Automata, Logic and Games

C.-H. L. Ong

January 28, 2015

Contents

0 Automata, Logic and Games **1**

0.1 Aims and Prerequisites . . . 1

0.2 Motivation . . . 2

0.3 Example: Modelling a Lift Control . . . 2

1 Büchi Automata **5**

1.1 Definition and Examples . . . 5

1.2 Closure Properties . . . 8

1.3 ω-Regular Expressions . . . 11

1.4 Decision Problems and their Complexity . . . 12

1.5 Determinisation and McNaughton's Theorem . . . 16

Problems . . . 22

2 Linear-time Temporal Logic **25**

2.1 Motivating Example: Mutual Exclusion Protocol . . . 25

2.2 Kripke Structures . . . 26

2.3 Syntax and Semantics . . . 27

2.4 Translating LTL to Generalised Büchi Automata . . . 30

2.5 The LTL Model Checking Problem and its Complexity . . . 33

2.6 Expressive Power of LTL . . . 38

Problems . . . 40

3 S1S **45**

3.1 Introduction . . . 45

3.2 The logical system S1S . . . 45

3.3 Semantics of S1S . . . 46

3.4 Büchi-Recognisable Languages are S1S-Definable . . . 48

3.5 S1S-Definable Languages are Büchi-Recognisable . . . 49

Problems . . . 51

4 Modal Mu-Calculus **53**

4.1 Knaster-Tarski Fixpoint Theorem . . . 53

4.2 Syntax of the Modal Mu-Calculus . . . 56

4.3 Labelled Transition Systems . . . 57

4.4 Syntactic Approximants Using Infinitary Syntax . . . 58

4.5 Intuitions from Examples . . . 60

4.6 Alternation Depth Hierarchy . . . 61

4.7 An Interlude: Computational Tree Logic (CTL) . . . 62

Problems . . . 65

5 Games and Tableaux for Modal Mu-Calculus 67

5.1 Game Characterisation of Model Checking . . . 67
5.2 Proof of the Fundamental Semantic Theorem . . . 72
5.3 Tableaux for modal mu-calculus . . . 75
5.4 Parity Games . . . 80
5.5 Solvability and Determinacy for Finite Parity Games . . . 81
5.6 Muller Games . . . 85
Problems . . . 88

6 Tree Automata, Rabin's Theorems and S2S 91

6.1 Trees and Non-deterministic Tree Automata . . . 91
6.2 Non-deterministic Parity Tree Automata . . . 93
6.3 Alternating Parity Tree Automata . . . 95
6.4 Closure Properties . . . 98
6.5 S2S and Rabin's Tree Theorem . . . 100
Problems . . . 102

Bibliography 102

A Ordinals and Transfinite Induction: A Primer 107

Chapter 0

Automata, Logic and Games

0.1 Aims and Prerequisites

To introduce the mathematical theory underpinning the computer-aided verification of computing (more generally *reactive*) systems.

- *Automata* on infinite words and trees as a model of computation of state-based systems.
- *Logical systems* such as temporal and modal logics for specifying correctness properties.
- *Two-person games* as a mathematical model of the interactions between a system and its environment.

Prerequisites

- Logic: 1st/2nd year *Logic and Proofs*, or *B1 Logic*
- Computability and Complexity: *Computational Complexity*

Connexions with other DCS courses This course can be viewed as a follow-up of *Computer-Aided Formal Verification*, emphasising the logical and algorithmic foundations. In addition there are several points of contact with *Software Verification*, and *Theory of Data and Knowledge Bases.*

Bibliography Many papers (and some book chapters) in the following can be viewed on the Web.

- (Bradfield and Stirling, 2007) [Must-read for modal mu-calculus.]
- (Khoussainov and Nerode, 2001) [Useful general reference for Büchi automata and S1S.]
- (Grädel et al., 2002) [Encyclopaedic, but uneven quality.]
- (Stirling, 2001) [Good for modal mu-calculus and parity games.]
- (Thomas, 1990) [Quite standard reference, but a little dated.]
- (Thomas, 1997) [Excellent reference for the relevant parts of the course.]
- (Vardi, 1996) [Easy to read; covers Büchi automata and LTL, but takes a different approach.]

Course webpage Lecture slides, exercises for the problem classes, resources, and administrative details of the course will be posted at the course webpage http://www.cs.ox.ac.uk/teaching/materials13-14/automatalogicgames/

0.2 Motivation

Reactive systems are computing systems that interact indefinitely with their environment. Typical examples are air traffic control systems, programs controlling mechanical devices such as trains and planes, ongoing processes such as nuclear reactors, operating systems and web servers.

Modelling Reactive Systems as Games There are different ways to model reactive systems. *Abstractly* we can model a reactive system by a two-player game:

- Player 0 (or Éloïse) representing the *System*
- Player 1 (or Abelard) representing the *Environment*

Desirable correctness properties of the System are coded as *winning conditions* for Éloïse. A strategy is *winning* for a player if it results in a win for the player, no matter what strategy is played by the other player. Winning strategies for Éloïse correspond to methods of constructing the System. Strategies are algorithms in abstract (and "neutral") form.

0.3 Example: Modelling a Lift Control

Assume a building of 8 levels.

Game perspective A 2-player game.

- Player 0 (Éloïse): Lift controller
- Player 1 (Abelard): Users

System state described by:

- A set of level numbers that have been requested, represented by a bit vector $(b_1, \cdots, b_8) \in \mathbb{B}^8$ whereby $b_i = 1$ iff level i has been requested.
- A level number $i \in \{1, \cdots, 8\}$ for the current position of the lift.
- A number (0 or 1) indicating whose turn it is.

State space of the system is $\mathbb{B}^8 \times \{1, \cdots, 8\} \times \{0, 1\}$.

State-transition graph A directed (bipartite) graph: vertices are states, and edges are transitions.

Transitions Two kinds: arrows from 0-states (Player 0's turn to play) to 1-states (Player 1's turn to play), and vice versa.

- $(b_1, \cdots, b_8, i, 0) \longrightarrow (b'_1, \cdots, b'_8, i', 1)$ such that $i \neq i', b'_{i'} = 0$ and $b'_j = b_j$ for $j \neq i'$.
  The actions involved are: door is closed, movement of lift, door is opened, and movement of people.

- $(b_1, \cdots, b_8, i, 1) \longrightarrow (b'_1, \cdots, b'_8, i, 0)$ such that $b_j \leq b'_j$ for all $j \in \{ 1, \cdots, 8 \}$.
  The actions are: Users push buttons.

Winning conditions (= correctness properties)

Example properties

1. Every requested level will be served eventually.
2. The lift will return to level 1 infinitely often.
3. When the top level (where the boss lives!) is requested, the lift serves it immediately and expeditiously (i.e. does not stop on the way there).
4. While moving in one direction, the lift will stop at every requested level, unless the top level is requested.

Some key questions

1. Is there a lift-control (0-strategy) that can meet all the requirements (winning conditions)?
   Does a winning strategy exist?
   Correctness conditions are typically encoded as logical formulas.
2. How much memory does the control need? Is finite memory enough?
   Is there a finite-state[1] *winning strategy (i.e. one that uses only a finite amount of memory)?*
3. Is there a method that can automatically derive a lift control from a given state-transition graph and a given set of winning conditions?
   Is the winning strategy effectively constructible?

[1]A model of computation is *finite state* if it has only finitely many possible configurations. Thus finite-state automata are finite state, but pushdown automata and Turing machines are infinite state.

Chapter 1

Büchi Automata

Synopsis

Definition and examples. Büchi automata are not determinisable. Closure properties of Büchi-recognisable languages. Büchi's proof of complementation via Ramsey's Theorem. Büchi's characterisation and ω-regular expressions. Decision problems and their complexity: non-emptiness is NL-complete, and universality is PSPACE-complete. Other acceptance conditions: Muller, Rabin, Streett and Parity. Determinisation and McNaughton's Theorem.

Notations Let U be a set.

- We write U^* to mean the set of finite sequences (or *words* or *strings*) of elements of U. The empty word is denoted ϵ. I.e. U^* is the free monoid over U: the associative binary operation is string concatenation $(u, v) \mapsto u \cdot v$ (or simply $u\,v$, eliding $\cdot$) and the identity is ϵ.
- We write U^ω to mean the set of infinite sequences (or ω-*words* or ω-*strings*) of elements of U. An ω-word is represented as a function from ω to U, ranged over by α, β, ρ, etc. Thus the map α represents the infinite word $\alpha(0)\,\alpha(1)\,\alpha(2)\cdots$.

Let $u \in U^*$ and $w \in (U^* \cup U^\omega)$. We say that u is a *prefix* of w, written $u \leq w$, just if $w = u\,v$ for some $v \in (U^* \cup U^\omega)$. We write $u < w$ just if $u \leq w$ and $u \neq w$.

Henceforth we assume a finite *alphabet* Σ i.e. a finite set of symbols (or letters). Subsets of Σ^* are called $*$-*languages*; subsets of Σ^ω are called ω-*languages*.

1.1 Definition and Examples

We use automata to define ω-languages.

Definition 1.1. A (non-deterministic) *Büchi automaton* is a quintuple $A = (Q, \Sigma, q_0, \Delta, F)$ where

- Q is a finite set of states
- Σ is a finite alphabet
- $q_0 \in Q$ is the initial state
- $\Delta \subseteq Q \times \Sigma \times Q$ is a transition relation
- $F \subseteq Q$ is the set of *final* (or *accepting*) states.

In case Δ is a function $Q \times \Sigma \longrightarrow Q$, we say that A is *deterministic*, and write δ for the function.

It is helpful to think of a Büchi automaton as a finite, labelled directed graph: each edge is labelled with an element of Σ; and the vertex labels are "initial" (labelling a unique vertex) and "final" (labelling a subset of vertices).

Language Recognised by a Büchi Automaton

A *run* of A on an ω-word $\alpha \in \Sigma^\omega$ is an infinite sequence of states $\rho = \rho(0)\ \rho(1)\ \rho(2) \cdots$ such that $\rho(0) = q_0$, and for all $i \geq 0$, we have

$$(\rho(i), \alpha(i), \rho(i+1)) \ \in \ \Delta.$$

In words, a *run* on α is an infinite path in the directed graph A, starting from the initial vertex, whose labels on the edges trace out the ω-word α.

Note that if A is deterministic then every word has a *unique* run.

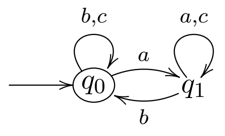
A run ρ on α is *accepting* just if there is a final state that occurs infinitely often in ρ; equivalently (because F is finite) $\mathsf{inf}(\rho) \cap F \neq \varnothing$, writing $\mathsf{inf}(\rho)$ for the set of states that occur infinitely often in ρ.

An ω-word α is *accepted* by an automaton A just if there is an accepting run of A on α. The *language recognised* by A, written $L(A)$, is the set of ω-words accepted by A. An ω-language is *Büchi recognisable* just if it is recognised by some Büchi automaton.

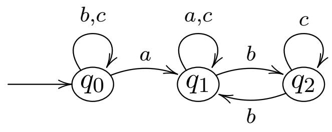
Convention When drawing automata as graphs, we circle the final states, and indicate the initial state by an arrow.

Example 1.1. Set $\Sigma = \{ a, b, c \}$.

(i) $L_1 \subseteq \Sigma^\omega$ consists of ω-words in which after every occurrence of a there is some occurrence of b.



(ii) L_2 consists of ω-words in which between every two occurrences of a, there is an even number of b.



When the automaton reaches state q_1 (respectively q_2), it has read an even (respectively odd) number of b since the last a.

Is L_1 recognised by a deterministic automaton? What about L_2?

Example 1.2 (A Non-Determinisable Büchi Automaton)**.** The Büchi-recognisable language L_3 consisting of ω-words over $\{ 0, 1 \}$ that have only finitely many occurrences of 1 is not recognised by any deterministic Büchi automaton.

Suppose, for a contradiction, L_3 is recognised by a deterministic automaton

$$A = (Q, \{ 0, 1 \}, q_0, \delta, F).$$

It follows that δ extends to a function $Q \times \{ 0, 1 \}^* \longrightarrow Q$. Since A has an accepting run on 0^ω, we have $\delta(q_0, 0^{n_1}) \in F$ for some n_1. Let $u_1 \in Q^*$ be the "run" for 0^{n_1}.

Similarly, since A has an accepting run on $0^{n_1}10^\omega$, we have $\delta(q_0, 0^{n_1}10^{n_2}) \in F$ for some n_2. Let $u_2 \in Q^*$ be the "run" for $0^{n_1}10^{n_2}$. Note that $u_1 \leq u_2$.

In this fashion, we obtain an infinite sequence of numbers $n_1, n_2, \cdots$, and an infinite ascending chain $u_1 \leq u_2 \leq u_3 \cdots$ whose limit is an accepting run of A on the ω-word $0^{n_1}10^{n_2}10^{n_3}10^{n_4}1\cdots$, which is a contradiction. □

Where does the argument break down if A is not deterministic?

Exercise 1.1. Construct a Büchi automaton that recognises (i) L_3 (ii) $\overline{L_3} = \{ 0, 1 \}^\omega \setminus L_3$.

Example 1.3. Construct a Büchi automaton for the language L_4 consisting of ω-words α over $\{ a, b, c \}$ such that α contains the segments $a\,b$ and $a\,c$ infinitely often, but $b\,c$ only finitely often.

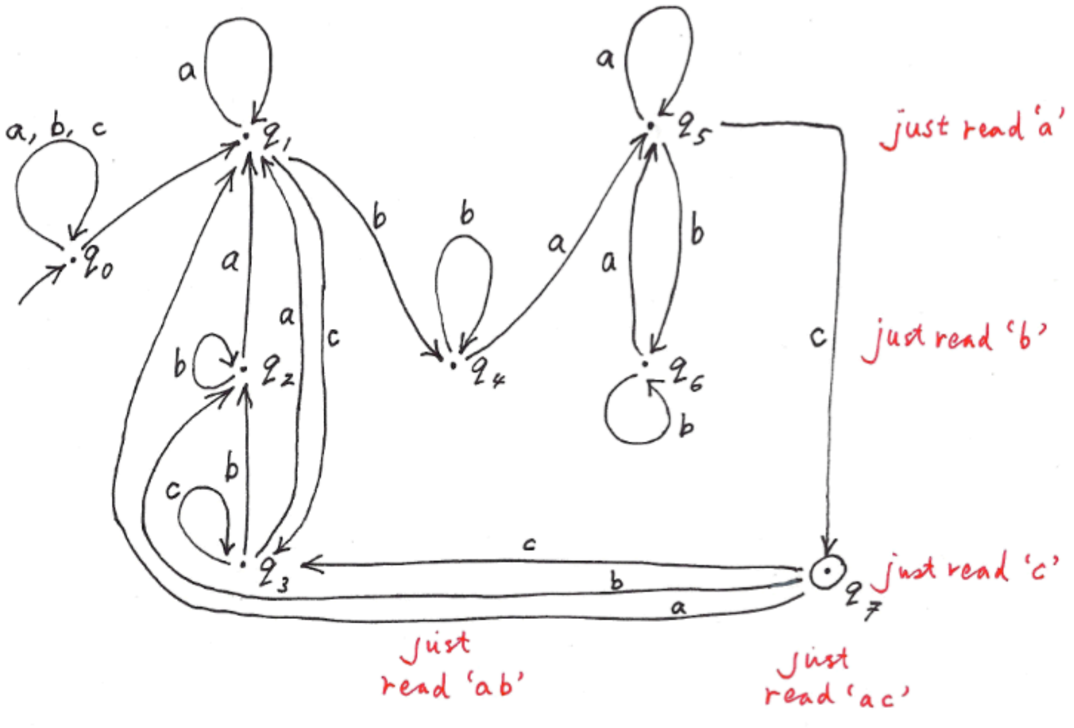


Figure 1.1: An automaton accepting infinitely many $a\,b$ and $a\,c$ but only finitely many $b\,c$.

We argue that the automaton in Figure 1.1 recognises L_4 as follows. By construction:

- when the automaton reaches the states q_1 and q_5, it has just read a
- when it reaches the states q_2, q_4 and q_6, it has just read b
- when it reaches the states q_3 and q_7, it has just read c.

Consequently, after leaving q_0, the automaton is unable to read $b\,c$. Further

- when the automaton reaches state q_4, it has just read $a\,b$
- when it reaches state q_7, it has just read $a\,c$.

It then remains to observe that q_7 is the (only) final state, and after leaving it, the automaton must visit q_4 before it is able to revisit q_7.

1.2 Closure Properties

Büchi recognisable languages are closed under many operations, including all boolean operations i.e. union, intersection and complementation.

Proposition 1.1. *(i) If $U \subseteq \Sigma^*$ is regular then U^ω is Büchi recognisable.*

(ii) If $U \subseteq \Sigma^*$ is regular and $L \subseteq \Sigma^\omega$ is Büchi recognisable then

$$U \cdot L := \{\, u \cdot \alpha \mid u \in U, \alpha \in L \,\}$$

is Büchi recognisable.

(iii) If L_1 and L_2 are Büchi-recognisable ω-languages, so are $L_1 \cup L_2$ and $L_1 \cap L_2$.

Exercise 1.2. Prove (i) and (ii) of the proposition.

Closure under Union

As a first attempt, use the standard "union construction" in automata for finite words, but note that we can't use ϵ-label edges. Thus for $i = 1, 2$, suppose L_i is recognised by $A_i = (Q_i, \Sigma, q_0^i, \Delta_i, F_i)$. Assume Q_1 and Q_2 are disjoint. Then $L_1 \cup L_2$ is recognised by the Büchi automaton

$$(Q_1 \cup Q_2 \cup \{\, q_0 \,\}, \Sigma, q_0, \Delta, F_1 \cup F_2)$$

where q_0 is a fresh state, and

$$\begin{array}{rcl} \Delta & := & \Delta_1 \cup \Delta_2 \\ & \cup & \{\, (q_0, a, q) \mid a \in \Sigma, (q_0^1, a, q) \in \Delta_1 \,\} \\ & \cup & \{\, (q_0, a, q) \mid a \in \Sigma, (q_0^2, a, q) \in \Delta_2 \,\} \end{array}$$

I.e. for each a-transition from q_0^i to q, we add a fresh a-transition from q_0 to q (for $i = 1, 2$).

Closure under Intersection

Suppose L_i is accepted by $A_i = (Q_i, \Sigma, \Delta_i, q_0^i, F_i)$, for $i = 1, 2$. As a first attempt, run the two automata synchronously i.e. in lockstep. Following finite automata for finite words, construct the product automaton

$$(Q_1 \times Q_2, \Sigma, \Delta, (q_0^1, q_0^2), F_1 \times F_2)$$

where $((p, q), a, (p', q')) \in \Delta$ iff $(p, a, p') \in \Delta_1$ and $(q, a, q') \in \Delta_2$. This does not work because we cannot guarantee that the final states of A_1 and A_2 are visited infinitely often *simultaneously.*

The point is that we need to ensure infinite alternation of a F_1-state and a F_2-state. Thus we construct a product automaton and cycle through the following:

1. Wait for an F_1-state in first component.
2. When an F_1-state is encountered in first component, wait for an F_2-state in second component.
3. When an F_2-state is encountered in second component, go to 1.

The Modified Intersection Automaton Work with state-set $Q_1 \times Q_2 \times \{1, 2\}$. Form modified product automaton:

$$A' \ := \ (Q_1 \times Q_2 \times \{1,2\},\ \Sigma,\ \Delta',\ (q_0^1, q_0^2, 1),\ Q_1 \times F_2 \times \{2\})$$

where: for every $(p, a, p') \in \Delta_1$ and every $(q, a, q') \in \Delta_2$, we have

- $((p, q, 1), a, (p', q', 1)) \in \Delta'$ if $p \notin F_1$
- $((p, q, 1), a, (p', q', 2)) \in \Delta'$ if $p \in F_1$
- $((p, q, 2), a, (p', q', 2)) \in \Delta'$ if $q \notin F_2$
- $((p, q, 2), a, (p', q', 1)) \in \Delta'$ if $q \in F_2$

It follows that a run ρ of A' on α simulates runs $\rho_1 = \pi_1^*(\rho)$ of A_1 on α and $\rho_2 = \pi_2^*(\rho)$ of A_2 on α—where $\pi_1 : (p, q, j) \mapsto p$ and π_1^* is the point-wise extension—such that ρ visits a state in $Q_1 \times F_2 \times \{2\}$ infinitely often iff ρ_1 visits F_1 infinitely often and ρ_2 visits F_2 infinitely often. □

Closure under Complementation

Theorem 1.1 (Büchi 1960)**.** *If $L \subseteq \Sigma^\omega$ is Büchi recognisable (by A say), so is $\Sigma^\omega \setminus L$. Further the automaton recognising $\Sigma^\omega \setminus L$ can be effectively constructed from A.*

As a first attempt, consider the standard method to complement a finite-state automaton for finite words: first determinise A, then "invert" the final states. Unfortunately, *Büchi automata are not determinisable.* As we have seen, there are non-deterministic Büchi automata (for example, any automaton that recognises L_3 of Example 1.2) that are not equivalent to any deterministic automata.

Büchi's proof We follow the account in (Thomas, 1990) of the proof by Büchi (1960b). Let L be recognisable by a Büchi automaton A. We aim to show that both L and $\Sigma^\omega \setminus L$ are representable as finite unions of sets of the form $L_1 \cdot L_2^\omega$ where L_1 and L_2 are regular $*$-languages. (Note that it is relatively straightforward to prove the result for L alone: *cf.* Proposition 1.2.)

