valuation* $\sigma_1 \colon V_1 \to \{0,1\}^*$ now assigns to each first-order variable a node in the tree. The *second-order valuation* $\sigma_2 \colon V_2 \to 2^{\{0,1\}^*}$ assigns to each second-order variable a set of nodes. The value of a term is then defined as follows:

- $[\varepsilon]_{\sigma_1} = \varepsilon$
- $[x]_{\sigma_1} = \sigma_1(x)$
- $[t0]_{\sigma_1} = [t]_{\sigma_1} 0$
- $[t1]_{\sigma_1} = [t]_{\sigma_1} 1$

We again distinguish free and bound occurrences of a variable, and identify the subsets $V_1' \subseteq V_1$ and $V_2' \subseteq V_2$ of free first-order and free second-order variables, respectively. An S2S *formula* φ defines the following language of infinite $2^{V_1' \cup V_2'}$-labeled binary trees:

$$\mathcal{L}(\varphi) = \{(\mathcal{T}, t_{\sigma_1,\sigma_2}) \mid \sigma_1, \sigma_2 \models \varphi\},$$

where, for all $n \in \{0,1\}^*$, $x \in t_{\sigma_1,\sigma_2}(n)$ iff $n = \sigma_1(x)$, and $X \in t_{\sigma_1,\sigma_2}(n)$ iff $n \in \sigma_2(X)$, and $\models$ is the smallest relation that satisfies the following:

- $\sigma_1, \sigma_2 \models t \in X$ iff $[t]_{\sigma_1} \in \sigma_2(X)$
- $\sigma_1, \sigma_2 \models t_1 = t_2$ iff $[t_1]_{\sigma_1} = [t_2]_{\sigma_1}$
- $\sigma_1, \sigma_2 \models \neg\varphi$ iff $\sigma_1, \sigma_2 \not\models \varphi$
- $\sigma_1, \sigma_2 \models \varphi_0 \vee \varphi_1$ iff $\sigma_1, \sigma_2 \models \varphi_0$ or $\sigma_1, \sigma_2 \models \varphi_1$
- $\sigma_1, \sigma_2 \models \exists x.\, \varphi$ iff there is an $n \in \{0,1\}^*$ s.t.

  $$\sigma_1'(y) = \begin{cases} \sigma_1(y) & \text{if } x \neq y \\ n & \text{otherwise} \end{cases}$$

  and $\sigma_1', \sigma_2 \models \varphi$
- $\sigma_1, \sigma_2 \models \exists X.\, \varphi$ iff there is an $S \subseteq \{0,1\}^*$ s.t.

  $$\sigma_2'(Y) = \begin{cases} \sigma_2(Y) & \text{if } X \neq Y \\ S & \text{otherwise} \end{cases}$$

  and $\sigma_1, \sigma_2' \models \varphi$

Example 10.3. Some examples for S2S formulas:

a) Node x is a prefix of node y:

$$x \leq y \quad :\equiv \quad \forall X.\, y \in X \wedge \forall z.\, (z0 \in X \to z \in X) \wedge (z1 \in X \to z \in X) \to x \in X$$

b) X is linearly ordered by $\leq$:

$$\mathit{Chain}(X) \quad :\equiv \quad \forall x.\, \forall y.\, x \in X \wedge y \in X \to (x \leq y \vee y \leq x)$$

c) X is a path:

$$\mathit{Path}(X) \quad :\equiv \quad \mathit{Chain}(X) \wedge \neg\exists Y.\, X \subseteq Y \wedge X \neq Y \wedge \mathit{Chain}(Y)$$

d) X is infinite:

$$\mathit{Inf}(X) \quad :\equiv \quad \exists Y.\, Y \neq \varnothing \wedge \forall y \in Y.\, \exists y' \in Y.\, \exists x' \in X.\, y < y' \wedge y < x'$$

Next, we analyze the connection of S2S definable languages and languages recognizable via tree automata over binary trees. We show that every S2S-definable language is recognizable by a parity tree automaton and vice versa.

Theorem 10.4. *For each parity tree automaton $\mathcal{A}$ over infinite 2^{V_2}-labeled binary trees, there is an S2S formula $\varphi_{\mathcal{A}}$ over V_2 such that $\mathcal{L}(\varphi_{\mathcal{A}}) = \mathcal{L}(\mathcal{A})$.*

Proof. For a parity tree automaton $\mathcal{A} = (\Sigma, Q, q_0, T, \text{PARITY}(c))$ we define an S2S formula $\varphi_{\mathcal{A}}$ such that $\mathcal{L}(\varphi_{\mathcal{A}}) = \mathcal{L}(\mathcal{A})$. Let $Q = \{q_0, q_1, \dots q_m\}$ and let $c : Q \to \{0, \dots, k\}$ for some $k \in \mathbb{N}$. For an input tree t, we "guess" an accepting run tree of $\mathcal{A}$:

$$\varphi_{\mathcal{A}} \;:\equiv\; \exists R_{q_0}, R_{q_1}, \dots R_{q_m}.\; \varphi_{part} \wedge \varphi_{init} \wedge \varphi_{trans} \wedge \varphi_{accept}$$

where

$$\begin{aligned}
\varphi_{part} &:\equiv \forall x. \bigvee_{q \in Q} State_q(x) \\
\varphi_{init} &:\equiv State_{q_0}(\varepsilon) \\
\varphi_{trans} &:\equiv \forall x. \bigvee_{(q,A,q_0',q_1') \in T} State_q(x) \wedge \bigwedge_{V \in A} x \in V \wedge \bigwedge_{V \notin A} x \notin V \wedge \\
&\qquad State_{q_0'}(x0) \wedge State_{q_1'}(x1) \\
\varphi_{accept} &:\equiv \forall X.\, Path(X) \to Parity(X) \\
State_q(x) &:\equiv x \in R_q \;\wedge \bigwedge_{q' \in Q \setminus \{q\}} x \notin R_{q'} \\
InfOcc_q(Y) &:\equiv \exists X. X \subseteq Y \wedge X \subseteq R_q \wedge Inf(X) \\
Parity(X) &:\equiv \bigvee_{i \in \{0,2,4,\dots,k\}} \bigvee_{c(q)=i} InfOcc_q(X) \wedge \bigwedge_{c(q)>i} \neg InfOcc_q(X)
\end{aligned}$$

□

Theorem 10.5. *Every S2S-definable language is recognizable by a parity tree automaton.*

Proof. Analogously to Lemma 7.1, we focus on a restricted sublogic (S2S_0), where the equalities are restricted to the following types:

$$x = \varepsilon \qquad x = y0 \qquad x = y1 \qquad x \in Y \qquad x = y$$

We inductively translate S2S_0 formulas to tree automata, as in the corresponding proof for S1S in Theorem 7.4. We only give an example here. We translate $x \in Y$ to $\mathcal{A} = (\Sigma, Q, q_0, T, \text{PARITY}(c))$, where

- $Q = \{q_0, q_1\}$
- $T = \{(q_0, A, q_0, q_1) \mid x \notin A\} \cup$
  $\{(q_0, A, q_1, q_0) \mid x \notin A\} \cup$
  $\{(q_0, A, q_1, q_1) \mid x \in A, Y \in A\} \cup$
  $\{(q_1, A, q_1, q_1) \mid x \notin A\}$
- $c(q_0) = 1, c(q_1) = 0.$

□

Since language-emptiness of parity tree automata is decidable (we can, for example, solve the emptiness game with McNaughton's algorithm), we obtain Michael Rabin's result that S2S is decidable.

Theorem 10.6 (Rabin's theorem)**.** *S2S is decidable.*

Generalizing from binary trees to n-ary trees, we obtain the *Monadic Second-Order Logic of n Successors* (SnS).

Theorem 10.7. *SnS is decidable.*

Proof. Repeat the proofs of Theorem 10.4 and Theorem 10.5 for automata on infinite n-ary trees (which are defined analogously to automata on infinite binary trees). □

Finally, we can even allow for trees where each node has countably infinitely many children. We obtain the *Monadic Second-Order Logic of ω Successors* (SωS).