We shall construct L_1 and L_2 as congruence classes. We say that a relation $\sim \subseteq \Sigma^* \times \Sigma^*$ is a *congruence* just if $\sim$ is an equivalence relation such that whenever $u \sim u'$ and $v \sim v'$ then $u \cdot v \sim u' \cdot v'$. Set

$$\begin{aligned} W_{q,q'} &:= \{ w \in \Sigma^* \mid q \overset{w}{\Longrightarrow} q' \} \\ W^F_{q,q'} &:= \{ w \in \Sigma^* \mid q \overset{w}{\Longrightarrow}_F q' \} \end{aligned}$$

where $q \xRightarrow{a_1 \cdots a_n}_X q'$ with $X \subseteq Q$ means that exist $q_0, \cdots, q_n \in Q$ such that $q = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n = q'$, and $\{ q_0, \cdots, q_n \} \cap X \neq \varnothing$; and in case $X = Q$, we omit the subscript X from $q \overset{w}{\Longrightarrow}_X q'$. Define

$$w \sim_A w' \ := \ \forall q, q' \in Q. \big((w \in W_{q,q'} \leftrightarrow w' \in W_{q,q'}) \ \wedge \ (w \in W^F_{q,q'} \leftrightarrow w' \in W^F_{q,q'}) \big)$$

It is straightforward to see that $\sim_A$ is an equivalence relation over Σ^* which has a finite index (because Q is a finite set). It is an easy exercise to show that $\sim_A$ is a congruence.

The equivalence classes can be described as follows: for $w \in \Sigma^*$

$$\begin{array}{rcl} [w]_{\sim_A} & = & \bigcap_{q,q' \in Q, w \in W_{q,q'}} W_{q,q'} \ \cap \ \bigcap_{q,q' \in Q, w \notin W_{q,q'}} (\Sigma^* \setminus W_{q,q'}) \\ & \cap & \bigcap_{q,q' \in Q, w \in W^F_{q,q'}} W^F_{q,q'} \ \cap \ \bigcap_{q,q' \in Q, w \notin W^F_{q,q'}} (\Sigma^* \setminus W^F_{q,q'}) \end{array}$$

Since each $W^F_{q,q'}$ is regular, so is each equivalence class $[w]_{\sim_A}$.

Let $X \subseteq \Sigma^\omega$. We say that a congruence relation $\simeq \ \subseteq \Sigma^* \times \Sigma^*$ *saturates* X just if for all $u, v \in \Sigma^*$, if $[u]_\simeq \cdot [v]_\simeq^\omega \cap X \neq \varnothing$ then $[u]_\simeq \cdot [v]_\simeq^\omega \subseteq X$.

Lemma 1.1. *(i) $\sim_A$ saturates L*

(ii) $\sim_A$ saturates $\Sigma^\omega \setminus L$.

Exercise 1.3. Prove (i) and (ii) of the lemma.

Finally it suffices to prove the following.

Claim. Let $X \subseteq \Sigma^\omega$. If a congruence $\simeq$ saturates X and has finite index, then

$$X = \bigcup \{ [u]_\simeq \cdot [v]_\simeq^\omega \mid [u]_\simeq \cdot [v]_\simeq^\omega \cap X \neq \varnothing \}.$$

Assume $\simeq$ saturates X. Then "$\supseteq$" follows from the definition of saturation.

To prove "$\subseteq$", let $w \in X$. Define an equivalence relation $\approx_w \ \subseteq D \times D$ where $D := \{ (i,j) \in \omega^2 \mid i < j \}$ by

$$(i,j) \approx_w (i',j') \ := \ w[i,j] \simeq w[i',j']$$

where $w[i,j] = a_i \cdots a_{j-1}$ with $w = a_0 \, a_1 \ \cdots$. The index of $\approx_w$ is finite because the index of $\simeq$ is finite by assumption.

Now it follows from Ramsey's Theorem[1] that there is an infinite set $H = \{ i_0, i_1, i_2, \cdots \}$ with $i_0 < i_1 < i_2 < \cdots$ which is *homogeneous* for the map:

$$\{ i, j \} \mapsto [(i,j)]_{\approx_w}$$

assuming $i < j$. I.e. there is a pair (i, i') such that whenever $k < l$ then $(i,i') \approx_w (i_k, i_l)$. In particular all pairs (i_k, i_{k+1}) are in $[(i,i')]_{\approx_w}$. Thus

$$w = w[0, i_0] \cdot w[i_0, i_1] \cdot w[i_1, i_2] \cdot \cdots \in [w[0,i_0]]_\simeq \cdot ([w[i,i']]_\simeq)^\omega$$

as required. This completes the proof of the Claim, and hence the proof of Theorem 1.1.

Given a Büchi automaton A with n states, there are n^2 different pairs (q, q') and hence $O(2^{2n^2})$ different $\sim_A$-classes.

Exercise 1.4. Show that Büchi's complement automaton has a size bound of $O(2^{4n^2})$ states. *Cf.* (Pécuchet, 1986; Sistla et al., 1987).

[1] Let A be a set. We write $(A)_n := \{ B \subseteq A : |B| = n \}$.

Theorem 1.2 (Frank P. Ramsey 1930)**.** *Suppose $f : (\omega)_n \to \{ 0, 1, \cdots, k-1 \}$. Then there is an infinite set $A \subseteq \omega$ which is homogeneous for f i.e. f is constant on $(A)_n$.*

If we think of f as a k-colouring of the n-elements subsets of ω, then "A is homogeneous for f" means that f maps all n-element subsets of A to the same colour.

1.3 ω-Regular Expressions

Let $U \subseteq \Sigma^*$.

$$\begin{array}{rcl} U^* & := & \{\, w \in \Sigma^* \mid w = u_1\, u_2 \, \cdots u_n \text{ for some } n \geq 0, \text{ each } u_i \in U \,\} \\ U^+ & := & \{\, w \in \Sigma^* \mid w = u_1\, u_2 \, \cdots u_n \text{ for some } n \geq 1, \text{ each } u_i \in U \,\} \\ U^\omega & := & \{\, w \in \Sigma^\omega \mid w = u_1\, u_2 \, \cdots \text{ where each } u_i \in U \,\} \\ \lim U & := & \{\, w \in \Sigma^\omega \mid w(0) \cdots w(j) \in U \text{ for infinitely many } j \in \omega \,\} \end{array}$$

In words, $w \in \lim U$ just if U contains infinitely many prefixes of w.

Example 1.4. (i) Let $U_1 = 0110^* + (00)^+$. Then $\lim U_1$ consists of only two ω-words, namely, $0110000\cdots$ and $0000\cdots$.

(ii) Let $U_2 = 0^*1$. Then $\lim U_2 = \varnothing$.

Proposition 1.2 (Büchi 1960)**.** *A language $L \subseteq \Sigma^\omega$ is Büchi recognisable if and only if L is a finite union of sets of the form $J \cdot K^\omega$, where $J, K \subseteq \Sigma^*$ are regular and $\varnothing \neq K \subseteq \Sigma^+$ (and we may assume $K \cdot K \subseteq K$).*

Proof. Suppose L is recognised by $A = (Q, \Sigma, \Delta, q_0, F)$. For $p, q \in Q$, let $A_{p\,q}$ be the finite automaton $(Q, \Sigma, \Delta, p, \{\, q \,\})$. Write $L^*(A_{q\,q'})$ for the *finite-word language* recognised by $A_{q\,q'}$. Then

$$\begin{array}{rl} & \alpha \in \Sigma^\omega \text{ is accepted by } A \\ \text{iff} & \text{there exists a run } \rho \text{ with } \mathsf{inf}(\rho) \cap F \neq \varnothing \\ \text{iff} & \text{there exists } q \in F \text{ and } \alpha = u_0 u_1 u_2 \cdots \text{ where } u_0 \text{ is accepted by } A_{q_0\,q} \\ & \text{and for each } i \geq 1,\ u_i \text{ is non-empty and accepted by } A_{q\,q}. \end{array}$$

Hence $L = \bigcup_{q \in F} L^*(A_{q_0\,q}) \cdot (L^*(A_{q\,q}))^\omega$. □

Regular Expressions: A Revision

Fix a finite alphabet Σ and let a range over Σ. Regular expressions e are defined by the grammar:

$$e \ ::= \ \varnothing \mid \epsilon \mid a \mid e + e \mid e \cdot e \mid e^*$$

For simplicity $e \cdot f$ is often written as $e\,f$. We define the *denotation* of a regular expression $[\![\, e \,]\!] \subseteq \Sigma^*$ as follows.

$$\begin{array}{rcl@{\qquad\qquad}rcl} [\![\, \epsilon \,]\!] & = & \{\, \epsilon \,\} & [\![\, e \cdot f \,]\!] & = & [\![\, e \,]\!] \cdot [\![\, f \,]\!] \\ [\![\, \varnothing \,]\!] & = & \varnothing & [\![\, e^* \,]\!] & = & [\![\, e \,]\!]^* \\ [\![\, a \,]\!] & = & \{\, a \,\} & [\![\, e + f \,]\!] & = & [\![\, e \,]\!] \cup [\![\, f \,]\!] \end{array}$$

Let $w \in \Sigma^*$. We say that w *matches* e just if $w \in [\![\, e \,]\!]$.

Theorem 1.3 (Kleene)**.** *A set of finite words is recognisable by a finite-state automaton if, and only if, it is the denotation of a regular expression.* □

An *ω-regular expression* has the form

$$e_1 \cdot f_1^\omega + \cdots + e_n \cdot f_n^\omega$$

where $n \geq 0$, and $e_1, f_1, \cdots, e_n, f_n$ are regular expressions such that $\varnothing \neq [\![\, f_i \,]\!] \subseteq \Sigma^+$ for all $i \in \{\, 1, \ldots, n \,\}$.

The *denotation* of a ω-regular expression $[\![\, e \,]\!] \subseteq \Sigma^\omega$ is defined by the same clauses as regular expressions, and $[\![\, e^\omega \,]\!] := [\![\, e \,]\!]^\omega$. We say that an ω-language is *ω-regular* just if it is the denotation of a ω-regular expression.

Corollary 1.1. *A set of ω-words is Büchi recognisable if and only if it is ω-regular.*

Proof. Immediate consequence of Proposition 1.2. □

Example 1.5. (i) A regular expression for L_3 (i.e. the set of binary words containing only finitely many 1s) is $(0+1)^*0^\omega$.

(ii) A regular expression for $\overline{L_3}$ is $(0^*1)^\omega$.

For ω-regular expressions e and f, we say $e \equiv f$ just if $[\![\, e \,]\!] = [\![\, f \,]\!]$.

Lemma 1.2. *For $X, Y \subset \Sigma^*$*

(i) $(X+Y)^\omega \equiv (X^*Y)^\omega + (X+Y)^*X^\omega$

(ii) $(XY)^\omega \equiv X(YX)^\omega$

(iii) *For all $n > 0$, $(X^n)^\omega \equiv (X^+)^\omega \equiv X^\omega$.*

(iv) $X^\omega \equiv X^+X^\omega$.

Proof. Exercise □

1.4 Decision Problems and their Complexity

Decision problems for Büchi automata are worth studying because algorithms for solving these problems are basic building blocks for the construction of algorithmic solutions to the complex problems that arise in the verification of computing systems.

Non-Emptiness Problem Given a Büchi automaton A, is $L(A) \neq \varnothing$?

Proposition 1.3. *The non-emptiness problem for Büchi automata $A = (Q, \Sigma, \Delta, q_0, F)$ is decidable in time $O(|Q| + |\Delta|)$.*

Proof. We have:

$L(A) \neq \varnothing$

iff there is a path from q_0 to some $q \in F$, and there is a path from q back to itself

iff automaton A (*qua* digraph) has a *non-trivial* SCC which is reachable from q_0 and contains a final state q

Recall that a *strongly connected component* (SCC) of a directed graph is a maximal subgraph such that for every pair of vertices in the subgraph, there is a directed path from one vertex to the other.

There is a simple algorithm to decide the Non-Emptiness Problem. The idea is to find a "lasso" in the graph underlying the automaton: the base of the lasso is the initial state, and the loop must include a final state.

Algorithm: *Finding a Lasso*
Input: Büchi automaton $A = (Q, \Sigma, \Delta, q_0, F)$.
Output: YES, if $L(A) \neq \varnothing$; NO otherwise.

```
1. Determine the set $Q_0$ of states reachable from $q_0$ using (say) depth-first search.
2. Generate all non-trivial SCCs over $Q_0$; at the same time, check for containment
   of a final state.
3. If there is a non-trivial SCC that contains a final state, return YES; otherwise
   return NO.
```

Stages 1 and 2 require time $O(|Q| + |\Delta|)$ (Tarjan, 1972). □

In fact the non-emptiness problem is *complete for NL* (Non-deterministic Logspace).

An Interlude: NL and NL-Completeness

Recall that a non-deterministic Turing machine *accepts* an input word just if there is a computation path from the initial configuration to an accepting configuration.

Definition 1.2. A decision problem is *NL-complete* just if it is

(i) solvable in **NL** (i.e. decidable by a non-deterministic Turing machine using $O(\log n)$ space on a work tape, where n is the size of the input), and

(ii) NL-hard (i.e. every NL-solvable problem is logspace-reducible to it).

Intuitively **L** (Logspace) is the collection of problems that are solvable using (i) a constant number of pointers into the input (because each number in $\{0, \ldots, n-1\}$ can be represented in binary in at most $\log n$ bits) (ii) and a logarithmic number of boolean flags. See Michael Sipser's book (Sipser, 2005) and Immerman's book (Immerman, 1999) for a systematic treatment.

Proposition 1.4. *The Non-Emptiness Problem for Büchi automata is NL-complete.*

Proof. We give an algorithm in NL that checks if there is a final state which is reachable from the initial state q_0, and reachable from itself. To do this, we first guess a final state f (say), and then a path from q_0 to f, and from f to f.

To guess a path from states x to y:

1. Make x the current state.
2. Guess a transition from the current state, and make the target of the transition the new current state.
3. If the current state is y, STOP; otherwise repeat from step 2.

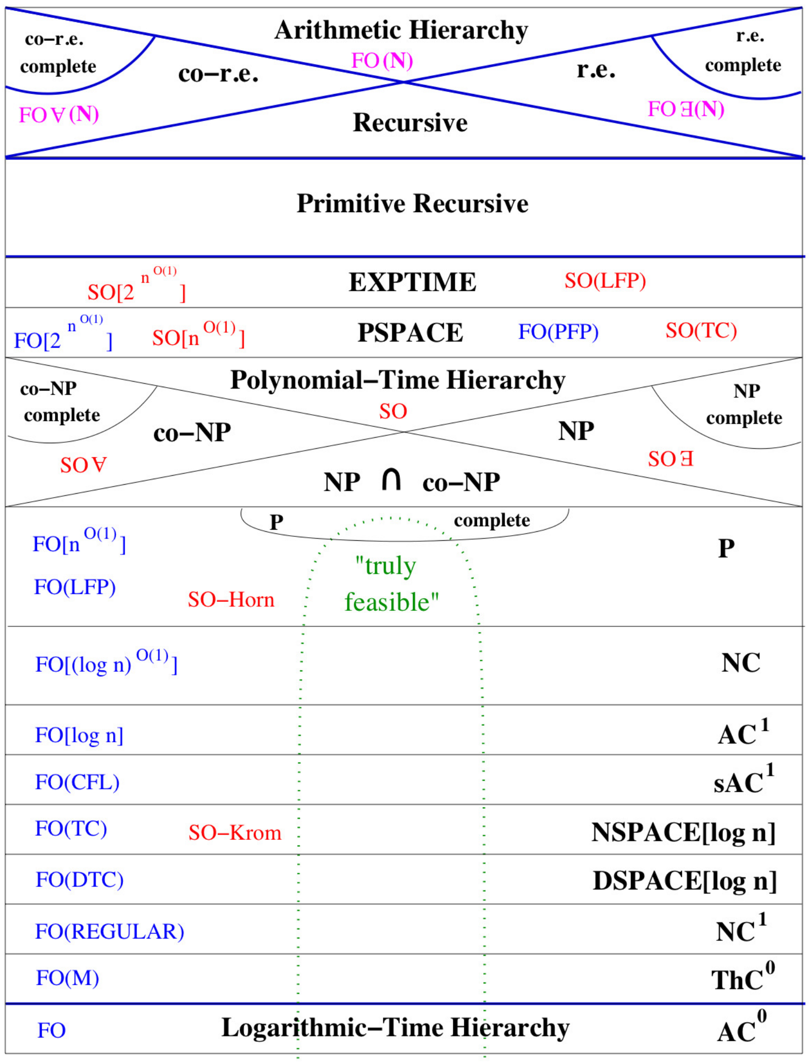


Figure 1.2: Immerman's World of Complexity Classes (Immerman, 1999)

The algorithm is in **NL**: at each stage, only 3 states are remembered (by 3 pointers into the input string).

NL-hardness is proved by reduction from the Graph Reachability Problem (Given nodes x and y in a finite directed graph, is y reachable from x?), which is NL-complete. □

Universality Problem Given a Büchi automaton A over Σ, is $L(A) = \Sigma^\omega$?

Proposition 1.5. *The Universality Problem is PSPACE-complete.*

To decide non-universality, given a Büchi automaton A, we could construct the complement automaton $\overline{A}$ (i.e. $L(\overline{A}) = \Sigma \setminus L(A)$), then use the non-emptiness algorithm on $\overline{A}$. Unfortunately this would give an algorithm that is exponential in space and time!

To show PSPACE decidability, we determinise the automaton (which may be of exponential size) but calculate the states only *on demand*, and look for a word that is not recognised.

PSPACE Hardness Proof We present a proof by Sistla et al. (1987), which is by reduction from the Universality Problem for Finite-State Automata (Given a FSA A, is $L^*(A) = \Sigma^+$?). The latter problem is PSPACE-complete (Meyer and Stockmeyer, 1972).

The idea is to define a transformation $L \subseteq \Sigma^* \mapsto L' \subseteq \Sigma^\omega$ such that whenever A is a FSA then $L(A)'$ is Büchi-recognisable; further $L(A) = \Sigma^*$ if and only if $L(A)' = \Sigma^\omega$.

Proof. Fix $\Sigma = \{ a^1, \cdots, a^n \}$. Given FSA $A = \langle \Sigma, Q, \Delta, q_0, F \rangle$, define two alphabets $\Sigma_i = \{ a_i^1, \cdots, a_i^n \}$ $(i = 1, 2)$. Consider automata $A_i = \langle \Sigma_i, Q, \Delta_i, q_0, F \rangle$ such that for $i = 1, 2$:

$$\forall q, q', j : \ (q, a_i^j, q') \in \Delta_i \quad \leftrightarrow \quad (q, a^j, q') \in \Delta$$

Thus A_1 and A_2 recognise the image of $L^*(A)$ over Σ_1 and Σ_2 respectively. Now define $L' \subseteq (\Sigma_1 \cup \Sigma_2)^\omega$ by

$$\begin{aligned} L' \ &:= \ (L_1 \, L_2)^\omega \ \cup \ (L_1 \, L_2)^* L_1^\omega \ \cup \ (L_1 \, L_2)^* L_2^\omega \\ &\cup \ (L_2 \, L_1)^\omega \ \cup \ (L_2 \, L_1)^* L_2^\omega \ \cup \ (L_2 \, L_1)^* L_1^\omega \end{aligned}$$

where $L_i := L^*(A_i)$.

Exercise 1.5. *Construct a Büchi automaton A' that recognises L', with size linear in that of A.*

Claim. *The FSA A is universal (i.e. $L^*(A) = \Sigma^+$) if and only if the Büchi automaton A' is universal.*

$\Rightarrow$: Assume A is universal. Then L_i contains every non-empty word over Σ_i (for $i = 1, 2$). Now every ω-word over $\Sigma_1 \cup \Sigma_2$ is either

(i) entirely over Σ_1 or entirely over Σ_2, or

(ii) alternates between Σ_1 and Σ_2 and then entirely over one of the two

(iii) alternates between Σ_1 and Σ_2 infinitely.

These cases are covered by L', by definition.

$\Leftarrow$: Assume A' is universal. Take $w \in \Sigma^+$. Let w_i be the image of w in Σ_i. Now, by definition of L', $(w_1 \, w_2)^\omega \in (L_1 L_2)^\omega$, because $(w_1 \, w_2)^\omega$ cannot belong to the other five components of L'. Further, by construction of A', this implies that $w_i \in L(A_i)$ (for $i = 1, 2$). Hence $w \in L(A)$ as required. □

In the preceding proof, note that taking L' to be L^ω where $L = L^*(A)$ does not work, because L^ω is universal does not imply that L is universal: just take $L = \{ a, b \}$ over alphabet $\{ a, b \}$.

1.5 Determinisation and McNaughton's Theorem

Other Acceptance Conditions

An *ω-automaton* is a quintuple $A = \langle Q,\ \Sigma,\ q_0 \in Q,\ \Delta \subseteq Q \times \Sigma \times Q,\ \mathit{Acc} \rangle$; the component *Acc* is its acceptance condition.