Theorem 10.8. *SωS is decidable.*

Proof. We give an effective translation from SωS to S2S. First, let β be the bijection from $\mathbb{N}^*$ to $0 \cdot \{0,1\}^*$, defined via

$$\beta(w) = \begin{cases} \epsilon & \text{if } w = \epsilon \\ \beta(w')01^n & \text{if } w = w'n \end{cases}$$

This allows us to define a one-to-many relation R between SωS and S2S structures. We label a position $\beta(x)$ in the binary tree with σ if and only if x is labeld with σ in the ω-ary tree. We then can translate the SωS formula to S2S by transforming it to normal form first, and then using the following rules:

- $x = yn \quad \mapsto \quad x = y01^n$ for $n \in \mathbb{N}$
- $\exists X.\, \varphi \quad \mapsto \quad \exists X.\, (\forall y \in X.\, 0 \leq y) \wedge \varphi$ □

11 Computation Tree Logic

Temporal logics over trees are called *branching-time* logics, because the tree structure reflects the branching into the possible future behaviors of a system. In this section, we study Computation Tree Logic (CTL) as an example of a branching-time logic. We translate CTL formulas into tree automata, which will allow us to subsequently check emptiness, solve the model checking problem etc. with standard constructions. Analogously to our translation of LTL formulas into word automata in Section 8, we simplify the translation by going through alternating automata.

11.1 CTL

Computation Tree Logic (CTL) is a modal logic over infinite trees due to Clarke and Emerson (1981). The *syntax* of CTL is defined as follows:

- *CTL state formulas:*
  - $p \in AP$ — atomic proposition
  - $\neg\,\Psi$ and $\Psi_1 \wedge \Psi_2$ — negation and conjunction
  - $\mathsf{E}\,\varphi$ — there *exists* a path fulfilling φ
  - $\mathsf{A}\,\varphi$ — *all* paths fulfill φ
- *CTL path formulas:*
  - $\bigcirc\,\Psi$ — the next state fulfills Ψ
  - $\Psi_1\,\mathcal{U}\,\Psi_2$ — Φ holds until a Ψ-state is reached

where Ψ, Ψ_1 and Ψ_2 are state formulas, and φ is a path formula. Note that $\bigcirc$ and $\mathcal{U}$ alternate with A and E. A CTL *formula* Φ over a set AP of atomic propositions defines the following language of infinite 2^{AP}-labeled binary trees: $\mathcal{L}(\Phi) = \{(\mathcal{T}, t) \mid t, \varepsilon \models \Phi\}$, where $\models$ is the smallest relation that satisfies the following:

- *CTL state formulas:*

$$
\begin{array}{lll}
t, n \models p & \text{iff} & p \in t(n) \\
t, n \models \neg\,\Psi & \text{iff} & \neg(t, n \models \Psi) \\
t, n \models \Psi_1 \wedge \Psi_2 & \text{iff} & (t, n \models \Psi_1) \wedge (t, n \models \Psi_2) \\
t, n \models \mathsf{E}\,\varphi & \text{iff} & t, \pi \models \varphi \text{ for } \textit{some} \text{ path } \pi \text{ that starts in } n \\
t, n \models \mathsf{A}\,\varphi & \text{iff} & t, \pi \models \varphi \text{ for } \textit{all} \text{ paths } \pi \text{ that start in } n
\end{array}
$$

- *CTL path formulas:*

$$
\begin{array}{ll}
t, \pi \models \bigcirc\,\Psi & \text{iff } t, \pi[1] \models \Psi \\
t, \pi \models \Psi_1\,\mathcal{U}\,\Psi_2 & \text{iff } (\exists j \geq 0.\ t, \pi[j] \models \Psi_2 \wedge (\forall 0 \leq k < j.\ t, \pi[k] \models \Psi_1))
\end{array}
$$

11.2 Alternating Tree Automata

Analogously to the definition of alternating word automata in Section 8, we generalize nondeterministic tree automata to alternating tree automata by introducing a transition function that maps states and input letters to positive boolean formulas. For word automata, the boolean formulas refer to states, for tree automata to pairs of directions and states.