An ω-automaton is called

- *Büchi*: if *Acc* is of the form $F \subseteq Q$, and a run ρ is accepting just if $\mathsf{inf}(\rho) \cap F \neq \varnothing$.
- *Muller*: if *Acc* is of the form $\mathcal{F} = \{ F_1, \cdots, F_k \}$ with each $F_i \subseteq Q$, and a run ρ is accepting just if $\mathsf{inf}(\rho) \in \mathcal{F}$.
- *Rabin*: if *Acc* is of the form $\{ (E_1, F_1), \cdots, (E_k, F_k) \}$ with $E_i, F_i \subseteq Q$, and a run ρ is accepting just if

$$\exists i \in \{ 1, \cdots, k \} \,.\, \mathsf{inf}(\rho) \cap E_i = \varnothing \ \wedge\ \mathsf{inf}(\rho) \cap F_i \neq \varnothing$$

  I.e. for some i, every state in E_i is visited only finitely often in ρ, but some state in F_i is visited infinitely often in ρ.
- *Streett*: if *Acc* is of the form $\{ (E_1, F_1), \cdots, (E_k, F_k) \}$ with $E_i, F_i \subseteq Q$, and a run ρ is accepting just

$$\forall i \in \{ 1, \cdots, k \} \,.\, \mathsf{inf}(\rho) \cap E_i \neq \varnothing \ \vee\ \mathsf{inf}(\rho) \cap F_i = \varnothing$$

- *Parity*: *Acc* is specified by a *priority function* $\Omega : Q \to \omega$, and a run ρ is accepting just if $\min \Omega(\mathsf{inf}(\rho))$ is even i.e. the least priority that occurs infinitely often is even.

Rabin condition is sometimes called *pairs condition*; Streett condition is sometimes called *complement pairs condition*; parity condition is sometimes called *Mostowski condition.*

It is straightforward to see that parity condition is closed under negation. Given a priority function $\Omega : Q \to \omega$, let ρ be a run that is not parity-accepting. I.e. $\min \Omega(\mathsf{inf}(\rho))$ is odd. Then ρ is parity-accepting w.r.t. the parity function $\Omega' : q \mapsto \Omega(q) + 1$. Note that the Muller condition is also closed under negation. The negation of a Rabin condition is a Streett condition, and vice versa. Given a set of pairs $\{ (E_1, F_1), \cdots, (E_k, F_k) \}$, let ρ be a run that is not Rabin-accepting. I.e. we have $\neg(\exists i.\mathsf{inf}(\rho) \cap E_i = \varnothing \wedge \mathsf{inf}(\rho) \cap F_i \neq \varnothing)$, which is equivalent to $\forall i.(\mathsf{inf}(\rho) \cap E_i \neq \varnothing \ \vee\ \mathsf{inf}(\rho) \cap F_i = \varnothing)$. Thus ρ is Streett-accepting w.r.t. the set of pairs $\{ (F_1, E_1), \cdots, (F_k, E_k) \}$.

Example 1.6 (Muller and Rabin Conditions)**.** Let $L_5 \subseteq \{ a, b, c \}^\omega$ be the language consisting of all words α satisfying: if a occurs infinitely often in α, so does b.

Take the complete state-transition graph with state-set $Q = \{ q_a, q_b, q_c \}$. The idea is that when the automaton reaches state q_a, it has just read a; similarly for q_b and q_c. The Muller and Rabin conditions for a 3-state automaton that recognises L_4 are as follows.

- Muller: Take $\mathcal{F} = \{ U \subseteq Q \mid q_a \in U \Rightarrow q_b \in U \}$.
- Rabin: Take $\Omega = \{ (\{ q_a \}, \{ q_b, q_c \}), (\varnothing, \{ q_b \}) \}$. Observe that $\alpha \in L_5$ iff a occurs only finitely often in α or b occurs infinitely often in α.

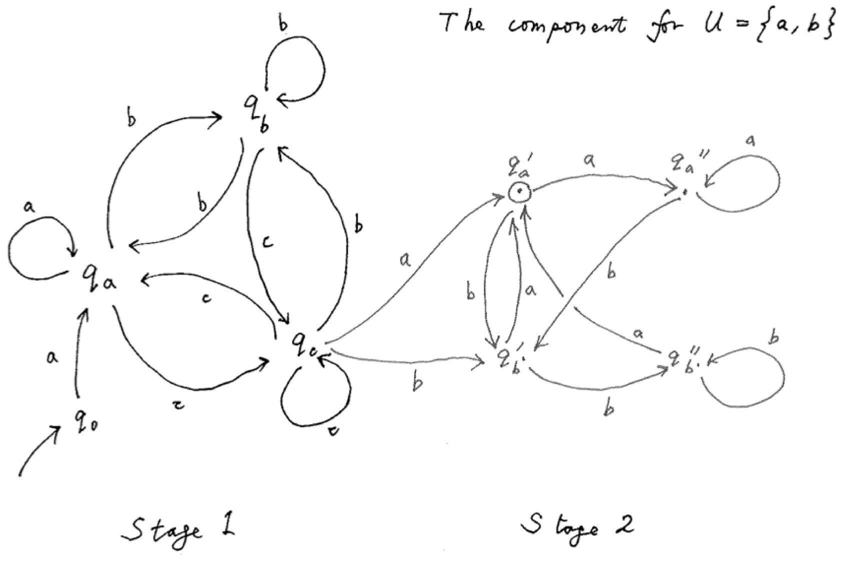


Figure 1.3: Transforming Muller to Büchi

Example 1.7 (From Muller to Büchi)**.** Let $A_M = (\{ q_a, q_b, q_c \}, \Sigma, q_0, \mathcal{F})$ be a Muller automaton recognising L_5. We can construct an equivalent Büchi automaton as follows.

Stage 1. Simulate a run ρ of A_M. Guess $\mathsf{inf}(\rho) = U$, for some $U \in \mathcal{F}$. At some point, guess that all states not in U have just been seen.

Stage 2. Check that henceforth:

(i) Every state reached is in U.

(ii) Every state in U is read infinitely often.

See Figure 1.3 for an illustration.

McNaughton's Theorem

Theorem 1.4 (McNaughton 1966)**.** *The following are equivalent:*

(i) non-deterministic Büchi automata (NB)

(ii) deterministic Rabin automata (DR)

(iii) non-deterministic Rabin automata (NR)

(iv) deterministic Muller automata (DM)

(v) non-deterministic Muller automata (NM)

$$\begin{tikzcd}
 & NM \arrow[ddd, bend left=60] & \\
NR \arrow[ur] & & DM \arrow[ul] \\
 & DR \arrow[ul] \arrow[ur] & \\
 & NB \arrow[u] &
\end{tikzcd}$$

Proof. We write $\Rightarrow$ to mean "can be simulated by".

- "DR $\Rightarrow$ NR" and "DM $\Rightarrow$ NM" are immediate.
- "NB $\Rightarrow$ NR" and "DB $\Rightarrow$ DR": Büchi conditions are instances of Rabin: Given $F \subseteq Q$, the corresponding Rabin condition is $\{ (\varnothing, F) \}$.

- "DR $\Rightarrow$ DM" and "NR $\Rightarrow$ NM": Given Rabin $\{(E_1, F_1), \cdots, (E_n, F_n)\}$. Set Muller

$$\mathcal{F} \ := \ \{ U \subseteq Q \mid \bigvee_{i=1}^{n} (U \cap E_i = \varnothing) \wedge (U \cap F_i \neq \varnothing) \}$$

- "NM $\Rightarrow$ NB": (informal)

  Stage 1 Simulate a run ρ of the given Muller automaton. Guess $\mathsf{inf}(\rho) = U$, some $U \in \mathcal{F}$. At some point, guess that all states of ρ that are not in U have just been seen.

  Stage 2 Check that henceforth:

  (i) Every state reached is in U.

  (ii) Every state in U is read infinitely often.

- "NB $\Rightarrow$ DR" is the most difficult: it was first shown by McNaughton (1966), using a double exponential construction.

□

Note that McNaughton's theorem provides an alternative route to complementing an ω-regular language, since it is easy to complement a Muller automaton. Given a Büchi-recognisable language $L(B)$, one could first transform the Büchi automaton B into an equivalent deterministic Muller automaton $M = (Q, \Sigma, \Delta, q_0, \mathcal{F})$. Then the Muller automaton $(Q, \Sigma, \Delta, q_0, \mathcal{P}(Q) \setminus \mathcal{F})$ recognises $\Sigma^\omega \setminus L(B)$.

NB $\Rightarrow$ DR: Safra's Construction

In order to determinise a non-deterministic automaton, we must find a finite data structure that can approximate the set of states across possible runs of the non-deterministic automaton, and can distinguish between accepted and rejected ω-words.

For automata over finite words, the *power set* data structure is sufficient. Given a non-deterministic automaton with state set Q, the deterministic automaton uses states $S \in \mathcal{P}(Q)$. Such a state S stores the set of states from Q that are reachable on a given input word, and the deterministic automaton accepts if it ends in a state S such that $S \cap F \neq \varnothing$.

Unfortunately, for non-deterministic Büchi automata, the same construction does not always distinguish between accepted and rejected ω-words. In fact, even with the more powerful Muller or Rabin acceptance condition, the powerset data structure is not sufficient (Michel, 1988; Löding, 1999).

The solution is to use a more sophisticated data structure called *Safra trees* (Safra, 1988). In these notes, we describe *history trees*, a variant of Safra trees introduced by Schewe (2009).

As a running example in this section, we will use the non-deterministic Büchi automaton in Figure 1.4 (for which there is no deterministic Büchi automaton recognizing the same language).

History trees

A Σ'-*labelled tree* t is a partial function $t : \mathbb{N}^* \to \Sigma'$ mapping a position/node $x \in \mathbb{N}^*$ to its label $t(x) \in \Sigma'$ such that $\mathsf{dom}(t)$ is prefix closed (i.e. t truly has a tree structure).

We say t is

- *finite* if $\mathsf{dom}(t)$ is finite;

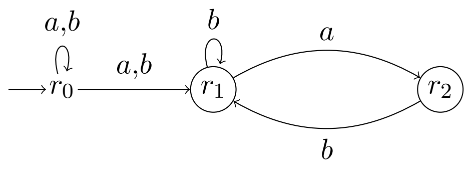


Figure 1.4: Non-deterministic Büchi automaton recognizing the language consisting of ω-words $u \in \{a, b\}^\omega$ where there are only finitely many occurrences of the string aa.

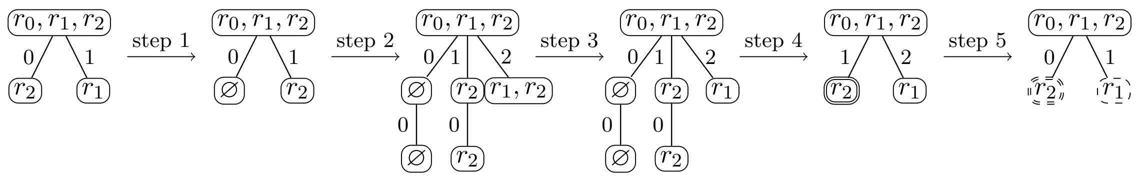


Figure 1.5: Updating history tree on input letter a for automaton in Figure 1.4.

- *ordered* if $xd \in \mathsf{dom}(t)$ implies that $xd' \in \mathsf{dom}(t)$ for all $d' \leq d$.

A *history tree* for a non-deterministic Büchi automaton $(Q, \Sigma, q_0, \delta, F)$ is a $(\mathcal{P}(Q) \setminus \varnothing)$-labelled ordered finite tree such that

- the label of each node is a *proper superset* of its children, and
- the labels of the children of each node are *disjoint*.

Note that because of these restrictions, there are only finitely many history trees for a given non-deterministic Büchi automaton.

An *enriched history tree* has additional labels to indicate whether each node is stable or if it is a breakpoint (to be defined below).

Updating history trees

Given history tree t for $A = (Q, \Sigma, q_0, \delta, F)$ and $\sigma \in \Sigma$, construct a new enriched history tree t' as follows:

1. **Update:** for all $x \in \mathsf{dom}(t)$, let $t'(x) = \bigcup_{q \in t(x)} \delta(q, \sigma)$;
2. **Create:** for all $x \in \mathsf{dom}(t')$, create new youngest child xd of x with $t'(xd) = t'(x) \cap F$;
3. **Horizontal merge:** for all $x \in \mathsf{dom}(t')$ and all $q \in Q$, if q occurs in an older sibling of x then remove q from $t'(x)$ and all of its descendants;
4. **Vertical merge:** for all $x \in \mathsf{dom}(t')$,
   - if $t'(x) = \varnothing$, then remove node x,
   - if $t'(x) = \bigcup_{d \in \mathbb{N}} t'(xd)$ then remove nodes xd (call this a *breakpoint for x*);
5. **Repair order:** rename to restore order (call nodes that are not renamed *stable*);

In these notes, $\boxed{}$ represent breakpoint positions, and ⬚ represent unstable positions.

Figure 1.5 provides an example of a history tree update for the automaton in Figure 1.4. Please refer to Schewe (2009) for more description of these steps, and a detailed example of a history tree update.

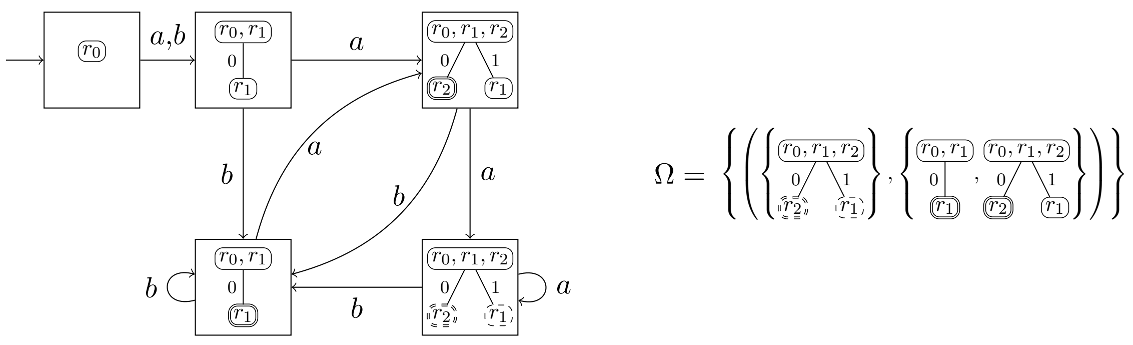


Figure 1.6: Deterministic Rabin automaton obtained using the construction in Schewe (2009) on the non-deterministic Büchi automaton in Figure 1.4.

Constructing equivalent deterministic Rabin automaton

Let $A = (\Sigma, Q, q_0, \delta, F)$ be a non-deterministic Büchi automaton. We construct a deterministic Rabin automaton $D = (\Sigma, S, s_0, \delta_D, \Omega)$ such that

- S is the finite set of enriched history trees for A,
- s_0 is the history tree such that $\mathsf{dom}(s_0) = \{\epsilon\}$ and $s_0(\epsilon) = \{q_0\}$,
- δ_D removes any markers for stable nodes and breakpoints, and then updates the history tree as described above,
- the Rabin acceptance condition is given by $\Omega = \{(E_x, F_x) : x \in J\}$ where
  - E_x is the set of enriched history trees where x is unstable or not in the domain,
  - F_x is the set of enriched history trees where there is a breakpoint for x,
  - J is the set of positions for history trees in S.

  In other words, the acceptance condition says:

  there is some position x that is *eventually always stable* and *always eventually a breakpoint.*

Figure 1.6 shows the deterministic Rabin automaton obtained using this construction applied to the non-deterministic Büchi automaton in Figure 1.4.

We now sketch the proof that D recognises the same language as A.

Proposition 1.6. *$L(D) = L(A)$.*

Proof. $L(A) \subseteq L(D)$: Fix an accepting run $\rho_A = q_0 q_1 \ldots$ of A on some $u \in L(A)$. Let ρ_D be the run of D on u.

Since ρ_A is infinite, the root is always stable. If the root has infinitely many breakpoints, then ρ_D is accepting.

Otherwise, consider the first position i_1 after the final breakpoint for the root where $q_{i_1} \in F$. For $j \geq i_1$, the states q_j must always be in the label of one of the children of the root (since no vertical merges to the root are possible after i_1). Horizontal merges can only transfer states to older siblings in the tree, so eventually the states in ρ_A are in a subtree of a stable child x_1 of the root. If x_1 has infinitely many breakpoints, then ρ_D is accepting.

Otherwise, consider the first position i_2 after the final breakpoint for x_1 where $q_{i_2} \in F$. As before, eventually the states in ρ_A are in a subtree of a stable child x_2 of x_1. If x_2 has infinitely many breakpoints, then ρ_D is accepting. Otherwise...

Since history trees have depth at most $|Q|$ and $x_i < x_{i+1}$, if we continue to reason like this, then we must eventually reach a depth d such that x_d is stable and has infinitely many breakpoints, so $u \in L(D)$.

$L(D) \subseteq L(A)$: Fix $u \in L(D)$ and let ρ_D be the accepting run of D on u. Then there is $x \in J$ such that x is eventually always stable, and has infinitely many breakpoints.

Let R_i be the label at position x for the i-th breakpoint at x (after x is stable), and let u_i be the infix of u read between the i-th breakpoint and $(i+1)$-st breakpoint. Let u_0 be the prefix of u read before R_1.

For all $r \in R_1$, there is a partial run of A on u_0 that ends in state r.

Likewise, for all $r \in R_i$, there is a state $q \in R_{i-1}$ such that there is a partial run of A on u_i starting in q, ending in r, and passing through at least one state in F.

Arrange these runs in a Q-labelled tree. By the observation above, there are infinitely many positions in this tree. We can now take advantage of König's lemma.

Lemma 1.3 (König's lemma)**.** *A finitely branching infinite tree contains an infinite path.*

The infinite branch guaranteed by König's lemma can be used to construct a run of A on u that visits F infinitely often, so $u \in L(A)$. $\square$

Complexity of determinisation

Given a NB with n states:

1. Safra (1988) constructed an equivalent DR with at most $(12)^n \, n^{2n}$ states and $2n$ pairs in the acceptance condition.
2. Piterman (2007) modified Safra trees to construct equivalent deterministic parity automata of size at most $2n \, n^n \, n!$.
3. By a finer analysis of Piterman, Liu and Wang (2009) obtained an upper bound of $2n \, (n!)^2$.
4. Schewe (2009) gave a NB to DR construction with state complexity $o((2.66\, n)^n)$, but requiring 2^{n-1} Rabin pairs.
5. Colcombet and Zdanowski (2009) proved that Schewe's construction is optimal for state complexity.

Problems

1.1 Let Σ be a finite alphabet. Prove that every $w \in \Sigma^\omega$ can be factorised as $w = u\,v$ where $u \in \Sigma^*$ and $v \in \Sigma^\omega$ and each letter in v occurs infinitely often in w.

1.2 Construct Büchi automata that recognise the following ω-languages over $\Sigma = \{\, a, b, c \,\}$:

(a) The set of words in which after each a, there is a b.

(b) The set of words in which a appears only at odd, or only at even positions.

1.3 Construct Büchi automata that recognise the following ω-languages over $\Sigma = \{\, a, b, c \,\}$:

(a) The set of ω-words in which abc appears as a segment at least once.

(b) The set of ω-words in which abc appears as a segment infinitely often.

(c) The set of ω-words in which abc appears as a segment only finitely often.

1.4 Prove that every nonempty Büchi-recognisable language contains an *ultimately periodic* word (i.e. an infinite word of the form $u\,v^\omega$ for finite words u and v).

1.5 Prove or disprove the following: for $U, V \subseteq \Sigma^+$

(a) $(U \cup V)^\omega = U^\omega \cup V^\omega$

(b) $\lim(U \cup V) = \lim U \cup \lim V$

(c) $U^\omega = \lim U^+$

(d) $\lim(U \cdot V) = U \cdot V^\omega$.

(e) $(U + V)^\omega \equiv (U^* V)^\omega + (U + V)^* U^\omega$

(f) $(UV)^\omega \equiv U(VU)^\omega$

(g) For all $n > 0$, $(U^n)^\omega \equiv (U^+)^\omega \equiv U^\omega$

(h) $U^\omega \equiv U^+ U^\omega$.

1.6 Prove that the ω-language $L = \{\, u^\omega : u \in \{\, 0, 1 \,\}^+ \,\}$ is not recognised by any Büchi automaton.

[*Hint.* Consider the word $(01^n)^\omega$ where n is a number greater than the number of states of A.]

1.7 Prove the following (from first principles):

(a) If $U \subseteq \Sigma^*$ is regular then U^ω is Büchi-recognisable.

(b) If $U \subseteq \Sigma^*$ is regular and $L \subseteq \Sigma^\omega$ is Büchi-recognisable then $U \cdot L$ is Büchi-recognisable.

1.8 Prove the following.

(i) $\sim_A$ saturates L

(ii) $\sim_A$ saturates $\Sigma^\omega \setminus L$.

1.9 Prove that an ω-language is deterministic Büchi-recognisable iff it is of the form $\lim U$ for some regular U.

1.10 (***Hard***) A quasi order (i.e. reflexive and transitive binary relation) $\lesssim$ over a set X is called a *well quasi ordering* (w.q.o.) if every infinite sequence $a_1, a_2, \cdots$ from X is saturated, meaning that there exist $i < j$ such that $a_i \lesssim a_j$.

Let Σ be a finite alphabet. The *subword ordering* $\lesssim \subseteq \Sigma^* \times \Sigma^*$ is defined as: $u_1 \cdots u_m \lesssim v_1 \cdots v_n$ just if there exist $1 \leq i_1 < i_2 < \cdots < i_m \leq n$ such that for each $1 \leq j \leq m$, $u_j = v_{i_j}$. Prove that $(\Sigma^*, \lesssim)$ is a w.q.o.

[*Hint.* Suppose, for a contradiction, there is an infinite sequence of words $w_1, w_2, \cdots$ that is unsaturated. For an appropriate notion of "minimal", choose a minimal such sequence. Then consider the derived sequence $v_1, v_2, \cdots$ whereby $w_i = a_i v_i$ and $a_i \in \Sigma$, for each i.]

1.11 Consider the ω-language

$$L \;:=\; \{\, \alpha \in \{\,0, 1\,\}^\omega \mid \alpha \text{ contains } 00 \text{ infinitely often, but } 11 \text{ only finitely often} \,\}.$$

(a) Construct a Büchi automaton that recognises L. Explain why it works.

(b) Show that L is not recognisable by a deterministic Büchi automaton.

(c) We say that a ω-automaton *co-Büchi recognises* an ω-word α if there is a run ρ of the automaton on α such that from some point onwards, only final states will be visited i.e. there is an $n \geq 0$ such that for every $i > n$, $\rho(i)$ is a final state.

Is L recognisable by a deterministic co-Büchi automaton? Justify your answer.

1.12

(a) Let L be an ω-language over the alphabet Σ. Define *right-congruence* $\sim_L \subseteq \Sigma^* \times \Sigma^*$ by

$$u \sim_L v \;:=\; \forall \alpha \in \Sigma^\omega . u\,\alpha \in L \leftrightarrow v\,\alpha \in L.$$

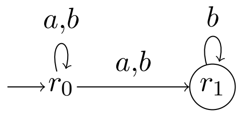
Prove that every deterministic Muller automaton that recognises L needs at least as many states as there are $\sim_L$-equivalence classes.

Show that there is a ω-language L, which is not ω-regular, such that $\sim_L$ has only finitely many equivalence classes.

Hence, or otherwise, state (without proof) a result about regular $*$-languages (i.e. sets of finite words) that does not generalise to ω-regular ω-languages.

(b) Is it true that an ω-language is ω-regular if and only if it is expressible as a Boolean combination of languages of the form $\lim U$ where U is a regular $*$-language? Justify your answer.

1.13 Apply Safra's construction (as described in Section 1.5) to obtain a deterministic Rabin automaton that is equivalent to the following non-deterministic Büchi automaton:



Chapter 2

Linear-time Temporal Logic

Synopsis[1]

Kripke structures. Examples of correctness properties of reactive systems. LTL: syntax and semantics. Transformation of LTL formulas to generalised Büchi automata. LTL model checking is PSPACE-complete: Savitch's algorithm; encoding polynomial-space Turing machines in LTL. Expressivity of LTL: Kamp's theorem.

2.1 Motivating Example: Mutual Exclusion Protocol

The model checking problem: Given a system *Sys* and a specification *Spec* on the runs of the system, does *Sys* satisfy *Spec*?

Example 2.1 (Mutual exclusion protocol)**.** A MUX protocol is modelled by a transition system over state-space $\mathbb{B}^5$:

```
Process 0:  Repeat
00:  <non-critical region 0>
01:  wait unless turn = 0
10:  <critical region 0>
11:  turn := 1

Process 1:  Repeat
00:  <non-critical region 1>
01:  wait unless turn = 1
10:  <critical region 1>
11:  turn := 0
```

A *state* is a bit-vector "$a_1\, a_2\, b_1\, b_2\, t$" where $a_1\, a_2$ are $b_1\, b_2$ are line no. of processes 0 and 1 respectively, and t is the value of shared variable `turn`; the initial state is 00000. Some examples of correctness properties *Spec*:

(i) *Safety*: The state $1010t$ is never reached.

[1]The contributions of past guest lecturers, Matthew Hague and Anthony Lin, are gratefully acknowledged.

(ii) *Liveness*: It is always the case that whenever $01b_1b_2t$ is reached, $10b_1'b_2't'$ is eventually reached (similarly for a_1a_201t and $a_1'a_2'10t'$).

Temporal Logic in Computer Science

Amir Pnueli (1941–2009) won the *ACM Turing Award 1996*

"For seminal work introducing temporal logic into computing science and for outstanding contributions to program and system verification."

A landmark publication is (Pnueli, 1977).

2.2 Kripke Structures

Fix a set $\{ p_1, \cdots, p_n \}$ of atomic propositions. We use Kripke structures $\mathcal{K}$ to model reactive systems.

Definition 2.1. A *Kripke structure* over a fixed set of atomic propositions $\{ p_1, \cdots, p_n \}$ is a quadruple (S, R, λ, s_0) with

- a finite state-set S, and $s_0 \in S$ is the initial state
- a transition relation $R \subseteq S \times S$, and
- a labelling function $\lambda : S \to \mathcal{P}(\{ p_1, \cdots, p_n \})$, associating with each $s \in S$ the set of those p_i that are satisfied at s.

A Kripke structure is just a directed graph whose nodes are labelled by elements of the power set, $\mathcal{P}(\{ p_1, \cdots, p_n \})$, as given by λ.

Notation We often write $\lambda(s)$ as a bit vector $\begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix} \in \mathbb{B}^n$ such that $b_i = 1$ iff $p_i \in \lambda(s)$.

A *path* through a Kripke structure (S, R, λ, s_0) is an infinite sequence of states, $s_0 s_1 s_2 \cdots$, where for each $i \geq 0$, $(s_i, s_{i+1}) \in R$. The corresponding *label sequence* is the ω-word over the alphabet $\mathbb{B}^n$: $\lambda(s_0)\,\lambda(s_1)\,\lambda(s_2) \cdots$.

Example 2.2. Fix atomic propositions p_1 and p_2.

$$\begin{tikzcd}
 & & \binom{1}{0} \arrow[r] \arrow[dd, shift left] & \binom{0}{0} \arrow[loop right] \\
{} \arrow[r] & \binom{1}{1} \arrow[ur] \arrow[dr] & & \\
 & & \binom{0}{1} \arrow[uu, shift left] &
\end{tikzcd}$$

Example label sequences:

(i) $\binom{1}{1}\binom{1}{0}\binom{0}{1}\binom{1}{0}\binom{0}{0}\binom{0}{0}\cdots$

(ii) $\binom{1}{1}\binom{0}{1}\binom{1}{0}\binom{0}{1}\binom{1}{0}\binom{0}{0}\binom{0}{0}\cdots$

What are the differences between Büchi automata and Kripke structures?

Correctness Properties of Reactive Systems: Examples

When a reactive system is modelled as a Kripke structure, runs of the system correspond to label sequences of $\mathcal{K}$, which are ω-words over $(\mathbb{B}^n)^\omega$. Correctness properties of the reactive system are thus naturally expressed as properties of ω-words. In other words, they are path properties.

The model checking problem asks: given a correctness property φ expressed as a property of ω-words, does every label sequence of $\mathcal{K}$ satisfy φ?

Example 2.3 (MUX protocol revisited)**.** For $i = 0, 1$, let

- p_{i+1} stand for "Process i is waiting (to enter the critical region)"
- p_{i+3} stand for "Process i is in critical region"

Consider the following φ:

(i) "It is always the case that when p_1 holds then sometime later p_3 holds" which means: for any label sequence, when letter $(1, b_2, b_3, b_4)$ occurs, subsequently a letter $(b'_1, b'_2, 1, b'_4)$ occurs.

(ii) "p_3 and p_4 never hold simultaneously" which means: no label sequence contains the letter $(b_1, b_2, 1, 1)$.

Example 2.4 (Sequence properties)**.** Fix state properties p_1 and p_2. Label sequences are ω-words over $\mathbb{B}^2 = \{ \binom{0}{0}, \binom{0}{1}, \binom{1}{0}, \binom{1}{1} \}$.

(i) *Recurrence*: "p_1 holds again and again (i.e. infinitely often)."

(ii) *Periodicity*: "p_1 is true initially and precisely at every third moment."

(iii) *Request-response*: "It is *always* the case that whenever p_1 holds, p_2 will hold sometime later."

(iv) *Obligation*: "p_1 eventually holds but p_2 never does."

(v) *Until condition*: "It is always the case that when p_1 holds, sometime later p_1 will be true again, and in the meantime p_2 is always true."

(vi) *Fairness*: "If p_1 is true again and again (infinitely often), then the same is true of p_2".

2.3 Syntax and Semantics

In our present modelling framework, correctness properties are path properties. We present Linear-time Temporal Logic, a logical system for expressing properties of ω-words.

LTL-formulas, over atomic propositions $p_1, \cdots, p_n$, are defined by the grammar:

$$
\begin{array}{rcll}
\varphi & ::= & p_i & \text{atomic proposition} \\
& | & \neg\varphi & \text{negation} \\
& | & \varphi \wedge \psi & \text{conjunction} \\
& | & \varphi \vee \psi & \text{disjunction} \\
& | & \boldsymbol{X}\,\varphi & \textit{next} \\
& | & \varphi\, \boldsymbol{U}\, \psi & \textit{until}
\end{array}
$$

Intuitively

$$
\begin{array}{ll}
\boldsymbol{X}\,\varphi & \text{“}\varphi \text{ is true at the } \textit{next} \text{ time-step”} \\
\varphi\, \boldsymbol{U}\, \psi & \text{“}\varphi \text{ is true } \textit{until } \psi \text{ is true (and } \psi \text{ holds eventually)”}
\end{array}
$$

(Picture of time-line)

Two additional constructs

$\boldsymbol{F}\varphi$ “φ is *eventually* true”
i.e. φ is true at *some* point in the future (starting from the present)

$\boldsymbol{G}\varphi$ “φ is *always* true”
i.e. φ is true at *every* point in the future (including the present)

They are expressible in LTL by

$$
\begin{array}{rcl}
\boldsymbol{F}\varphi & := & \mathsf{true}\ \boldsymbol{U}\,\varphi \\
\boldsymbol{G}\varphi & := & \neg(\boldsymbol{F}\neg\varphi)
\end{array}
$$

(Henceforth we regard the above as definitions.)

LTL-formulas over atomic propositions $p_1, \cdots, p_n$ are interpreted as sets of ω-words α over the alphabet $\mathbb{B}^n$.

Notation Let $\alpha = \alpha(0)\ \alpha(1)\ \alpha(2) \cdots \in (\mathbb{B}^n)^\omega$:

- α^i stands for $\alpha(i)\ \alpha(i+1)\ \alpha(i+2) \cdots$, so $\alpha = \alpha^0$.
- $(\alpha(i))_j$ is the j-th component of the vector $\alpha(i)$.

Definition 2.2 (Satisfaction)**.** Let $i \geq 0$. Define $\alpha^i \vDash \varphi$ by recursion over the syntax of φ:

$$
\begin{array}{rcl}
\alpha^i \vDash p_j & := & (\alpha(i))_j = 1 \\
\alpha^i \vDash \neg\varphi & := & \neg(\alpha^i \vDash \varphi) \\
\alpha^i \vDash \varphi \vee \psi & := & \alpha^i \vDash \varphi\ \vee\ \alpha^i \vDash \psi \\
\alpha^i \vDash \varphi \wedge \psi & := & \alpha^i \vDash \varphi\ \wedge\ \alpha^i \vDash \psi \\
\alpha^i \vDash \boldsymbol{X}\varphi & := & \alpha^{i+1} \vDash \varphi \\
\alpha^i \vDash \varphi\,\boldsymbol{U}\,\psi & := & \exists j \geq i : \big(\alpha^j \vDash \psi\ \wedge\ \forall i \leq k \leq j-1 : \alpha^k \vDash \varphi\big)
\end{array}
$$

We say that $\alpha \vDash \varphi$, read α *satisfies* φ, just if $\alpha^0 \vDash \varphi$.

Examples of LTL-definable Correctness Properties

Example 2.5 (Sequence properties revisited). (i) *Recurrence*: p_1 holds again and again (i.e. infinitely often).

$$\boldsymbol{G}\,(\boldsymbol{F} p_1)$$

(ii) *Periodicity*: p_1 is true initially and precisely at every third moment.

$$p_1 \ \wedge\ \boldsymbol{X}\neg p_1 \ \wedge\ \boldsymbol{X}\boldsymbol{X}\neg p_1 \ \wedge\ \boldsymbol{G}\,(p_1 \leftrightarrow \boldsymbol{X}\boldsymbol{X}\boldsymbol{X} p_1)$$

(iii) *Request-response*: It is *always* the case that whenever p_1 holds, p_2 will hold sometime later.

$$\boldsymbol{G}\,(p_1 \rightarrow \boldsymbol{X}\boldsymbol{F} p_2)$$

(iv) *Obligation*: Eventually p_1 holds but p_2 never does.

$$\boldsymbol{F} p_1 \wedge \neg \boldsymbol{F} p_2$$

(v) *Until condition*: It is always the case that when p_1 holds, sometime later p_1 will hold again, and in the meantime p_2 is always true.

$$\boldsymbol{G}\,(p_1 \rightarrow \boldsymbol{X}\,(p_2\ \boldsymbol{U} p_1))$$

(vi) *Fairness*: If p_1 is true again and again, then the same is true of p_2.

$$\boldsymbol{G}\boldsymbol{F} p_1 \rightarrow \boldsymbol{G}\boldsymbol{F} p_2$$

Exercise 2.1. Verify the following:

(i) $\alpha^i \vDash \boldsymbol{F}\varphi \ \leftrightarrow\ \exists j \geq i : \alpha^j \vDash \varphi$

(ii) $\alpha^i \vDash \boldsymbol{G}\varphi \ \leftrightarrow\ \forall j \geq i : \alpha^j \vDash \varphi$

Definition 2.3. (i) An ω-language $L \subseteq (\mathbb{B}^n)^\omega$ is *LTL-definable* just if there is an LTL-formula φ over $p_1, \cdots, p_n$ such that $L = \{\, \alpha \in (\mathbb{B}^n)^\omega \mid \alpha \vDash \varphi \,\}$. We say that L is definable by φ.

(ii) We say that two LTL-formulas φ and ψ are *equivalent*, written $\varphi \equiv \psi$, if they define the same ω-language.

(iii) A Kripke structure $\mathcal{K} = (S, R, \lambda, s_0)$ *satisfies* an LTL-formula φ, written $\mathcal{K} \vDash \varphi$, just if every label sequence of $\mathcal{K}$ satisfies φ.

Translating LTL formulas into Büchi automata We consider the translation of LTL formulas into equivalent Büchi automata by examples.

Example 2.6.

$\alpha \vDash \boldsymbol{F}(p_1 \wedge \boldsymbol{X}(\neg p_2\ \boldsymbol{U} p_1))$

iff for some $j \geq 0 : \alpha^j \vDash p_1$ and $\alpha^{j+1} \vDash \neg p_2\ \boldsymbol{U} p_1$

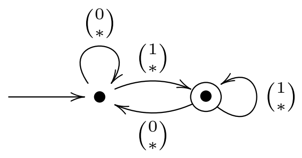
iff for some $j \geq 0 : \alpha^j \vDash p_1$ and for some $j' \geq j+1$: $\alpha^{j'} \vDash p_1$ and for all $j+1 \leq k \leq j'-1 : \alpha^k \vDash \neg p_2$

iff for some j and some $j' > j : \alpha(j)$ and $\alpha(j')$ have 1 in the 1st component, and for all $j < k < j'$, $\alpha(k)$ has 0 in 2nd component

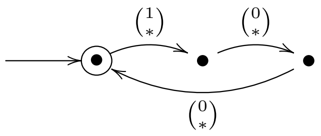
iff α has two occurrences of $\binom{1}{*}$ between which only letters of the form $\binom{*}{0}$ occur.

Exercise Draw a Büchi automaton that recognises the same ω-language.

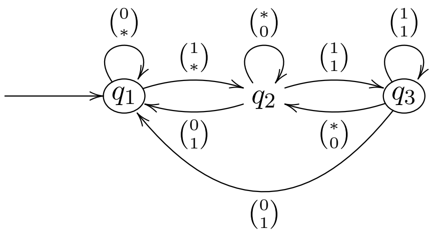
Example 2.7. (i) $\boldsymbol{G}(\boldsymbol{F}p_1)$



(ii) $p_1 \wedge \boldsymbol{X}\neg p_1 \wedge \boldsymbol{X}\boldsymbol{X}\neg p_1 \wedge \boldsymbol{G}(p_1 \leftrightarrow \boldsymbol{X}\boldsymbol{X}\boldsymbol{X}p_1)$



(iii) $\boldsymbol{G}(p_1 \to \boldsymbol{X}\boldsymbol{F}p_2)$



At state q_1, the automaton has no obligation to read a $\binom{*}{1}$; q_2 means that it is obliged to, but has not yet, read a $\binom{*}{1}$ since the last $\binom{1}{*}$; q_3 is reached after reading $\binom{1}{1}$.

Exercise 2.2. Translate the following to Büchi automata:

(i) $(\boldsymbol{F}p_1) \wedge \neg\boldsymbol{F}p_2$

(ii) $\boldsymbol{G}(p_1 \to \boldsymbol{X}(p_2\ \boldsymbol{U}\ p_1))$

2.4 Translating LTL to Generalised Büchi Automata

We can systematically translate a given LTL formula to an equivalent (generalised) Büchi automaton. In fact, we shall utilise such an automaton construction to design a decision procedure for the LTL model checking problem.

Definition 2.4. A *generalised Büchi automaton* (GBA) is a 5-tuple

$$(Q, \Sigma, \Delta, q_0, \{ F_0, \cdots, F_{l-1} \})$$

with final state-sets $F_0, \cdots, F_{l-1} \subseteq Q$. A run ρ is *accepting* just if for each i, there is some state in F_i which occurs infinitely often in ρ i.e. $\bigwedge_i(\mathsf{inf}(\rho) \cap F_i \neq \varnothing)$.

Proposition 1. *Given a generalised Büchi automaton* $A = (Q, \Sigma, \Delta, q_0, \{ F_0, \cdots, F_{l-1} \})$, *define Büchi automaton*

$$A' = (Q \times \{ 0, 1, \cdots, l-1 \}, \Sigma, \Delta', (q_0, 0), F_0 \times \{ 0 \})$$

with Δ' *consisting of*

- $((p, i), a, (q, i))$ *if* $p \notin F_i$
- $((p, i), a, (q, (i + 1) \bmod l))$ *if* $p \in F_i$

assuming that $(p, a, q) \in \Delta$. *Prove that* A *and* A' *recognise the same* ω*-language over* Σ.

Exercise 2.3. *Prove the proposition.*

The rest of the section is concerned with the proof of the following theorem.

Theorem 2.1 (Translating LTL to GBA)**.** *Let* φ *be an LTL formula over* $p_1, \cdots, p_n$. *Suppose* m *is the number of distinct non-atomic subformulas of* φ. *There is a generalised Büchi automaton* A_φ *with state-set* $\{ q_0 \} \cup \mathbb{B}^{n+m}$ *that is equivalent to* φ *i.e. the language definable by* φ *coincides with the language recognised by* A_φ. *Further the translation* $\varphi \mapsto A_\varphi$ *is effective.*

Evaluating LTL-formula φ over $\alpha \in (\mathbb{B}^n)^\omega$ Given ω-word α over $(\mathbb{B}^n)^\omega$, and LTL-formula φ over $p_1, \cdots, p_n$. Define formulas $\varphi_1, \varphi_2, \cdots, \varphi_{n+m}$ where

- $\varphi_1 = p_1, \cdots, \varphi_n = p_n$, and
- $\varphi_{n+1}, \cdots, \varphi_{n+m} = \varphi$ are all the distinct *non-atomic* subformulas of φ, listed in non-decreasing order of size.

We construct a two-dimensional semi-infinite array of truth values, $\beta \in (\mathbb{B}^{n+m})^\omega$, defined by: $(\beta(i))_j = 1$ (i.e. j-th row, i-th column is 1) if and only if $\alpha^i \vDash \varphi_j$. In particular $\alpha \vDash \varphi \leftrightarrow (\beta(0))_{m+n} = 1$. We call $\beta \in (\mathbb{B}^{n+m})^\omega$ the φ*-expansion of* α.

Example 2.8. Take $\varphi = \boldsymbol{F}(\neg p_1 \wedge \boldsymbol{X}(\neg p_2\, \boldsymbol{U} p_1))$ and $\alpha = \binom{1}{0} \binom{0}{1} \binom{1}{1} \binom{0}{0} \binom{1}{0} \binom{0}{1} \cdots$. We construct $\beta \in (\mathbb{B}^{2+6})^\omega$, the φ-expansion of α:

$\varphi_1 = p_1$	1	0	1	0	1	0	$\cdots$
$\varphi_2 = p_2$	0	1	1	0	0	1	
$\varphi_3 = \neg p_1$	0	1	0	1	0	1	$\cdots$
$\varphi_4 = \neg p_2$	1	0	0	1	1	0	$\cdots$
$\varphi_5 = \neg p_2\, \boldsymbol{U} p_1$	1	0	1	1	1	0	$\cdots$
$\varphi_6 = \boldsymbol{X}(\neg p_2\, \boldsymbol{U} p_1)$	0	1	1	1	0	·	$\cdots$
$\varphi_7 = \neg p_1 \wedge \boldsymbol{X}(\neg p_2\, \boldsymbol{U} p_1)$	0	1	0	1	0	·	$\cdots$
$\varphi_8 = \underbrace{\boldsymbol{F}(\neg p_1 \wedge \boldsymbol{X}(\neg p_2\, \boldsymbol{U} p_1))}_{\varphi}$	1	1	1	1	·	·	$\cdots$

Note that the 3rd (resp. 4th) row is the negation of the 1st (resp. 2nd) row.