Definition 11.1. An alternating tree automaton over binary Σ-labeled trees is a tuple $\mathcal{A} = (\Sigma, Q, q_0, \delta, Acc)$, where

- Σ is a finite alphabet,
- Q is a finite set of states,
- $q_0 \in Q$ is an initial state,
- $\delta\colon Q \times \Sigma \to \mathbb{B}^+(\{0,1\} \times Q)$ is the transition function, and
- $Acc \subseteq Q^\omega$ is the acceptance condition.

For nondeterministic tree automata, run trees of automata over binary trees are again binary trees. This is no longer the case for alternating automata, because the automaton may branch to multiple states along the same direction of the input tree. This then results in multiple directions in the run tree that correspond to the same direction of the input tree. In the following, let D be some arbitrary finite set of directions, and let a tree $\mathcal{T}$ be some prefix-closed subset of D^*.

Definition 11.2. A *run* of an alternating automaton on a Σ-labeled binary tree $(\mathcal{T}, v)$ is a $\{0,1\}^* \times Q$-labeled tree $(\mathcal{T}, r)$ with the following properties:

- $r(\varepsilon) = (\varepsilon, q_0)$ and
- for all $n \in T$, if $r(n) = (m, q)$, then $\{(d, q') \mid (m \cdot d, q') = r(n') \text{ for some } n' \in \text{children}(n)\}$ satisfies $\delta(q, v(m))$.

Definition 11.3. A run (T, r) is *accepting* iff, for every infinite branch $n_0 n_1 n_2 \ldots$,

$$r(n_0)r(n_1)r(n_2)\ldots \in Acc.$$

As for nondeterministic automata, the acceptance mechanism of an alternating tree automaton can also be characterized via its *acceptance game*.

Definition 11.4. Let $\mathcal{A} = (\Sigma, Q, q_0, \delta, Acc)$ be an alternating tree automaton and let $(\mathcal{T}, t)$ be a Σ-labeled binary tree. Then the *acceptance game* of $\mathcal{A}$ on $(\mathcal{T}, t)$ is the game $\mathcal{G}_{\mathcal{A},t} = (\mathcal{A}', \text{Win}')$ with the infinite game arena $\mathcal{A}' = (V', V_0', V_1', E')$ and the winning condition Win' defined as follows:

- $V_0' = \{(w, q) \mid w \in \{0,1\}^*,\ q \in Q\}$
- $V_1' = \{(w, X) \mid w \in \{0,1\}^*,\ X \subseteq \{0,1\} \times Q\}$
- $E' = \{((w, q), (w, X)) \mid X \models \delta(q, t(w))\} \cup$
  $\{((w, X), (w', q')) \mid (d, q') \in X \text{ and } w' = w \cdot d\}$
- $\text{Win}' = \{(\alpha[0,0], q_0)(\alpha[0,0], X_0)(w[\alpha, 1], q_1)(\alpha[0,1], X_1)\ldots \mid$
  $\alpha \in \{0,1\}^\omega,\ q_0 q_1 \ldots \in Acc\}$

In model checking, the tree is often given as the unfolding of a finite graph. In this case, the acceptance game can be simplified to a finite game, where the positions refer to the nodes

of the graph instead of to the nodes of the tree.

Theorem 11.1. *An alternating tree automaton $\mathcal{A} = (\Sigma, Q, q_0, T, Acc)$ accepts an input tree $(\mathcal{T}, t)$ if and only if Player 0 wins the acceptance game $\mathcal{G}_{\mathcal{A},t} = (\mathcal{A}', \mathrm{Win}')$ from position (ε, q_0).*

As for word automata, alternation does not make the automata more expressive. Alternating Büchi tree automata can be translated into equivalent nondeterministic Büchi tree automata.

Theorem 11.2. *For every alternating Büchi tree automaton $\mathcal{A}$, there exists a nondeterministic Büchi tree automaton $\mathcal{A}'$ with $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.*

11.3 From CTL to Alternating Tree Automata

Similar to the construction of alternating word automata from LTL formulas in Section 8.2, we now translate CTL formulas to alternating tree automata.