Finite characterisation of the φ-expansion of an ω-word α The semantics of an LTL formula φ over an ω-word α is captured by the φ*-expansion of* α, which is an infinite object. We first characterise φ-expansions by a finite set of compatibility conditions, and then construct a generalised Büchi automaton that *guesses* the φ-expansion of α, as α is read. We divide these rules into local and global as follows.

Local compatibility conditions: *Local* in the sense that the conditions relate contiguous letters (*qua* column vectors) of $\beta \in (\mathbb{B}^{m+n})^\omega$.

Cases	*Local conditions*
$\varphi_j = \neg(\varphi_k)$	$(\beta(i))_j = 1 \leftrightarrow (\beta(i))_k = 0$
$\varphi_j = \varphi_k \wedge \varphi_l$	$(\beta(i))_j = 1 \leftrightarrow [(\beta(i))_k = 1$ and $(\beta(i))_l = 1]$
$\varphi_j = \varphi_k \vee \varphi_l$	$(\beta(i))_j = 1 \leftrightarrow [(\beta(i))_k = 1$ or $(\beta(i))_l = 1]$
$\varphi_j = \boldsymbol{X}\,\varphi_k$	$(\beta(i))_j = 1 \leftrightarrow (\beta(i+1))_k = 1$
$\varphi_j = \varphi_k\ \boldsymbol{U}\,\varphi_l$	$(\beta(i))_j = 1 \leftrightarrow (\beta(i))_l = 1$ or $[(\beta(i))_k = 1$ and $(\beta(i+1))_j = 1]$

The last clause can be explained by the equivalence: $\varphi\ \boldsymbol{U}\,\psi \equiv \psi \vee (\varphi \wedge \boldsymbol{X}\,(\varphi\ \boldsymbol{U}\,\psi))$.

Global compatibility condition $\varphi_j = \varphi_k\ \boldsymbol{U}\,\varphi_l$: *There is no m such that for all $n \geq m$, we have $(\beta(n))_j = 1$ and $(\beta(n))_l = 0$.*

If $\alpha \in (\mathbb{B}^n)^\omega$ and $\gamma \in (\mathbb{B}^m)^\omega$, let $\alpha \mathbin{/\!\!/} \gamma$ denote the ω-word over $(\mathbb{B}^{n+m})^\omega$ obtained by stacking α *on top of* γ.

Lemma 2.1. *$\beta := \alpha \mathbin{/\!\!/} \gamma \in (\mathbb{B}^{n+m})^\omega$ satisfies the compatibility conditions if and only if it is the φ-expansion of α.*

Proof. "$\Leftarrow$" direction is obvious. "$\Rightarrow$": Let B_j be the statement $\forall i \geq 0 : (\beta(i))_j = 1 \leftrightarrow \alpha^i \models \varphi_j$. We prove $\forall j \geq 1 : B_j$ by induction on j.

Base case: $\varphi_j = p_j$. Vacuously true.

Inductive case: The only less obvious case is when $\varphi_j = \varphi_k\ \boldsymbol{U}\,\varphi_l$. If $(\beta(i_0))_l = 1$ then for all $i \leq i_0$, the entry $(\beta(i))_j$ is "correct", because of $\varphi_k\ \boldsymbol{U}\,\varphi_l = \varphi_l \vee (\varphi_k \wedge \boldsymbol{X}\,(\varphi_k\ \boldsymbol{U}\,\varphi_l))$ and the induction hypothesis as $k, l < j$. It follows that if $(\beta(i))_l = 1$ for infinitely many i, then the entry $(\beta(i))_j$ is "correct" for all $i \geq 0$. Now suppose for some i_0, we have $(\beta(i))_l = 0$ for all $i \geq i_0$. We claim that for all $i \geq i_0$, $(\beta(i))_j = 0$, and hence the entry is correct; for otherwise we have $(\beta(i))_j = 1$ for *all* $i \geq i_1$, for some $i_1 \geq i_0$ (because of local compatibility for until formulas), and so, violating global compatibility. □

Proof of Theorem 2.1

Lemma 2.2. *The generalised Büchi automaton*

$$A_\varphi \ = \ (\{\, q_0 \,\} \cup \mathbb{B}^{n+m},\ \mathbb{B}^n,\ \Delta,\ q_0,\ \{\, F_1, \cdots, F_p \,\}),$$

defined as follows, accepts $\alpha \in (\mathbb{B}^n)^\omega$ if and only if $\alpha \models \varphi$.

Proof. Write non-initial state $\overline{x}\overline{y} = (x_1, \cdots, x_n, y_1, \cdots, y_m)$. The transition relation Δ is defined as follows: for $\overline{x}\overline{y}$ and $\overline{x}'\overline{y}'$ ranging over $\mathbb{B}^{n+m}$

- $q_0 \xrightarrow{\overline{x}} \overline{x}\overline{y}$ provided $\overline{x}\overline{y}$ satisfies the local compatibility conditions and $y_m = 1$
- $\overline{x}\overline{y} \xrightarrow{\overline{x}'} \overline{x}'\overline{y}'$ provided $\overline{x}\overline{y}$ and $\overline{x}'\overline{y}'$ satisfy the local compatibility conditions (i.e. $\overline{x}\overline{y}$ corresponds to the i-column and $\overline{x}'\overline{y}'$ to the $(i+1)$-column in the table of local compatibility conditions).

For each until subformula $\varphi_j = \varphi_k\ \boldsymbol{U}\,\varphi_l$, a final state-set F containing all states with j-component $= 0$ or l-component $= 1$. Let $F_1, \cdots, F_p$ be all such sets, one for each until subformula of φ. Thus we have

A_φ accepts $\alpha \in (\mathbb{B}^n)^\omega$

iff { Definition of acceptance }

for some A_φ-run $\rho \in (\mathbb{B}^{n+m})^\omega$ on α, each F_j is visited infinitely often

iff { Lemma 2.1 }

ρ is the φ-expansion of α, and $(\rho(0))_{m+n} = 1$

iff { Definition of φ-expansion of α }

$\alpha = \alpha^0 \vDash \varphi_{m+n} = \varphi$

as desired. □

2.5 The LTL Model Checking Problem and its Complexity

Definition 2.5. A Kripke structure $\mathcal{K} = (S, R, \lambda, s_0)$ over $AP = \{p_1, \ldots, p_n\}$ *satisfies* an LTL-formula φ over AP, written $\mathcal{K} \vDash \varphi$, if every label sequence of $\mathcal{K}$ satisfies φ.

LTL Model-Checking Problem Given a Kripke structure $\mathcal{K} = (S, R, \lambda, s_0)$ over the atomic propositions $p_1, \cdots, p_n$, and an LTL-formula φ, does $\mathcal{K}$ *satisfy* φ?

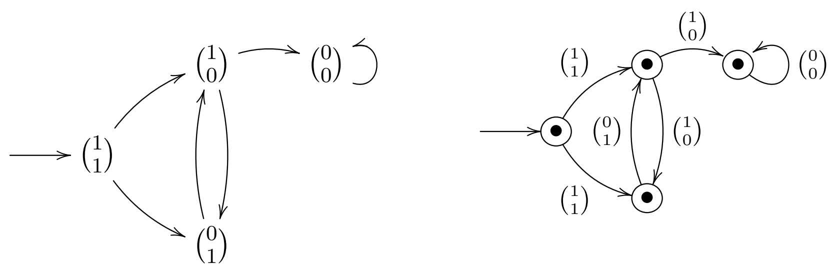
The approach is to verify the negation: Is there a label sequence through K that does *not* satisfy φ? Note that $P \subseteq Q$ iff $P \cap \overline{Q} = \varnothing$.

Label sequences of a given Kripke structure are Büchi recognisable Given a Kripke structure $\mathcal{K} = (S, R, \lambda, s_0)$ over $p_1, \cdots, p_n$. Construct a Büchi automaton $A_\mathcal{K} = (S, \mathbb{B}^n, s_0, \Delta, S)$ whereby

$$(s, (b_1, \cdots, b_n), s') \in \Delta \quad \leftrightarrow \quad (s, s') \in R \text{ and } \lambda(s) = (b_1, \cdots, b_n)$$

Thus each transition of $A_\mathcal{K}$ has the label of the source state, and every state is final. Then $A_\mathcal{K}$ recognises the language of label sequences of $\mathcal{K}$.

Example 2.9. The Büchi automaton on the right recognises the set of label sequences of the Kripke structure on the left.



Proposition 2. *The LTL Model Checking Problem is solvable in time polynomial in the size of the Kripke structure $\mathcal{K}$ and exponential in the size of the formula φ.*

Proof. We give the model checking algorithm as follows.

LTL Model Checking: Given a Kripke structure $\mathcal{K}$ and an LTL formula φ, does $\mathcal{K} \vDash \varphi$?

Algorithm:

1. Construct a Büchi automaton $A_\mathcal{K}$ that recognises the ω-language of all label sequences through $\mathcal{K}$.
2. Construct a generalised Büchi automaton $A_{\neg\varphi}$ that recognises the ω-language of all label sequences that do not satisfy φ.
3. Construct the *intersection automaton* $A_\mathcal{K} \times A_{\neg\varphi}$ i.e. the Büchi automaton that recognises $L(A_\mathcal{K}) \cap L(A_{\neg\varphi})$.
4. Check for *non-emptiness* of $A_\mathcal{K} \times A_{\neg\varphi}$.

Stages 1, 3 and 4 are all polytime. Stage 2 require exponential time in the size of φ.

□

Theorem 2.2 (Sistla and Clarke 1985)**.** *The LTL Model Checking Problem is PSPACE-complete in the size of the formula.*

We present a proof of a result due to Sistla and Clarke (1985). To prove that LTL model checking is solvable in PSPACE, we improve the EXPTIME algorithm of Theorem 2.1. For PSPACE-hardness, we encode polynomial space Turing machines.

An Interlude: Savitch's Algorithm We review the famous result of Savitch (1970); see (Sipser, 2005, Ch. 8 Space Complexity) or (Papadimitriou, 1994).

In time complexity, non-determinism is exponentially more expensive than determinism. But in space complexity, thanks to Savitch, non-determinism is only quadratically more expensive than determinism. Savitch proved that if a nondeterministic Turing machine can solve a problem using $f(n)$ space, then a deterministic Turing machine can solve the same problem in the square of that space bound.

Savitch's insight lies in a method to decide graph reachability which, though wasteful in time, is highly efficient in space. The well-known depth-first and breadth-first graph search algorithms are linear in the size of the graph. Savitch's algorithm could be viewed as "middle-first search" based on the fact that every path of length 2^i has a mid-way point which is reachable from the start, and from which the end is reachable, in no more than 2^{i-1} steps.

Savitch's Algorithm
Input: A finite digraph $G = (V, E)$, vertices $u, v \in V$, $i \in \mathbb{N}$
Output: YES iff there is a path in G from u to v of length at most 2^i

```
$Path(G, u, v, i)$ =
  if $i = 0$
    if $u = v$ or $(u, v) \in E$
      return YES
      else return NO
  for all vertices $w \in V$
    if $Path(G, u, w, i-1)$ and $Path(G, w, v, i-1)$
      return YES
  return NO
```

Theorem 2.3 (Savitch 1970)**.** *Reachability (given a graph $G = (V, E)$ and vertices $u, v \in V$, is there a path from u to v?) can be solved by calling $Path(G, u, v, \log |V|)$, which is computable in space $O(\log^2 |V|)$.*

To obtain the $O(\log^2 |V|)$ space bound, we use a Turing machine to implement the recursive program $Path(G, u, v, \log |V|)$, with its work tape acting like the stack of activation records. At any time, the work tape contains $\log |V|$ or fewer triples of the form (x, y, j) where $x, y \in V$ and $j \leq \log |V|$, where each triple has length at most $3 \log |V|$. For a proof, see for example (Papadimitriou, 1994, p. 149-150).

Corollary 2.1 (Savitch 1970)**.** *For every function $f(n) \geq \log(n)$*

$$NSPACE(f(n)) \subseteq DSPACE(f(n)^2).$$

It follows that **PSPACE** = **NPSPACE**.

Proof. Let P be a problem in $NSPACE(f(n))$. Let M be a nondeterministic Turing machine with space usage bounded by $f(n)$ and accepting P. To determine whether $x \in P$, check whether the configuration graph of M has a path of length at most $2^{O(f(|x|))}$ from the initial to an accepting configuration. This can be done in $DSPACE(O(f(|x|)^2))$. □

LTL Model Checking is in PSPACE

The idea is to use the algorithm of Proposition 2 without building the intersection automaton $A_{\mathcal{K}} \times A_{\neg\varphi}$ in full; rather we compute the states of the automaton *on demand*. From Savitch's algorithm, we know that if the space required to store a state and decide a given transition $q \to q'$ is polynomial in the size of the input, so is the space required to decide reachability $q \to^* q'$.

States of the intersection automaton $A_{\mathcal{K}} \times A_{\neg\varphi}$, which are elements of the shape

$$(s, \overline{x}\, \overline{y}, i) \;\in\; S \times \mathbb{B}^{n+m} \times \{1, \cdots, l\},$$

can be stored in space polynomial in $|\varphi|$ and $|\mathcal{K}|$. Note that $m, l = O(|\varphi|)$. To decide $(s, \overline{x}\, \overline{y}, i) \to (s', \overline{x}'\, \overline{y}', j)$, we need to verify:

- $\overline{x}\,\overline{y}$ and $\overline{x}'\,\overline{y}'$ satisfy the local compatibility conditions, such as $[(\beta(i))_k = 1$ or $(\beta(i))_l = 1]$; since there are only linearly many conditions, they are easy to check.
- i, j are as determined by the global compatibility condition
- Finally $s \to s'$ is a transition in $A_{\mathcal{K}}$.

Thus transitions can be decided in space polynomial in $|\varphi|$. It follows from Savitch that we can decide $(s, \overline{x}\,\overline{y}, i) \to^* (s', \overline{x}'\,\overline{y}', j)$ in polynomial space.

To decide non-emptiness of $L(A_{\mathcal{K}} \times A_{\neg\varphi})$, we seek a "lasso" on a final state $(s, \overline{x}\,\overline{y}, i)$ in the intersection automaton i.e.

$$(s_0, q_0, 0) \to^* (s, \overline{x}\,\overline{y}, i) \to^+ (s, \overline{x}\,\overline{y}, i)$$

by the following algorithm:

```
for all $(s, \overline{x}\,\overline{y}, i)$
  if $(s, \overline{x}\,\overline{y}, i)$ is final and $(s_0, q_0, 0) \to^* (s, \overline{x}\,\overline{y}, i)$
    for all $(s, \overline{x}\,\overline{y}, i) \to (s', \overline{x}'\,\overline{y}', j)$
      if $(s', \overline{x}'\,\overline{y}', j) \to^* (s, \overline{x}\,\overline{y}, i)$
        return YES
return NO
```

Thus we conclude that LTL model checking is in PSPACE.

LTL Model Checking is PSPACE-hard

Let T be a Turing machine with space usage bounded by a polynomial function $s(n)$. WLOG, assume that T loops at each accepting configuration. We shall build a Kripke structure $\mathcal{K}$ and an LTL formula φ such that $\mathcal{K} \not\models \varphi$ iff T can reach an accepting state.

(1) Runs of T can be represented as ω-words.

(2) $\mathcal{K}$ tries to construct all runs of T.

(3) The LTL formula φ asserts that the run in question is *non*-accepting (or malformed).

Runs as ω-words An accepting run of T can be written as a sequence of configurations c_i, separated by a marker $[\!]$ as follows.

$$[\!]\; c_0 \;[\!]\; c_1 \;[\!]\; \cdots \;[\!]\; c_k \;[\!]\; c_k \;[\!]\; c_k \;[\!]\; c_k \;\cdots$$

Each c_i, which has the shape $a_0\, a_1 \cdots (q, a_j) \cdots a_{s(n)}$ where $0 \leq j \leq s(n)$ and a_i ranging over input symbols, represents the configuration comprising the tape

$$a_0\; a_1\; \cdots\; a_{s(n)-1}\; a_{s(n)}\; \Box\; \Box\; \Box\; \Box\; \cdots$$

with control state q and tape head position j. The initial configuration, c_0, is the sequence $\underbrace{(q_0, \Box)\, \Box \cdots \Box}_{s(n)}$. Further, for all i, c_{i+1} follows from c_i, and the control state in c_k is accepting.

Thus an accepting run *is an ω-word* of a certain shape.

$\mathcal{K}$ **constructs every run** by generating *every* possible ω-word.

$$\begin{tikzcd} {} \arrow[r] & \bullet \arrow[loop right, "*"] \end{tikzcd}$$

Recognising a bad run We want φ to characterise exactly the non-accepting or malformed ω-words. Such a word is

(i) *either* not a sequence of configurations. For example $a \; [] \; (q,b) \; [] \; [] \; a \; a \; a \; [] \; (q,a) \; (q,b) \; \cdots$

(ii) *or* it does not start with the initial configuration

(iii) *or* it does not reach a final configuration

(iv) *or* for some i, c_{i+1} cannot follow from from c_i.

Then φ is the disjunction of the formulas describing the respective cases above. We consider them in turn.

Not a sequence of configurations The ω-word is not of the form $[] \underbrace{\cdots}_{s(n)} [] \underbrace{\cdots}_{s(n)} [] \cdots$

$$\neg \left[[] \;\wedge\; \boldsymbol{G} \left([] \Rightarrow \left(\boldsymbol{X}^{s(n)+1} \, [] \wedge \bigwedge_{1 \le i \le s(n)} \boldsymbol{X}^i \neg [] \right) \right) \right]$$

or some configuration does not contain exactly one head.

$$\neg \boldsymbol{G} \, [[] \Rightarrow \boldsymbol{X} (\mathsf{Cell} \; \boldsymbol{U} (\mathsf{Head} \wedge \boldsymbol{X} (\mathsf{Cell} \; \boldsymbol{U} \, [])))]$$

where

$$\begin{aligned} \mathsf{Head} &:= \bigvee_{q,a} (q,a) \\ \mathsf{Cell} &:= \bigvee_{a} a \end{aligned}$$

Initial and final conditions The ω-word:

- does not start with the initial configuration $(q_0, \Box) \; \Box \; \cdots \; \Box$.

$$\neg \left(\boldsymbol{X} (q_0, \Box) \wedge \bigwedge_{2 \le i \le s(n)} \boldsymbol{X}^i \, \Box \right)$$

(Why don't we test for $[]$ characters?)

- or does not reach a final configuration.

$$\neg \boldsymbol{F} \left(\bigvee_{q \text{ final}, a} (q,a) \right)$$

Some c_{i+1} does not follow from c_i Let r range over the transitions of T.

$$\neg \boldsymbol{G} \left(\square \Rightarrow \bigvee_r \mathsf{Follows}_r \right)$$

Let (q, a) be the head of r, q' the next state, b the character to write, $d \in \{-1, 0, 1\}$ is the direction of the head movement. Then

$$\mathsf{Follows}_r := \bigvee_{1 \le i \le s(n)} \left[\begin{pmatrix} \boldsymbol{X}^i \, (q, a) \, \wedge \\ \boldsymbol{X}^{s(n)+1+i} \, \mathsf{Char}(b) \, \wedge \\ \boldsymbol{X}^{s(n)+1+i+d} \, \mathsf{State}(q') \end{pmatrix} \wedge \bigwedge_{j \ne i} \bigvee_a \begin{pmatrix} \boldsymbol{X}^j \, \mathsf{Char}(a) \wedge \\ \boldsymbol{X}^{s(n)+1+j} \, \mathsf{Char}(a) \end{pmatrix} \right]$$

where

$$\begin{aligned} \mathsf{Char}(b) &:= b \vee \bigvee_{q''} (q'', b) \\ \mathsf{State}(q') &:= \bigvee_c (q', c) \end{aligned}$$

Putting all of the above together, we have $\mathcal{K} \not\models \varphi$ iff T has an accepting run.

Further considerations:

- What if the formula is fixed?
- What if the model is fixed?

2.6 Expressive Power of LTL

We say that $L \subseteq \Sigma^\omega$ *non-counting* just if there is an $n_0 \ge 0$ such that for every $n \ge n_0$ and for every $u, v \in \Sigma^*$ and $\beta \in \Sigma^\omega$, we have $u\, v^n \, \beta \in L \;\leftrightarrow\; u\, v^{n+1} \, \beta \in L$. I.e. if L contains an infinite word that embeds a finite word repeated sufficiently often (i.e. more often than the threshold), then for every n larger than the threshold, L contains such an embedded word in which the finite word is repeated n-times.

For example, $(0\,0)^* \, 1^\omega$ is not non-counting: for each $n \ge 0$, $0^{2n} 1^\omega$ matches $(0\,0)^* \, 1^\omega$, but $0^{2n+1} 1^\omega$ does not.

Proposition 3. *Every LTL-definable ω-language is non-counting. It follows that there are Büchi recognisable ω-languages that are not LTL-definable.* □

Some FAQs

(1) What does “linear time” in LTL mean?

Linear-time specifications set same conditions on every infinite path through system modelled by Kripke structure. *Branching-time specifications* are conditions on the structure of tree formed by all paths through a Kripke structure. Well-known logics for describing branching-time properties are computational tree logic (CTL) and CTL*. See (Vardi, 2001) for a readable study on linear-time versus branching-time logics.

(2) Is LTL a “robust” logic? Are there nice characterisations of the LTL-definable languages?

A *star-free regular expression* over Σ is an expression built up using ϵ, symbols $a \in \Sigma$, concatenation, union and complementation with respect to Σ^*. A *star-free regular language* is a language that matches a star-free regular expression.

Theorem 2.4 (Characterisations of LTL Definability). *Let $L \subseteq \Sigma^\omega$. The following are equivalent.*

(i) L is definable in LTL

(ii) L is star-free ω-regular i.e. a finite union of ω-languages of the form $L_1 \cdot L_2^\omega$ where $L_1, L_2 \subseteq \Sigma^*$ are star-free regular

(iii) L is a finite union of ω-languages of the form $\lim L_1 \cap (\Sigma^\omega \setminus \lim L_2)$ where $L_1, L_2 \subseteq \Sigma^*$ are star-free regular. $\square$

(3) What is Kamp's Theorem?

In his UCLA PhD thesis, Kamp (1968) proved that an ω-language is LTL-definable if and only if it is definable in $FO(<, (P_a)_{a\in\Sigma})$ i.e. first-order logic with a binary predicate symbol $<$ and a unary predicate P_a for each $a \in \Sigma$. We write $\mathcal{M}_\alpha = (\omega, <, (P_a)_{a\in\Sigma})$ for the obvious structure determined by $\alpha \in \Sigma^\omega$ where for each $a \in \Sigma$, $n \in P_a \leftrightarrow \alpha(n) = a$.

Theorem 2.5 (Kamp 1968). *Let α be an ω-word over Σ and $n \in \omega$.*

(i) For each LTL formula φ there exists a formula $\chi_\varphi(x)$ in $FO(<, (P_a)_{a\in\Sigma})$ with a free variable x such that

$$\alpha^n \vDash \varphi \;\leftrightarrow\; \mathcal{M}_\alpha \vDash \chi_\varphi(\underline{n}).$$

(ii) For each formula $\chi(x)$ in $FO(<, (P_a)_{a\in\Sigma})$, there exists an LTL formula φ_χ such that

$$\mathcal{M}_\alpha \vDash \chi(\underline{n}) \;\leftrightarrow\; \alpha^n \vDash \varphi_\chi.$$

It follows that for each FO sentence χ there exists an LTL formula φ_χ such that $\mathcal{M}_\alpha \vDash \chi \leftrightarrow \alpha^0 \vDash \varphi_\chi$. $\square$

See (Rabinovich, 2012) for a new proof of Kamp's theorem.

(4) Can we extend LTL to make it *equi-expressive* with Büchi automata (for ω-languages)?

Yes. μLTL: LTL augmented by (a least μ, and hence also greatest ν) fixpoint operators.

Theorem 2.6 (Characterisations of ω-Regularity). *Let $L \subseteq \Sigma^\omega$. The following are equivalent.*

(i) L is ω-regular

(ii) L is definable in μLTL

(iii) L is definable in S1S (see the following chapter)

(iv) L is definable in Weak S1S. $\square$

(5) Is there a characterisation of the subclass of Büchi automata that is equivalent to LTL?

The automata that are equi-expressive with LTL formulas are called *linear weak alternating automata*. For details see (Löding and Thomas, 2000; Gastin and Oddoux, 2001; Hammer et al., 2005).

Problems

2.1 Consider the following properties for the lift system introduced in the introductory chapter:

(A1) Every requested level will be served eventually.

(A2) The lift will return to level 1 again and again.

(A3) Whenever the top level is requested, the lift serves it immediately and does not stop on the way there.

(A4) It is always the case that while moving in one direction, the lift will stop at every requested level, unless the top level is requested.

Assume that the lift serves only four levels. By introducing appropriate atomic propositions (ten should suffice), describe the above properties as LTL-formulas. You should begin by constructing the state-transition graph.

2.2 Let φ, ψ and χ be LTL-formulas. We say that two formulas are *equivalent* if they define the same language. For each of the following, prove or disprove each of the two implications:

(a) $\boldsymbol{F}\,\boldsymbol{G}\,\varphi \equiv \boldsymbol{G}\,\boldsymbol{F}\,\varphi$

(b) $\boldsymbol{X}\,(\varphi \wedge \psi) \equiv \boldsymbol{X}\,\varphi \wedge \boldsymbol{X}\,\psi$

(c) $(\varphi \vee \psi)\ \boldsymbol{U}\,\chi \equiv (\varphi\ \boldsymbol{U}\,\chi) \vee (\psi\ \boldsymbol{U}\,\chi)$

(d) $(\varphi\ \boldsymbol{U}\,\psi)\ \boldsymbol{U}\,\chi \equiv \varphi\ \boldsymbol{U}\,(\psi\ \boldsymbol{U}\,\chi)$

2.3 Let φ and ψ be LTL-formulas. Consider the following temporal operators:

(a) "at next" $\varphi\,\boldsymbol{AX}\,\psi$: When ψ next[2] holds (if it does), so does φ.
(Note that ψ may never hold.)

(b) "while" $\varphi\ \boldsymbol{W}\,\psi$: φ holds for at least as long as ψ does.
(Note that ψ is not assumed to hold at the beginning.)

(c) "before" $\varphi\,\boldsymbol{B}\,\psi$: When ψ next holds (if it does), φ does so before.
(Note that ψ may never hold.)

For each construct, find an equivalent LTL-formula.

[2] We do *not* mean "when φ hold at the next time step", but rather "when φ holds at some point in the future".

2.4 Translate the following LTL formulas to Büchi automata:

(a) $\boldsymbol{F} p_1 \wedge \neg \boldsymbol{F} p_2$

(b) $\boldsymbol{G}(p_1 \rightarrow \boldsymbol{X}(p_2 \, \boldsymbol{U} p_1))$

(c) $\boldsymbol{G}\boldsymbol{F} p_1 \rightarrow \boldsymbol{F}\boldsymbol{G} p_2$.

2.5 We define a sublogic $\mathcal{T}(\boldsymbol{U})$ of LTL consisting of formulas that are built up from the atomic propositions, using conjunction, negation and the until-operator $\varphi \, \boldsymbol{U} \psi$. (Thus we may write LTL as $\mathcal{T}(\boldsymbol{X}, \boldsymbol{U})$.)

Suppose there is only one atomic proposition, p_1. Consider the label sequence

$$\alpha \; = \; (1)(1)(0)(0)\cdots$$

(a) Prove, by structural induction on formulas, that for all $\varphi \in \mathcal{T}(\boldsymbol{U})$, we have $\alpha^0 \vDash \varphi$ iff $\alpha^1 \vDash \varphi$.

(b) Find an LTL-formula ψ satisfying $\alpha^0 \vDash \psi$ and $\alpha^1 \nvDash \psi$.

(c) Hence prove that $\mathcal{T}(\boldsymbol{U})$ is strictly less expressive than LTL.

2.6 Prove that for each generalised Büchi automaton

$$A = (Q, \Sigma, \Delta, q_0, \{ F_0, \cdots, F_{l-1} \})$$

there is an equivalent Büchi automaton A' i.e. they recognise the same ω-language.

2.7 Consider the LTL-formula $\varphi = p_1 \, \boldsymbol{U}(\boldsymbol{X} p_2)$.

(a) Let $\alpha \in (\{0, 1\}^2)^\omega$. Formulate the compatibility conditions for the φ-expansion of α.

(b) Construct a generalised Büchi automaton A equivalent to φ. What are the final states of A?

(c) Construct directly a Büchi automaton recognising $L = \{ \alpha \in (\{0, 1\}^2)^\omega : \alpha \vDash \varphi \}$.

2.8

(a) Show that the ω-language $L_1 = \{ (01)^\omega \}$ is non-counting.

(b) Show that the ω-language $L_2 = \{ 01(0101)^* 0^\omega \}$ is counting.

2.9 An ω-language L over an alphabet Σ is said to be *stuttering* if for each letter $a \in \Sigma$, we have

$$u \, a \, \beta \in L \; \leftrightarrow \; u \, a \, a \, \beta \in L$$

(where u ranges over Σ^* and β over Σ^ω). Which of the following are true? Justify your answers.

(a) If φ is an LTL formula then $L(\varphi)$ is stuttering.

(b) If φ is an LTL formula without $\boldsymbol{X}$(-) then $L(\varphi)$ is stuttering.

2.10 Let p be the only atomic proposition. For $n \geq 0$, let γ_n be the infinite word $1^n 0 1^\omega$. Thus if $i \neq n$ then $\gamma_n^i \vDash p$, and $\gamma_n^n \nvDash p$.

Let φ be an LTL-formula. Prove, by structural induction, that if φ has no more than n occurrences of $\boldsymbol{X}(-)$, then for all $i, j > n$, we have $\gamma_j \vDash \varphi$ iff $\gamma_i \vDash \varphi$.

[*Hint.* For the inductive case of $\varphi = \varphi_1 \, \boldsymbol{U} \, \varphi_2$: $\gamma_i \vDash \varphi$ means that for some $l \geq 0$, we have $\gamma_i^l \vDash \varphi_2$, and $\gamma_i^k \vDash \varphi_1$ for all $0 \leq k < l$. Consider the cases of $l \leq i$ and $l > i$ in turn. In the former case, analyse the cases of $i - l > n$ and $i - l \leq n$. Note that $\gamma_i^l = \gamma_{i-l}$ if $l \leq i$. What is γ_i^l if $l > i$?]

2.11 † We define a sublogic $\mathcal{T}(\boldsymbol{X}, \boldsymbol{G})$ of LTL consisting of formulas that are built up from the atomic propositions, using conjunction, negation, next-time operator $\boldsymbol{X}\varphi$, and the always-modality $\boldsymbol{G}\varphi$. We shall prove:

Theorem 2.7. *$\mathcal{T}(\boldsymbol{X}, \boldsymbol{G})$ is strictly less expressive than LTL.*

Consider a Kripke structure (over atomic propositions p_1, p_2) that has states $s_0, \cdots, s_{4m-1}$ and the transition relation is specified by:

$$s_i \to s_j \quad \Longleftrightarrow \quad j = i + 1 \text{ or } (i = 4m - 1 \text{ and } j = 2m).$$

The proposition p_1 is assumed to hold for all states except s_{3m}, and p_2 is assumed to hold for just the states s_{2m-1} and s_{4m-1}. Let α be the uniquely determined label sequence starting in state s_0. I.e.

$$\alpha = \underbrace{\binom{1}{0} \cdots \binom{1}{0} \binom{1}{1}}_{2m} \left(\underbrace{\underbrace{\binom{1}{0} \cdots \binom{1}{0}}_{m} \binom{0}{0} \binom{1}{0} \cdots \binom{1}{0} \binom{1}{1}}_{2m} \right)^{\omega}$$

(a) List the elements of the set $\{ 0 \leq l \leq 4m - 1 : \alpha^l \vDash p_1 \, \boldsymbol{U} p_2 \}$.

(b) Prove that for all $\varphi \in \mathcal{T}(\boldsymbol{X}, \boldsymbol{G})$ containing fewer than $m - 1$ occurrences of $\boldsymbol{X}$, we have

$$\alpha^0 \vDash \varphi \quad \Longleftrightarrow \quad \alpha^{2m} \vDash \varphi.$$

(c) Observe that $\alpha^0 \vDash p_1 \, \boldsymbol{U} p_2$ and $\alpha^{2m} \nvDash p_1 \, \boldsymbol{U} p_2$. Hence or otherwise prove that $\mathcal{T}(\boldsymbol{X}, \boldsymbol{G})$ is strictly less expressive than LTL.

2.12 The aim of the problem is to establish the co-NP-hardness of the LTL Model Checking Problem. We will do this by reducing 3-UNSAT (a co-NP-complete problem) to it.

Given an instance of the 3-UNSAT problem, we want to produce in polynomial time an instance of the LTL Model Checking Problem. More precisely, we want to prove that a propositional formula ψ in conjunctive normal form (with three literals per clause, where a literal is a variable or the negation of a variable) can be transformed in polynomial time into a Kripke structure $\mathcal{K}_\psi$ and an LTL-formula F_ψ such that ψ is satisfiable if, and only if, the pair $(\mathcal{K}_\psi, F_\psi)$ is a no-instance of the LTL Model Checking Problem.

Take $\psi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$ which can be satisfied with the assignment $x_1 \mapsto 1, x_2 \mapsto 0, x_3 \mapsto 0$.

Consider the following Kripke structure $\mathcal{K}_\psi$

$$\begin{tikzcd}
 & & \overset{x_1}{\binom{1}{0}} \arrow[dr] & & \overset{x_2}{\binom{0}{0}} \arrow[dr] & & \overset{x_3}{\binom{1}{0}} \arrow[dr] & \\
{} \arrow[r] & \underset{\binom{0}{0}}{y_0} \arrow[ur] \arrow[dr] & & \underset{\binom{0}{0}}{y_1} \arrow[ur] \arrow[dr] & & \underset{\binom{0}{0}}{y_2} \arrow[ur] \arrow[dr] & & \underset{\binom{0}{0}}{y_3} \arrow[loop right] \\
 & & \underset{\neg x_1}{\binom{0}{1}} \arrow[ur] & & \underset{\neg x_2}{\binom{1}{1}} \arrow[ur] & & \underset{\neg x_3}{\binom{0}{1}} \arrow[ur] &
\end{tikzcd}$$

The labels are defined as follows: the j-component of the label of the literal ℓ is 1 just if ℓ occurs in the j-clause of ψ. The idea is that an assignment determines a path through the Kripke structure.

(a) What is F_ψ for the above example?

(b) By giving the construction $\psi \mapsto \mathcal{K}_\psi$ and the corresponding formula F_ψ, prove that there is a polynomial reduction of the desired kind.

Thus conclude that the LTL Model Checking Problem is co-NP-hard.

Chapter 3

S1S

Synopsis

Examples. Syntax and semantics. Büchi-recognisable languages are S1S-definable. S1S-definable languages are Büchi-recognisable.

References (Büchi, 1960a; Elgot, 1961; Büchi, 1962; Muller, 1963)

3.1 Introduction

In this chapter, we will consider a logic called S1S that is equivalent in expressive power to Büchi automata, i.e. it can define precisely the ω-regular languages.

S1S is the monadic second-order logic (MSO) of one-successor. *Second-order* means that we allow quantification over *relations*; *monadic* means that quantification is restricted to *monadic* (or *unary*) relations, namely, sets.

Why study S1S?

- *Historical importance*: the first time automata and logic connection is established and exploited.
- *Powerful decidable theory*: many decisions problems can be shown decidable by reduction to S1S.
- monadic second-order logic is considered a gold standard of logics for describing correctness properties of reactive systems.

3.2 The logical system S1S

The *vocabulary* consists of a unary function symbol $\mathbf{s}$ and a binary predicate symbol $\in$. The corresponding *logical structure* is $(\omega, \mathbf{s}, \in)$ where[1] $\mathbf{s}$ is the successor function $x \mapsto x+1$, and $\in \subseteq \omega \times 2^{\omega}$ is the standard membership relation between elements and sets.

[1]We use the same LaTeX-symbol for the *function symbol* $\mathbf{s}$ (in the vocabulary) and its *interpretation* (in the logical structure) – it should be clear from the context which is intended; similarly for $\in$.

The logical system S1S is defined as follows.

- *Variables.* 1st-order variables (x, y, z, etc.) range over natural numbers (regarded as positions in ω-words). 2nd-order variables (X, Y, Z, etc.) range over sets of natural numbers.
- *Terms.* 1st-order variables are terms. If t is a term, so is $\mathbf{s}\,t$.
- *Formulas. Atomic formulas* are of the shape $t \in X$ where t is a term and X is a 2nd-order variable.
  S1S formulas are built up from atomic formulas using standard Boolean connectives, with $\forall$- and $\exists$-quantifications over 1st and 2nd-order variables.

Constructs definable in S1S Note that 0 and the atomic formulas $s = t$ and $s < t$ are definable in terms of set-membership and successor.

- "$x = y$" $:=$ $\forall X. x \in X \leftrightarrow y \in X$
- "$X \subseteq Y$" $:=$ $\forall x. x \in X \rightarrow x \in Y$
- "$X = Y$" $:=$ $X \subseteq Y \,\wedge\, Y \subseteq X$
- "$x = 0$" $:=$ $\forall y. \neg(x = \mathbf{s}\,y)$ "x has no predecessor"
- "$x = 1$" $:=$ $x = \mathbf{s}\,0$
- "$x \leq y$" $:=$ $\forall X.(x \in X \wedge (\forall z. z \in X \rightarrow \mathbf{s}\,z \in X)) \;\rightarrow\; y \in X$
  "Every set X that contains x and is closed under successor (in particular, the *smallest* such X) also contains y."
- "X is finite" $:=$ $\exists x. \forall y.(y \in X \rightarrow y \leq x)$

Subsystems of S1S

- *First-order fragment: S1S*$_1$. Formulas are built up from atomic formulas ($s \in X, s = t$ and $s < t$ where s and t are terms) using boolean connectives and first-order quantifiers.
- *Existential S1S.* Formulas are S1S$_1$-formulas preceded by a block $\exists Y_1 \cdots Y_m$ of existential second-order quantifiers.

We remark that in these subsystems we will treat $=$ and $<$ as relations in their own right, not as shorthand for the S1S formulas that define these relations using $\mathbf{s}$, $\in$, and second-order quantifiers.

3.3 Semantics of S1S

Write $\varphi(x_1, \cdots, x_m, X_1, \cdots, X_n)$ to mean: φ has free 1st-order variables from $x_1, \cdots, x_m$ and free 2nd-order variables from $X_1, \cdots, X_n$. Let $a_i \in \omega$ and $P_j \subseteq \omega$. For $\overline{a} = a_1, \cdots, a_m$ and $\overline{P} = P_1, \cdots, P_n$, we write

$$(\omega, \mathbf{s}, \in); \overline{a}; \overline{P} \;\vDash\; \varphi(x_1, \cdots, x_m, X_1, \cdots, X_n)$$

or simply $\overline{a}, \overline{P} \vDash \varphi$, to mean " the structure $(\omega, \mathbf{s}, \in)$ with the assignment $\overline{x} \mapsto \overline{a}; \overline{X} \mapsto \overline{P}$ satisfies φ".

Definition 3.1. We define the *satisfaction relation*

$$\overline{a}; \overline{P} \vDash \varphi(\overline{x}, \overline{X})$$

by recursion over the syntax of φ:

$$\begin{array}{rcl}
\overline{a};\overline{P} \vDash \underbrace{\mathbf{s}(\cdots(\mathbf{s}}_{k}\ x_i)\cdots) \in X_j & := & a_i + k \in P_j \\
\overline{a};\overline{P} \vDash \neg\varphi(\overline{x},\overline{X}) & := & \overline{a};\overline{P} \nvDash \varphi(\overline{x},\overline{X}) \\
\overline{a};\overline{P} \vDash \varphi_1(\overline{x},\overline{X}) \vee \varphi_2(\overline{x},\overline{X}) & := & \overline{a};\overline{P} \vDash \varphi_1(\overline{x},\overline{X}) \quad \text{or} \quad \overline{a};\overline{P} \vDash \varphi_2(\overline{x},\overline{X}) \\
\overline{a};\overline{P} \vDash \exists y.\varphi(\overline{x},y,\overline{X}) & := & \overline{a},b;\overline{P} \vDash \varphi(\overline{x},y,\overline{X}) \text{ for some } b \in \omega \\
\overline{a};\overline{P} \vDash \exists Z.\varphi(\overline{x},\overline{X},Z) & := & \overline{a};\overline{P},Q \vDash .\varphi(\overline{x},\overline{X},Z) \text{ for some } Q \subseteq \omega
\end{array}$$

Standardly $\varphi_1 \wedge \varphi_2$ is equivalent to $\neg\varphi_1 \vee \neg\varphi_2$, and $\forall X.\varphi$ and $\forall x.\varphi$ are equivalent to $\neg(\exists X.\neg\varphi)$ and $\neg(\exists x.\neg\varphi)$ respectively.

Representing a set of natural numbers as an infinite word We represent any $P \subseteq \omega$ by its *characteristic word*, written $\ulcorner P \urcorner \in \mathbb{B}^\omega$, defined by

$$\ulcorner P \urcorner(i) = 1 \quad \leftrightarrow \quad i \in P.$$

E.g. the characteristic words of the set of multiples of 3 and the set of prime numbers are respectively:

$$1001001001001001001001001\cdots$$
$$0011010100010100010100001\cdots$$

We represent $a \in \omega$ by the characteristic word of the singleton set $\{ a \}$.

More generally the *characteristic word* of a tuple

$$(a_1, \cdots, a_m, P_1, \cdots, P_n) \ \in \ \omega^m \times (2^\omega)^n$$

written $\ulcorner a_1, \cdots, a_m, P_1, \cdots, P_n \urcorner$, is an infinite word over the alphabet $\mathbb{B}^{m+n}$ such that each of the $m+n$ tracks (or rows) is the characteristic word of the corresponding component of the tuple $(\overline{a}, \overline{P})$.

Defining ω-languages by S1S formulas We say $L \subseteq \mathbb{B}^\omega$ is *S1S-definable* by $\varphi(X)$ just if $L = \{ \ulcorner P \urcorner \in \mathbb{B}^\omega : P \vDash \varphi(X) \}$. I.e. Each P that satisfies $\varphi(X)$ contains exactly the numbers denoting the positions of '1' in an ω-word in $L \subseteq \mathbb{B}^\omega$.

Example 3.1. (i) The set $L_1 = \{ \alpha \in \mathbb{B}^\omega : \alpha \text{ has infinitely many 1s} \}$ is first-order definable by $\varphi_1(X) = \forall x.\exists y.x < y \ \wedge \ y \in X$.