Construction 11.1. Let Φ be an CTL formula. We construct the alternating Büchi tree automaton $\mathcal{A}_\Phi = (\Sigma, Q, \Phi, \delta, \textsc{büchi}(F))$ as follows:

- $Q = \mathrm{closure}(\Phi) := \{\Psi, \neg\Psi \mid \Psi \text{ is subformula of } \Phi\}$
- for $p \in AP$:

$$\delta(p, \sigma) = \begin{cases} \mathit{true} & \text{if } p \in \sigma \\ \mathit{false} & \text{if } p \notin \sigma \end{cases}$$

- $\delta(\Psi_1 \wedge \Psi_2, \sigma) = \delta(\Psi_1, \sigma) \wedge \delta(\Psi_2, \sigma)$
- $\delta(\mathrm{A} \bigcirc \Psi, \sigma) = (0, \Psi) \wedge (1, \Psi)$
- $\delta(\mathrm{E} \bigcirc \Psi, \sigma) = (0, \Psi) \vee (1, \Psi)$
- $\delta(\mathrm{A}(\Psi_1 \,\mathcal{U}\, \Psi_2), \sigma) = \delta(\Psi_2, \sigma) \vee (\delta(\Psi_1, \sigma) \wedge (0, \mathrm{A}(\Psi_1 \,\mathcal{U}\, \Psi_2)) \wedge (1, \mathrm{A}(\Psi_1 \,\mathcal{U}\, \Psi_2))$
- $\delta(\mathrm{E}(\Psi_1 \,\mathcal{U}\, \Psi_2, \sigma) = \delta(\Psi_2, \sigma) \vee (\delta(\Psi_1, \sigma) \wedge (0, \mathrm{E}(\Psi_1 \,\mathcal{U}\, \Psi_2)) \vee (1, \mathrm{E}(\Psi_1 \,\mathcal{U}\, \Psi_2))$
- $\delta(\neg\Psi, \sigma) = \overline{\delta(\Psi, \sigma)}$
- $F = \{\neg\mathrm{A}(\Psi_1 \,\mathcal{U}\, \Psi_2), \neg\mathrm{E}(\Psi_1 \,\mathcal{U}\, \Psi_2) \in \mathrm{closure}(\Phi)\}$.

12 Summary

We conclude with a summary of the course's main results.

12.1 Automata

The various types of automata studied in this course can be classified according to their *input* (words vs. trees), their *branching mode* (deterministic vs. nondeterministic vs. universal vs. alternating) and their *acceptance condition* (Büchi vs. co-Büchi vs. parity vs. Streett vs. Rabin vs. Muller).

Word automata. For automata on infinite words, we know from Büchi's Characterization Theorem (Theorem 3.6) that the languages that are recognizable by nondeterministic Büchi automata are exactly the ω-regular languages. We showed in Theorem 4.1 that deterministic Büchi automata are strictly less expressive: the language $(a+b)^*b^\omega$ is not recognizable by a deterministic Büchi automaton. In Problem 4.5 of the tutorial we showed that nondeterministic co-Büchi automata are also strictly less expressive: the language $(a^*b)^\omega$ cannot be recognized by a nondeterministic co-Büchi automaton. Universal co-Büchi automata are the dual of nondeterministic Büchi automata. Since nondeterministic Büchi automata are closed under complement (Theorem 5.1), universal co-Büchi automata therefore recognize the ω-regular languages. Likewise, because universal Büchi automata are the dual of nondeterministic co-Büchi automata, they are also strictly less expressive. Alternating Büchi automata and alternating co-Büchi automata recognize the ω-regular languages because the include nondeterministic Büchi automata and universal co-Büchi automata, respectively.

By McNaughton's theorem (Theorem 6.5), deterministic Muller automata (and, hence, also nondeterministic, universal, and alternating Muller automata) are as expressive as nondeterministic Büchi automata.

	Büchi	co-Büchi	Muller
deterministic	–	–	+
nondeterministic	+	–	+
universal	–	+	+
alternating	+	+	+

Tree automata. For automata on infinite trees, we showed, in Problem 13.2 of the tutorial, that deterministic parity tree automata are strictly less expressive than nondeterministic parity tree automata: the tree language over $\{a,b\}$-labeled binary trees that consists of all trees that have at least one a-labeled node cannot be recognized by a deterministic parity tree automaton. In Problem 13.5 of the tutorial, we showed that nondeterministic Büchi tree automata are strictly less expressive than nondeterministic parity tree automata: The tree language over $\{a,b\}$-labeled binary trees that consists of all trees where every branch has only finitely many a-labeled nodes cannot be recognized by a nondeterministic Büchi automaton.