(ii) $(00)^*1^\omega$ is definable by

$$\varphi_2(X) \ = \ \exists Y.\exists x. \left(\begin{array}{ll} & 0 \in Y \\ \wedge & \forall y.y \in Y \leftrightarrow \mathbf{s}\, y \notin Y \\ \wedge & x \in Y \\ \wedge & \forall z.z < x \rightarrow z \notin X \\ \wedge & \forall z.x \leq z \rightarrow z \in X \end{array} \right)$$

(What is Y?) Recall that $(00)^*1^\omega$ is a "counting" language, hence not LTL-definable.

Translating LTL to S1S Fix atomic formulas $p_1, \cdots, p_n$. We say S1S-formula $\varphi(X_1, \cdots, X_n)$ is *equivalent* to an LTL-formula ψ just if $[\![\psi]\!] = \{ \ulcorner \overline{P} \urcorner \in (B^n)^\omega : \overline{P} \vDash \varphi(\overline{X}) \}$.

Example 3.2. (i) $\boldsymbol{X}\,\boldsymbol{X}(p_2 \to \boldsymbol{F}\,p_1)$

$$\varphi_3(X_1, X_2) \;=\; \mathbf{s}\,\mathbf{s}\,0 \in X_2 \to \exists x.(\mathbf{s}\,\mathbf{s}\,0 \leq x \wedge x \in X_1)$$

(ii) $\boldsymbol{F}(p_1 \wedge \boldsymbol{X}(\neg p_2\ \boldsymbol{U}\,p_1))$

$$\begin{array}{ll} & \varphi_4(X_1, X_2) \\ = & \exists x. \left(\begin{array}{ll} & x \in X_1 \\ \wedge & \exists y. \left(\begin{array}{ll} & \mathbf{s}\,x \leq y \\ \wedge & y \in X_1 \\ \wedge & \forall z.(\mathbf{s}\,x \leq z \wedge z < y) \to \neg(z \in X_2) \end{array} \right) \end{array} \right) \end{array}$$

3.4 Büchi-Recognisable Languages are S1S-Definable

Definition 3.2. An ω-language $L \subseteq (\mathbb{B}^n)^\omega$ is *S1S definable* just if there is an S1S-formula $\varphi(X_1, \cdots, X_n)$ such that $L = \{ \ulcorner P_1, \cdots, P_n \urcorner \in (\mathbb{B}^n)^\omega : \overline{P} \vDash \varphi(\overline{X}) \}$.

Theorem 3.1 (Büchi)**.** *For every Büchi automaton A over the alphabet $\mathbb{B}^n$, there is an S1S formula $\varphi_A(X_1, \cdots, X_n)$ such that for every $(P_1, \cdots, P_n) \in (2^\omega)^n$, we have $\overline{P} \vDash \varphi_A(\overline{X})$ if and only if A accepts $\ulcorner P_1, \cdots, P_n \urcorner$.*

The proof idea is simple: take a Büchi automaton $A = (Q, \Sigma, q_1, \Delta, F)$ where $\Sigma = \mathbb{B}^n$, and construct an S1S-formula $\varphi_A(X_1, \cdots, X_n)$ that asserts "there is an accepting run of A on input given by the characteristic word of $(X_1, \cdots, X_n)$".

Proof. The aim is to code a run. Suppose $Q = \{ q_1, \cdots, q_m \}$. A run $\rho(0)\rho(1) \cdots \in Q^\omega$ is coded by m subsets of ω, namely $Y_1, \cdots, Y_m$, such that

$$i \in Y_k \quad \leftrightarrow \quad \rho(i) = q_k$$

Clearly $Y_1, \cdots, Y_m$ form a *partition* of ω. Each tuple "$Y_1, \cdots, Y_m$" describes an infinite word over Q, α, whereby each

$$Y_j = \{ \text{positions in } \alpha \text{ with symbol } q_j \}.$$

Define $\mathit{partition}(Y_1, \cdots, Y_m)$ to be

$$\forall x. \left(\bigvee_{i=1}^{m} x \in Y_i \right) \quad \wedge \quad \neg \left(\exists y. \bigvee_{i \neq j} (y \in Y_i \wedge y \in Y_j) \right)$$

Coding letters of the alphabet $\mathbb{B}^n$ For $a = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix} \in \mathbb{B}^n$, introduce shorthand

$$x \in X_a \;:=\; [b_1](x \in X_1) \wedge [b_2](x \in X_2) \wedge \;\cdots\; \wedge [b_n](x \in X_n)$$

where

$$[b_i](x \in X_i) \;:=\; \begin{cases} x \in X_i & \text{if } b_i = 1 \\ \neg(x \in X_i) & \text{otherwise} \end{cases}$$

Given a Büchi automaton $A = (\{1, \cdots, m\}, \mathbb{B}^n, 1, \Delta, F)$, define $\varphi_A(X_1, \cdots, X_n)$ to be

$$\exists Y_1 \cdots Y_m \,.\, \left(\begin{array}{ll} & \mathit{partition}(Y_1, \cdots, Y_m) \\ \wedge & 0 \in Y_1 \\ \wedge & \forall x. \bigvee_{(i,a,j)\in\Delta}(x \in Y_i \;\wedge\; x \in X_a \;\wedge\; \mathbf{s}\, x \in Y_j) \\ \wedge & \forall x.\exists y.(x < y \;\wedge\; \bigvee_{i\in F} y \in Y_i) \end{array} \right)$$

Thus for every $(P_1, \cdots, P_n) \in (2^\omega)^n$, A accepts $\ulcorner P_1, \cdots, P_n \urcorner$ iff $\overline{P} \vDash \varphi_A(X_1, \cdots, X_n)$. □

Observe that $\varphi_A(X_1, \cdots, X_m)$ is an existential S1S-formula (recall that in this subsystem, we treat $<$ as an atomic relation).

3.5 S1S-Definable Languages are Büchi-Recognisable

Theorem 3.2 (Büchi). *For every S1S formula $\varphi(x_1, \cdots, x_m, X_1, \cdots, X_n)$, there is an equivalent non-determinstic Büchi automaton A_φ over alphabet $\mathbb{B}^{m+n}$, in the sense that*

$$L(A_\varphi) \;=\; \{\ulcorner a_1, \cdots, a_m, P_1, \cdots, P_n \urcorner \in (\mathbb{B}^{m+n})^\omega \mid \overline{a}, \overline{P} \vDash \varphi\}$$

Proof. The proof is by induction on the size of φ. An *atomic formula* has the form $\underbrace{\mathbf{s}\,(\mathbf{s} \cdots (\mathbf{s}}_{k}\; x_i) \cdots) \in X_j$. We build a Büchi automaton to read the tracks i and $m + j$ only (corresponding to x_i and X_j respectively), performing the following check: if the unique 1 of the x_i-track is at position l (say), then the X_j-track has a 1 in position $l + k$.

Disjunction: Consider $\varphi_1(\overline{x}, \overline{X}) \vee \varphi_2(\overline{x}, \overline{X})$. By the induction hypothesis, suppose automata A_{φ_1} and A_{φ_2} are equivalent to φ_1 and φ_2 respectively. Set $A_{\varphi_1\vee\varphi_2}$ to be the Büchi automaton that accepts the union of (the respectively ω-languages of) A_{φ_1} and A_{φ_2}.

Negation: Consider $\neg\varphi(\overline{x}, \overline{X})$. By the induction hypothesis, suppose A_φ is equivalent to φ. Set $A_{\neg\varphi}$ to be the automaton that recognises the complement of $L(A_\varphi)$.

Second-order existential quantification: Consider $\exists Y.\varphi(\overline{x}, \overline{X}, Y)$. By the induction hypothesis, suppose A_φ is the Büchi automaton equivalent to $\varphi(\overline{x}, \overline{X}, Y)$. I.e. for all $(\overline{a}, \overline{P}, Q) \in \omega^m \times (2^\omega)^{n+1}$,

$$\overline{a}, \overline{P}, Q \vDash \varphi \quad \leftrightarrow \quad A_\varphi \text{ accepts } \ulcorner \overline{a}, \overline{P}, Q \urcorner$$

We construct $A_{\exists Y.\varphi}$ by replacing each transition label $\begin{pmatrix} b_1 \\ \vdots \\ b_{m+n} \\ b \end{pmatrix}$ in A_φ by $\begin{pmatrix} b_1 \\ \vdots \\ b_{m+n} \end{pmatrix}$ thus introducing (further) non-determinacy.

Consequently a transition via $\begin{pmatrix} b_1 \\ \vdots \\ b_{m+n} \end{pmatrix}$ in $A_{\exists Y.\varphi}$ corresponds to a transition via $\begin{pmatrix} b_1 \\ \vdots \\ b_{m+n} \\ 0 \end{pmatrix}$ or $\begin{pmatrix} b_1 \\ \vdots \\ b_{m+n} \\ 1 \end{pmatrix}$ in A_φ. Hence

$$\begin{array}{ll} & A_{\exists Y.\varphi} \text{ accepts } \ulcorner \overline{a}, \overline{P} \urcorner \in (\mathbb{B}^{m+n})^\omega \\ \text{iff} & \text{for some } c_0 c_1 \cdots \in \mathbb{B}^\omega \text{, we have } A_\varphi \text{ accepts } \binom{\alpha(0)}{c_0} \binom{\alpha(1)}{c_1} \cdots \\ & \text{(Suppose } c_0 c_1 \cdots = \ulcorner Q \urcorner \text{ and } \alpha = \ulcorner \overline{a}, \overline{P} \urcorner\text{.)} \\ \text{iff} & \text{for some } Q \subseteq \omega \text{ we have } \overline{a}, \overline{P}, Q \vDash \varphi(\overline{x}, \overline{X}, Y) \\ \text{iff} & \overline{a}, \overline{P} \vDash \exists Y.\varphi(\overline{x}, \overline{Y}) \end{array}$$

First-order existential quantification: Consider $\exists y.\varphi(\overline{x}, y, \overline{X})$. Exactly the same as above. $\square$

The theory S1S The *theory S1S* is the set of S1S sentences that are satisfied in the structure $(\omega, \mathbf{s}, \in)$. For instance, $\forall X.\exists Y.\forall x.(x \in X \to x \in Y)$ is part of the theory, but $\forall X.\exists y.\forall x.(x \in X \to x < y)$ is not.

A corollary of the previous theorem (and the decidability of non-emptiness for Büchi automata) is that given an S1S sentence φ, it is decidable whether or not φ is in the theory S1S, by constructing A_φ and testing whether $L(A_\varphi)$ is non-empty.

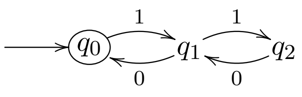
Corollary 3.1 (Büchi)**.** *The theory S1S is decidable.*

However, there is no elementary time decision procedure for membership in the theory S1S. That is, there is no (fixed) $h \geq 0$ such that for all S1S sentences φ, it is decidable in time bounded by $exp_h(n)$ whether φ is in the theory, where $exp_h(n)$ is the tower-of-exponentials function of height h:

$$exp_0(n) := n \qquad exp_{h+1}(n) := 2^{exp_h(n)}.$$

Problems

3.1 Consider the following Büchi automaton A:



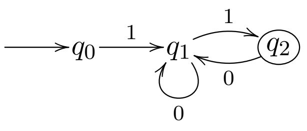
(a) Construct an Existential S1S-formula equivalent to A.

(b) Construct an LTL-formula equivalent to A.

(c) A *star-free ω-regular language* over an alphabet Σ is a finite union of ω-languages of the form $U \cdot V^\omega$, where U and V are (regular) languages constructed from a finite set of finite words over Σ using the Boolean operations, namely, complementation, union and concatenation.

Prove that A recognises a star-free regular ω-language.

3.2 Let A be the following Büchi automaton A:



Construct an S1S-formula $\varphi(X)$ such that $\alpha \in \mathbb{B}^\omega$ satisfies φ iff A accepts α.

3.3

(a) Prove that for every LTL formula φ over a single proposition p, there is a formula $\widetilde{\varphi}(X)$ in S1S$_1$ such that φ and $\widetilde{\varphi}(X)$ define the same ω-language.

(b) Prove that *equi-cardinality* of sets, that is, the predicate

$$EqCard(A, B) \;:=\; A, B \subseteq \omega \text{ have the same cardinality}$$

cannot be expressed in S1S.

3.4 Give S1S-formulas $\varphi_1(X_1, X_2)$ and $\varphi_2(X_1, X_2)$ for the following ω-languages:

(a) $L_1 = \binom{1}{1}\binom{1}{0}^*\begin{pmatrix}1&0\\1&0\end{pmatrix}^\omega$

(b) $L_2 = \begin{pmatrix}1&1&1\\1&1&1\end{pmatrix}\binom{0}{1}^\omega$

Explain the purpose of the main subformulas of $\varphi_1(X_1, X_2)$ and $\varphi_2(X_1, X_2)$.

3.5 Show that (natural numbers) addition $x = y + z$ is not definable in S1S.

[*Hint.* Show that S1S-definability of addition would imply that the language $\{ a^n b^n c^\omega : n \geq 0 \}$ is Büchi recognizable.]

3.6 *Presburger arithmetic* is first-order logic over the structure $(\omega, +)$ where

$$+ \;=\; \{ (a, b, c) \in \omega^3 : a + b = c \}.$$

For example, $\exists y.\forall x.(x + y = x)$ is a sentence of Presburger arithmetic that holds in $(\omega, +)$.

The goal in this question is to use the decidability of S1S to prove that Presburger arithmetic is decidable. The idea is to encode numbers in the Presburger arithmetic formulas using sets in an S1S formula. A number can be represented as a finite set of numbers corresponding to the positions of 1s in its (reverse) binary representation. For instance, we could represent the number 9 by the set $P = \{ 0, 3 \}$, since the characteristic word of P is $\ulcorner P \urcorner := 10010^\omega$, the reverse binary representation of 9.

(a) Show that there is an S1S formula $\varphi(X, Y, Z)$ asserting that the numbers a, b and c represented respectively by the finite sets X, Y and Z are related by the equation $a + b = c$. For instance, the sets $X = \{ 0, 3 \}$, $Y = \{ 0, 1 \}$, and $Z = \{ 2, 3 \}$ should satisfy the formula φ since X represents 9, Y represents 3, and Z represents 12.

(b) Deduce that formulas of Presburger arithmetic can be translated into S1S.

Hence prove that Presburger arithmetic is decidable. Why does this not contradict the preceding question?

3.7 *Weak monadic second-order theory of one successor*, WS1S, is defined in the same way as S1S except that second-order variables range over only finite sets of natural numbers.

(a) Fix a deterministic Muller automaton A. Since it is not possible to say anything in WS1S about any complete run directly, we restrict ourselves to prefixes of runs. Note that every ω-word that is accepted by a deterministic Muller automaton has a *unique* accepting run.

Give a WS1S-formula that defines the ω-language recognized by A.

(b) Hence deduce that an ω-language is S1S-definable iff it is WS1S-definable.

Chapter 4

Modal Mu-Calculus

Synopsis

Knaster-Tarski fixpoint theorem. Syntax and semantics. Syntactic approximants via infinitary syntax. Intuitions from examples. A branching-time temporal logic: computational tree logic (CTL).

References (Bradfield and Stirling, 2001, 2007; Stirling, 2001, 1997) For a primer on ordinals and Knaster-Tarski Theorem, see (Kozen, 2006, pp. 35-43).

Background Modal mu-calculus's defining feature – use of least and greatest fixpoint operators. The idea goes back a long way:

- Fixpoints in program logics: de Bakker, Park and Scott (late 60s).
- Fixpoints in modal logics of programs: Pratt (1980), Emerson and Clark (1980), Kozen (1983).

Formulas of modal mu-calculus are notoriously hard to read. A good intuitive appreciation is essential for understanding the theory.

4.1 Knaster-Tarski Fixpoint Theorem

Posets, Supremums and Infimums: A Revision A *partially-ordered set* is a pair $\langle L, \leq \rangle$ such that $\leq$ is a binary relation over L that is

(i) reflexive: for every $x \in L$, $x \leq x$

(ii) antisymmetric: for every $x, y \in L$, if $x \leq y$ and $y \leq x$ then $x = y$

(iii) transitive: for every $x, y, z \in L$, if $x \leq y$ and $y \leq z$ then $x \leq z$

Let $M \subseteq L$. An element $l \in L$ is the *least upper bound* (LUB, or supremum) of M, written $\bigvee M$, just if:

(i) for all $x \in M$, $x \leq l$

(ii) for all $y \in L$, if for all $x \in M$ we have $x \leq y$, then $l \leq y$.

Similarly for *greatest lower bound* (GLB, or infimum), written $\bigwedge M$.

Complete lattices and monotone functions A *complete lattice* is a partially-ordered set $\langle L, \leq \rangle$ in which every subset $M \subseteq L$ has a least upper bound $\bigvee M$ and a greatest lower bound $\bigwedge M$ in L. Every such L has a greatest $\bigvee L$ ($= \bigwedge \varnothing$) and least element $\bigwedge L$ ($= \bigvee \varnothing$).

Example 4.1. (i) Is $\langle \omega, \leq \rangle$ a complete lattice? No, because $\bigvee \omega = \omega \notin \omega$.

(ii) $\langle \mathcal{P}(S), \subseteq \rangle$ is a complete lattice. For every $\mathcal{U} \subseteq \mathcal{P}(S)$, we have

$$\bigvee \mathcal{U} \;=\; \bigcup \mathcal{U} \;:=\; \{\, s \in S : \exists U \in \mathcal{U} \,.\, s \in U \,\}.$$

The least and greatest elements are $\varnothing$ and S respectively.

A function from L to L is said to be *monotone* just if $f(x) \leq f(y)$ whenever $x \leq y$. An element $x \in L$ is a *fixpoint* of f just if $f(x) = x$. We say x is a *postfixed point* of f just if $x \leq f(x)$; x is a *prefixed point* of f if $f(x) \leq x$.

Example 4.2. (i) Reals: $\langle \mathbb{R}, \leq \rangle$

(ii) $\langle \mathbb{N} \cup \{\omega\}, \leq \rangle$

(iii) Divisibility: $\langle \mathbb{Z} \setminus \{0\}, \_ \mid \_ \rangle$ where $x \mid y := \exists u \,.\, x \times u = y$.

(iv) Words ordered by prefix: $\langle \Sigma^*, \leq_{\text{pref}} \rangle$. E.g. $a\,b <_{\text{pref}} a\,b\,b\,a$.

(v) Lexicographical: $\langle \Sigma^*, \leq_{\text{lexico}} \rangle$. E.g. $a\,b\,b\,a <_{\text{lexico}} a\,b\,c$ (assuming that Σ is linearly ordered).

(vi) Subword: $\langle \Sigma^*, \leq_{\text{subw}} \rangle$, with $a\,b\,a \leq_{\text{subw}} b\,a\,a\,b\,b\,a\,a$

	Poset	Complete Lattice
$\langle \mathbb{R}, \leq \rangle$	Y	N
$\langle \mathbb{N} \cup \{\omega\}, \leq \rangle$	Y	Y
$\langle \mathbb{Z} \setminus \{0\}, \_ \mid \_ \rangle$	N	N
$\langle \Sigma^*, \leq_{\text{pref}} \rangle$	Y	N
$\langle \Sigma^*, \leq_{\text{lexico}} \rangle$	Y	N
$\langle \Sigma^*, \leq_{\text{subw}} \rangle$	Y	N

Least prefixed and greatest postfixed points

Lemma 4.1. *Let L be a complete lattice and $f : L \to L$ be a monotone function.*

(i) The least prefixed point *of f, denoted $lpr(f)$, exists, and is $\bigwedge \{\, x \in L : f(x) \leqslant x \,\}$.*

(ii) The greatest postfixed point *of f exists and is $\bigvee \{\, x \in L : x \leqslant f(x) \,\}$.*

Proof. (i) Let $pref(f) := \{\, x \in L : f(x) \leqslant x \,\}$ be the set of prefixed points of f. It suffices to show that $\bigwedge pref(f)$ is a prefixed point. Let $x \in pref(f)$. Then $\bigwedge pref(f) \leq x$. Since f is monotone we have $f(\bigwedge pref(f)) \leq f(x)$, but $f(x) \leq x$ because $x \in pref(f)$. Hence $f(\bigwedge pref(f)) \leq x$ for every $x \in pref(f)$. Since $f(\bigwedge pref(f))$ is an lower bound, we have $f(\bigwedge pref(f)) \leq \bigwedge pref(f)$ as required. (ii) Exercise. $\square$

Construction of fixpoints Let $\langle L, \leqslant \rangle$ be a complete lattice, and $f : L \longrightarrow L$ is monotone. Define, by transfinite induction, a family of elements of L, indexed by *ordinals*:

$$\begin{array}{rcll} f^{\alpha+1} & := & f(f^{\alpha}) & \\ f^{\lambda} & := & \bigvee_{\alpha<\lambda} f^{\alpha} & \text{for } \lambda \text{ a limit ordinal} \end{array}$$

We then set $f^* := \bigvee_{\alpha \in \mathbf{Ord}} f^{\alpha}$.

(Base case — $f^0 = \bot$ — is included in the case for limit ordinal.)