Alternating Büchi tree automata have the same expressiveness as nondeterministic Büchi tree automata. By duality, alternating (and universal) co-Büchi tree automata are also less expressive than parity tree automata.

	Büchi	co-Büchi	parity
deterministic	–	–	–
nondeterministic	–	–	+
universal	–	–	+
alternating	–	–	+

12.2 Characterization Theorems

We proved several characterization theorems.

- An ω-language L is *Büchi recognizable* iff L is ω-regular (Büchi's Characterization Theorem, Theorem 3.6).
- An ω-language L is recognizable by a *deterministic Büchi automaton* iff there is a regular language $W \subseteq \Sigma^*$ s.t. $L = \overrightarrow{W}$ (Theorem 4.2).
- An ω-language L is recognizable by a *deterministic Muller* automaton iff L is a boolean combination of languages $\overrightarrow{W}$ where $W \subseteq \Sigma^*$ is regular (Theorem 6.4).

12.3 Translating Branching Modes

We translated automata with more complex branching modes to automata with simpler branching modes.

- *nondeterministic Büchi word* automata $\rightarrow$ *deterministic Muller word* automata (Safra's construction, Construction 6.7).
- *alternating Büchi word* automata $\rightarrow$ *nondeterministic Büchi word* automata (Miyano and Hayashi's construction, Construction 8.2).
- *alternating Büchi tree* automata $\rightarrow$ *nondeterministic Büchi tree* automata (Theorem 11.2).

12.4 Translating Acceptance Conditions

We translated automata with different acceptance conditions into each other.

- Büchi, co-Büchi, parity $\rightarrow$ *parity, Rabin, Streett*
  Since the acceptance conditions on the left are special cases of the acceptance conditions on the right, these translations are simple (cf. Problem 5.4 in the tutorial).
- Büchi, co-Büchi, Rabin, Streett, parity $\rightarrow$ *Muller*
  Since the acceptance conditions on the left can be expressed as Muller conditions, these translations are also conceptually simple; the Muller condition may, however, be exponentially large (cf. Theorem 6.1 and Problem 5.4 in the tutorial).
- Muller $\rightarrow$ *parity*
  This construction involves a modification of the state space of the automaton (latest appearence record construction, Problem 6.4 in the tutorial).

12.5 Automata and Games

We studied the connection between automata and games with two useful games defined by automata:

- In the *acceptance* game $\mathcal{G}_{\mathcal{A},t}$ of a nondeterministic or alternating tree automaton $\mathcal{A}$ (Definition 10.4), Player 0 wins iff $\mathcal{A}$ accepts t.
- In the *emptiness* game $\mathcal{G}_{\mathcal{A}}$ of a nondeterministic tree automaton $\mathcal{A}$ (Definition 10.5), Player 0 wins iff the language of $\mathcal{A}$ is non-empty.

If the alphabet consists of a single letter, the two games are equivalent. Applications of the game-based view are the complementation of tree automata (Theorem 10.3) and algorithms for checking language emptiness of tree automata.

12.6 Determinacy

A game is memoryless determined if, from each position, one of the players has a memoryless winning strategy.

- Reachability, Büchi, co-Büchi, parity games are *memoryless determined* (Theorems 9.1, 9.2, 9.3).

- Muller, Streett, Rabin games are determined, but *not memoryless determined* (Problem 11.3 in the tutorial).

An implication of these results is that memoryless runs suffice for alternating Büchi, co-Büchi, and parity word automata. We proved this explicitly for alternating Büchi automata in Theorem 8.2.

12.7 Logics

We studied logics over words and trees.

- Logics over words: QPTL, S1S, and WS1S have the same expressiveness as Büchi word automata (Theorems 7.2, 7.3, 7.4, and 7.5), LTL is less expressive (Theorem 7.1).

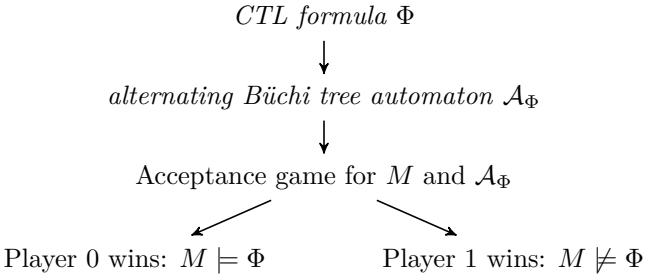
- Logics over trees: S2S has the same expressiveness as parity tree automata over binary trees (Theorems 10.4 and 10.5), CTL is less expressive.

An implication if these results is that the satisfiability problems for LTL, QPTL, S1S, WS1S, CTL, S2S are all decidable. To check if a given formula is satisfiable, translate it to the equivalent automaton and check if the language of the automaton is non-empty.

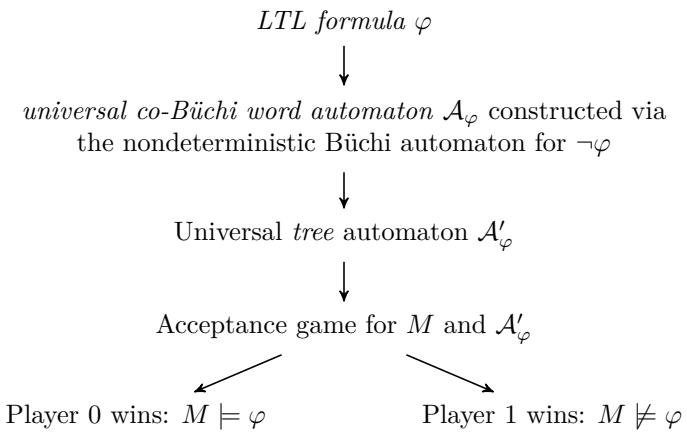
12.8 Model Checking and Synthesis

We can use the standard constructions to solve the model checking and synthesis problems for various logics.

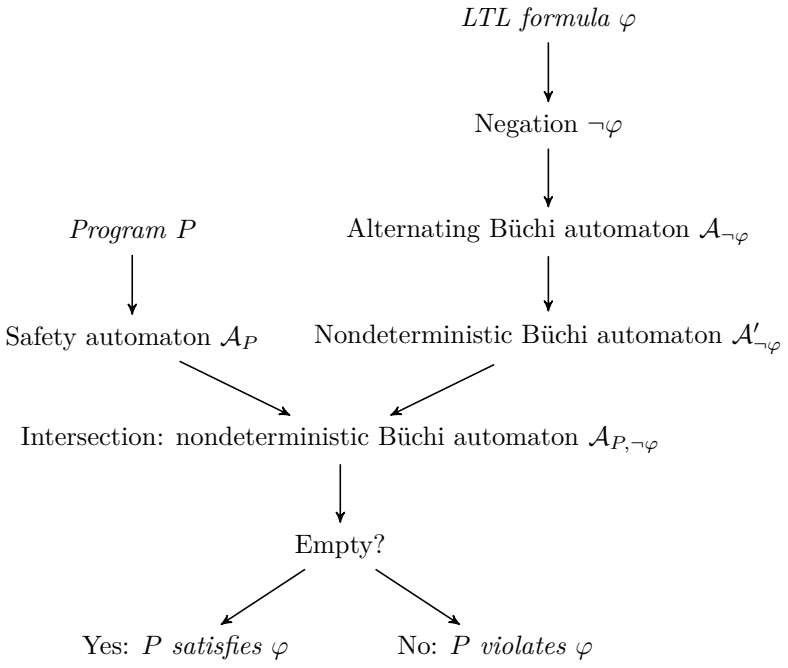
CTL model checking. Does a given transition system M satisfy an CTL formula Φ?



LTL model checking. Does a given transition system M satisfy an LTL formula φ?



An alternative view on LTL model checking, which avoids tree automata, is the following:



LTL synthesis. Does there exist a transition system M that satisfies an LTL formula φ over inputs I and outputs O?