Lemma 4.2. *If $\alpha \leq \beta$ then $f^{\alpha} \leq f^{\beta}$.*

Proof. We prove by transfinite induction on α. Two cases:

(i) $\alpha = \alpha_0 + 1$ is successor ordinal. Two cases of β:

- If $\beta = \beta_0 + 1$ then $f^{\alpha} := f(f^{\alpha_0}) \leq f(f^{\beta_0}) = f^{\beta}$, by monotonicity of f and IH.
- If $\beta = \bigvee_{\beta_0<\beta} \beta_0$, then $\alpha_0 \leq \beta_0$ for some $\beta_0 < \beta$, and so by monotonicity of f and IH

$$f^{\alpha} := f^{\alpha_0+1} \leq f^{\beta_0+1} \leq \bigvee_{\delta<\beta} f^{\delta} = f^{\beta}.$$

(ii) α is a limit ordinal. For each $\alpha_0 < \alpha < \beta$, by IH, $f^{\alpha_0} \leq f^{\beta}$. Because f^{β} is an upper bound, we have

$$f^{\alpha} := \bigvee_{\alpha_0<\alpha} f^{\alpha_0} \leq f^{\beta}.$$

□

Thanks to the Lemma, we have a chain:

$$\bot = f^0 \leq f^1 \leq f^2 \leq \cdots \leq f^n \leq \cdots \leq f^{\omega} \leq \cdots$$

Since **Ord** is a class (not a set), the map $\mathbf{Ord} \longrightarrow L$ defined by $\alpha \mapsto f^{\alpha}$ cannot be injective, and so, the chain must "plateau out" at some point.

The *closure ordinal* of f is defined to be the smallest ordinal κ such that $f^{\kappa} = f^{\kappa+1}$.

Hence $f^* = f^{\kappa}$ where κ is the closure ordinal.

Theorem 4.1 (Knaster-Tarski)**.** *Let $\langle L, \leqslant \rangle$ be a complete lattice. If $f : L \longrightarrow L$ is a monotone function, the* least prefixed point *of f, written $lpr(f)$, is f^*.*

Proof. Recall $f^* := \bigvee_{\alpha \in \mathbf{Ord}} f^{\alpha} = f^{\kappa}$, where κ is the closure ordinal.

"$lpr(f) \leq f^*$": It suffices to prove that f^* is a prefixed point of f. We have $f(f^{\kappa}) = f^{\kappa+1} = f^{\kappa}$.

"$lpr(f) \geq f^*$": We shall prove by transfinite induction that $f^{\alpha} \leq lpr(f)$, for all ordinals α. This is sufficient, since $f^* := \bigvee_{\alpha \in \mathbf{Ord}} f^{\alpha} \leq lpr(f)$. For successor ordinals $\alpha + 1$:

$$\begin{array}{rcll} f^{\alpha+1} & = & f(f^{\alpha}) & \\ & \leq & f(lpr(f)) & \text{induction hypothesis} \\ & \leq & lpr(f) & \text{definition of } lpr(f) \end{array}$$

For limit ordinals λ, we have $f^{\alpha} \leq lpr(f)$ for all $\alpha < \lambda$ by the IH; therefore

$$f^{\lambda} := \bigvee_{\alpha<\lambda} f^{\alpha} \leq lpr(f)$$

□

Exercise 4.1. State and prove a corresponding version of the Theorem for greatest postfixed points.

Lemma 4.3. *$lpr(f)$ exists, and coincides with the* least fixpoint *of f, denoted $lfp(f)$. Similarly greatest fixpoint exists and coincides with greatest postfixed point.*

Proof. By definition $f(lpr(f)) \leq lpr(f)$. Since f monotone, $f(lpr(f))$ is also a prefixed point. Hence $lpr(f) \leq f(lpr(f))$. I.e. $lpr(f)$ is a fixpoint of f. That $lpr(f)$ is the least fixpoint follows from the fact that every fixpoint is also a prefixed point. □

Fixpoints as recursion Given a state transition system (graph) $\langle S, R \subseteq S \times S \rangle$. We give the semantics of a basic state-based modal logic by the mapping

$$\varphi \ \mapsto \ \|\varphi\| \ := \ \{ s \in S : s \vDash \varphi \}$$

I.e. denotation of a formula is an element of $\mathcal{P}(S)$. Hence $\varphi(Z)$, with a free second-order variable Z that ranges over $\mathcal{P}(S)$, can be viewed as a function $f_\varphi : \mathcal{P}(S) \longrightarrow \mathcal{P}(S)$. Recall:

(i) $\langle \mathcal{P}(S), \subseteq \rangle$ is a complete lattice.

(ii) If $f_\varphi : \mathcal{P}(S) \longrightarrow \mathcal{P}(S)$ is monotone, by Knaster-Tarski, f_φ has unique least and greatest fixpoints, denoted μf_φ and νf_φ respectively.

Thus we can extend modal logic by

- *least fixpoint operator*, $\mu Z.\varphi(Z)$, interpreted as μf_φ, and
- *greatest fixpoint operator*, $\nu Z.\varphi(Z)$, interpreted as νf_φ.

Fixpoints give a semantics of recursion, as in domain theory. Recursive modal logic formulas give *succinct* expressions of the usual operators of temporal logic.

4.2 Syntax of the Modal Mu-Calculus

Given

- Var, a set of 2nd-order variables, ranged over by X, Y, Z, etc.
- $Prop$, a set of atomic propositions, ranged over by P, Q, etc.
- $\mathcal{L}$, a set of labels, ranged over by a, b, etc.

modal mu-calculus formulas are defined by the grammar:

$$\varphi \quad ::= \quad P \quad | \quad Z \quad | \quad \varphi_1 \wedge \varphi_2 \quad | \quad [a]\varphi \quad | \quad \neg\varphi \quad | \quad \nu Z.\varphi$$

The *last case*, namely, formation of $\nu Z.\varphi$, is subject to the requirement that each free occurrence of Z in φ be *positive* i.e. in the scope of an even number of negations (so that $\varphi(Z)$ denotes a monotone function in Z).

Notation. If φ is written $\varphi(Z)$, subsequent writing of $\varphi(\psi)$ means "φ with ψ substituted for all free occurrences of Z".

Positive, and positive normal forms *Derived operators*:

$$\begin{aligned}\varphi_1 \vee \varphi_2 &:= \neg(\neg\varphi_1 \wedge \neg\varphi_2)\\ \langle a\rangle\varphi &:= \neg([a]\neg\varphi)\\ \mu Z.\varphi &:= \neg\nu Z.\neg\varphi(\neg Z)\end{aligned}$$

A modal mu-calculus formula is in *positive form* if it is written, using derived operators where necessary, so that $\neg$ is only applied to atomic propositions and free variables. A formula φ is in *positive normal form* if, in addition, all bound variables are distinct. I.e. if $\sigma X.\psi$ and $\sigma' Y.\chi$ are distinct subterms of φ (where $\sigma X.-$ and $\sigma' Y.-$ are fixpoint operators), then $X \neq Y$. E.g. $\mu Z.\neg P \vee \nu Y.(Y \wedge \langle a\rangle Z)$

Operator precedence

$$\begin{array}{c}[a]-\\ \langle a\rangle-\end{array} \quad > \quad \text{Booleans} \quad > \quad \begin{array}{c}\mu Z.-\\ \nu Z.-\end{array}$$

Reading: If $[a]-$ and $\wedge$ contend for a formula, then $[a]-$ wins. For example $[a]\varphi \wedge \psi$ means $([a]\varphi) \wedge \psi$, and $\mu Z.Z \wedge \varphi$ means $\mu Z.(Z \wedge \varphi)$. Thus the scope of a fixpoint extends as far right as possible.

Note that every formula can be converted to positive normal form using *de Morgan laws*, and *α-conversion*: replacing a bound name by a fresh name.

4.3 Labelled Transition Systems

A modal mu-calculus structure over $(\mathit{Prop}, \mathcal{L})$ is a *labelled transition system* (LTS), namely, a triple $T = \langle S, \rightarrow, \rho \rangle$ with

- a set S of states
- a transition relation $\rightarrow \subseteq S \times \mathcal{L} \times S$ (as usual we write $s \xrightarrow{a} t$ to mean $(s, a, t) \in \rightarrow$)
- a function $\rho : \mathit{Prop} \longrightarrow \mathcal{P}(S)$ interpreting the atomic propositions.

Observe that an LTS is just a directed graph, whose edges are labelled by elements of $\mathcal{L}$, and vertices are labelled by elements of $\mathcal{P}(\mathit{Prop})$ i.e. each $x \in S$ is labelled by $\{ P \in \mathit{Prop} : x \in \rho(P) \}$.

Definition 4.1. Given an LTS T, and a *valuation* (or *assignment*) $V : Var \longrightarrow \mathcal{P}(S)$, we define:

$$\begin{aligned}\|P\|^T_V &:= \rho(P)\\ \|Z\|^T_V &:= V(Z)\\ \|\neg\varphi\|^T_V &:= S \setminus \|\varphi\|^T_V\\ \|\varphi_1 \wedge \varphi_2\|^T_V &:= \|\varphi_1\|^T_V \cap \|\varphi_2\|^T_V\\ \|[a]\varphi\|^T_V &:= \{ s \mid \forall t \in S \,.\, s \xrightarrow{a} t \implies t \in \|\varphi\|^T_V \}\\ \|\nu Z.\varphi\|^T_V &:= \bigcup\{ U \subseteq S \mid U \subseteq \|\varphi\|^T_{V[Z\mapsto U]} \}\end{aligned}$$

where the valuation $V[Z \mapsto U]$ is defined by:

$$V[Z \mapsto U](X) := \begin{cases} U & \text{if } X = Z\\ V(X) & \text{if } X \neq Z.\end{cases}$$

N.B. $\bigcup\{ U_i \subseteq S : i \in I \} := \{ x \in S : x \in U_i \text{ for some } i \in \mathcal{I} \}$

Remark 4.1. (i) Generally let $K \subseteq \mathcal{L}$, define

$$\|[K]\varphi\|^T_V \;:=\; \{\, s \mid \forall a \in K \,.\, \forall t \in S \,.\, s \xrightarrow{a} t \;\implies\; t \in \|\varphi\|^T_V \,\}$$

Write $[-]\varphi$ to mean $[\mathcal{L}]\varphi$, similarly for $\langle - \rangle\varphi$.

(ii) Equivalently we can define

$$\|\nu Z.\varphi\|^T_V \;:=\; \mathit{gfp}(f_{\varphi,Z,V})$$

where $f_{\varphi,Z,V} : \mathcal{P}(S) \longrightarrow \mathcal{P}(S)$ is the function $U \mapsto \|\varphi\|^T_{V[Z\mapsto U]}$.
Note that $f_{\varphi,Z,V}$ is monotone. By two lemmas (greatest fixpoint equals greatest postfixed point, which is the supremum of all postfixed points), we have

$$\begin{aligned} \mathit{gfp}(f_{\varphi,Z,V}) &= \bigvee\{\, U \in \mathcal{P}(S) : U \le f_{\varphi,Z,V}(U) \,\} \\ &= \bigcup\{\, U \subseteq S : U \subseteq \|\varphi\|^T_{V[Z\mapsto U]} \,\}. \end{aligned}$$

Exercise 4.2. Prove the following.

$$\begin{aligned} \|\varphi_1 \vee \varphi_2\|^T_V &= \|\varphi_1\|^T_V \cup \|\varphi_2\|^T_V \\ \|\langle a\rangle\varphi\|^T_V &= \{\, s \mid \exists t \in S \,.\, s \xrightarrow{a} t \;\wedge\; t \in \|\varphi\|^T_V \,\} \\ \|\mu Z.\varphi\|^T_V &= \bigcap\{\, U \subseteq S \mid \|\varphi\|^T_{V[Z\mapsto U]} \subseteq U \,\} \\ &= \bigwedge\{\, U \in \mathcal{P}(S) : f_{\varphi,Z,V}(U) \le U \,\} = \mathit{lfp}(f_{\varphi,Z,V}) \end{aligned}$$

Note that $\bigcap\{\, U_i \subseteq S : i \in I \,\} := \{\, x \in S : x \in U_i$ for every $i \in \mathcal{I} \,\}$

Since $\mu Z.\varphi = \neg\nu Z.\neg\varphi(\neg Z)$, we have

$$\begin{aligned} \|\mu X.\varphi\|^T_V &= \overline{\bigcup\{\, U : U \subseteq \|\neg\varphi(\neg Z)\|^T_{V[Z\mapsto U]} \,\}} \\ &= \bigcap\{\, \overline{U} : U \subseteq \|\neg\varphi(Z)\|^T_{V[Z\mapsto \overline{U}]} \,\} \\ &= \bigcap\{\, \overline{U} : \|\varphi(Z)\|^T_{V[Z\mapsto \overline{U}]} \subseteq \overline{U} \,\} \\ &= \bigcap\{\, U : \|\varphi(Z)\|^T_{V[Z\mapsto U]} \subseteq U \,\} \end{aligned}$$

Notations

(i) We sometimes write $s \vDash^T_V \varphi \;:=\; s \in \|\varphi\|^T_V$.
In case T is understood, and φ is closed, we simply write $s \vDash \varphi$.

(ii) We often write $\mathsf{t} := \nu Z.Z$ and $\mathsf{f} := \neg\mathsf{t}$.
What are $\|\nu Z.Z\|^T_V$ and $\mathsf{f} \equiv \mu Z.Z$?

(iii) For closed formulas φ, and ψ, we write $\varphi \equiv \psi$ to mean "for all LTS T, $\|\varphi\|^T_\varnothing = \|\psi\|^T_\varnothing$".

4.4 Syntactic Approximants Using Infinitary Syntax

The fixpoint formulas of the modal mu-calculus denote fixpoints of monotone functions in a complete lattice. When reasoning about fixpoints, it is convenient to introduce a notation for approximants of these fixpoints. To this end, we introduce an infinitary syntax.

Let λ range over limit ordinals

$$\begin{aligned} \mu^0 Z.\varphi(Z) &:= \mathsf{f} \\ \mu^{\alpha+1} Z.\varphi(Z) &:= \varphi(\mu^\alpha Z.\varphi(Z)) \\ \mu^\lambda Z.\varphi(Z) &:= \textstyle\bigvee_{\alpha<\lambda} \mu^\alpha Z.\varphi(Z) \end{aligned}$$

The semantics of these infinitary terms are defined as:

$$\begin{array}{rcl} \|\mu^{\alpha+1}Z.\varphi(Z)\|_V^T & := & \|\varphi(Z)\|^T_{V[Z\mapsto\|\mu^\alpha Z.\varphi(Z)\|_V^T]} \\ \|\mu^{\lambda}Z.\varphi(Z)\|_V^T & := & \bigvee_{\alpha<\lambda}\|\mu^\alpha Z.\varphi(Z)\|_V^T \end{array}$$

Thus $\|\mu Z.\varphi(Z)\|_V^T = \|\mu^\kappa Z.\varphi(Z)\|_V^T$ where κ is the closure ordinal. If $\|\mu^\alpha Z.\varphi(Z)\|_V^T = \|\mu^{\alpha+1}Z.\varphi(Z)\|_V^T$ then $\|\mu^\alpha Z.\varphi(Z)\|_V^T = \|\mu Z.\varphi(Z)\|_V^T$.

Similarly for approximants of the greatest fixpoints:

$$\begin{array}{rcl} \nu^0 Z.\varphi(Z) & := & \mathsf{t} \\ \nu^{\alpha+1}Z.\varphi(Z) & := & \varphi(\nu^\alpha Z.\varphi(Z)) \\ \nu^{\lambda}Z.\varphi(Z) & := & \bigwedge_{\alpha<\lambda}\nu^\alpha Z.\varphi(Z) \end{array}$$

and

$$\begin{array}{rcl} \|\nu^{\alpha+1}Z.\varphi(Z)\|_V^T & := & \|\varphi(Z)\|^T_{V[Z\mapsto\|\nu^\alpha Z.\varphi(Z)\|_V^T]} \\ \|\nu^{\lambda}Z.\varphi(Z)\|_V^T & := & \bigwedge_{\alpha<\lambda}\|\nu^\alpha Z.\varphi(Z)\|_V^T \end{array}$$

Note that the infinitary terms $\nu^{\alpha+1}Z.\varphi(Z), \nu^\lambda Z.\varphi(Z), \mu^{\alpha+1}Z.\varphi(Z)$, etc. are *not* part of the modal mu-calculus. They are notations denoting subsets of the state-set that are useful for calculations.

Lemma 4.4 (Approximation)**.** *Let $T = \langle S, \rightarrow, \rho\rangle$ be an LTS. For any $s \in S$, we have*

(i) $s \in \|\mu Z.\varphi(Z)\|_V^T$ iff $s \in \|\mu^\alpha Z.\varphi(Z)\|_V^T$ for some α.

(ii) $s \in \|\nu Z.\varphi(Z)\|_V^T$ iff $s \in \|\nu^\alpha Z.\varphi(Z)\|_V^T$ for all α.

Proof. (i) We have $\|\mu Z.\varphi(Z)\|_V^T = \mathit{lfp}(f_{\varphi,Z,V})$ by definition. Note that for each ordinal α, $\|\mu^\alpha Z.\varphi(Z)\|_V^T$ coincides with $f^\alpha_{\varphi,Z,V}$ i.e. the element of $\mathcal{P}(S)$ in the chain $f^0_{\varphi,Z,V}, f^1_{\varphi,Z,V}, \cdots$ indexed by α. Hence, by Knaster-Taski

$$\|\mu Z.\varphi(Z)\|_V^T = \bigvee_{\alpha\in\mathbf{Ord}} f^\alpha_{\varphi,Z,V} = \bigcup_{\alpha\in\mathbf{Ord}}\|\mu^\alpha Z.\varphi(Z)\|_V^T.$$

(ii) Exercise

□

Lemma 4.5. *Let $T = \langle S, \rightarrow, \rho\rangle$ be an LTS. For any $s \in S$, we have*

(i) If $s \in \|\mu Z.\varphi(Z)\|_V^T$ then there is a least ordinal α such that $s \in \|\mu^\alpha Z.\varphi(Z)\|_V^T$ but $s \notin \|\mu^\beta Z.\varphi(Z)\|_V^T$ for all $\beta < \alpha$.

(ii) If $s \notin \|\nu Z.\varphi(Z)\|_V^T$ then there is a least ordinal α such that $s \notin \|\nu^\alpha Z.\varphi(Z)\|_V^T$ but $s \in \|\mu^\alpha Z.\varphi(Z)\|_V^T$ for all $\beta < \alpha$.

Hint for (3) and (4). Make use of the fact that the class of ordinals is *well-founded*: A binary relation, R, is *well-founded* on a class X just if every non-empty subset of X has a minimal element with respect to R. Equivalently, there is no infinite descending R-chain of elements from X. □

4.5 Intuitions from Examples

Correctness properties of reactive systems are often classified into safety or liveness properties. Intuitively

- *safety properties* say that "something bad will never happen"
- *liveness properties* say that "something good will eventually happen".

Useful Slogans

1. "ν is looping, whereas μ is *finite* looping"
2. "μ is liveness and ν is safety".

Example 4.3 ("ν is looping")**.** (i) $\nu Z.P \wedge [a]Z$ relativized 'always' formula. "P is true along every a-path".

(ii) $\nu Z.Q \vee (P \wedge [a]Z)$ relativized 'while' formula. "On every a-path, P holds while Q fails". The formula is true if either Q holds, or P holds and wherever we go next (via a), the formula is true, and .... In particular, if P is always true, and Q never holds, the formula is true. *Cf.* $\mu Z.Q \vee (P \wedge \langle a\rangle Z)$—next slide.

Mu-formulas require something to happen (i.e. to exit the loop), and are thus liveness properties.

Example 4.4 ("μ is finite looping")**.** (i) $\mu Z.P \vee [a]Z$: "On all a-paths, P eventually holds".

(ii) $\mu Z.Q \vee (P \wedge \langle a\rangle Z)$: "On some a-path, P holds until Q holds (and Q *must* eventually hold)." I.e. we are not allowed to repeat the unfolding forever, so we must eventually "bottom out" through the Q disjunct.

Example 4.5. Consider the simple transition system

$$\begin{tikzcd} T_1 = & s \arrow[loop below, "a"] \end{tikzcd}$$

with state-set $S = \{ s \}$ and a single transition $s \xrightarrow{a} s$. We have $s \models \nu Z.(\langle a\rangle Z \vee [a]\mathsf{f})$. *Intuitively* this is because from s it is always possible to do an a-transition and then an a-transition and then an a-transition ... *ad infinitum.*

Example 4.6. Take T_1 as before. We show that $s \not\models \mu Z.(\langle a\rangle Z \vee [a]\mathsf{f})$. If s satisfies $\mu Z.(\langle a\rangle Z \vee [a]\mathsf{f})$ to hold, then it must be possible for s to do a finite[1] number of a-transitions and then satisfy $[a]\mathsf{f}$, but the latter is impossible since there is always an a-transition from s. I.e. s can never satisfy $\underbrace{\langle\, a\,\rangle(\cdots(\langle\, a\,\rangle}_{\text{finite times}}([a]\mathsf{f})))$. Now we calculate. Note that $\mu^1 Z.(\langle a\rangle Z \vee [a]\mathsf{f}) = (\langle a\rangle\mathsf{f} \vee [a]\mathsf{f})$. Now $\|\langle a\rangle\mathsf{f}\| = \varnothing$ and $\|[a]\mathsf{f}\| = \varnothing$. So $\|\mu^1 Z.(\langle a\rangle Z \vee [a]\mathsf{f})\| = \varnothing$. Similarly $\|\mu^2 Z.(\langle a\rangle Z \vee [a]\mathsf{f})\| = \varnothing$. Hence $\|\mu Z.(\langle a\rangle Z \vee [a]\mathsf{f})\| = \varnothing$

Alternatively consider the function

$$\begin{array}{rcl} f \;:\; U & \mapsto & \|\langle a\rangle Z \vee [a]\mathsf{f}\|^T_{\varnothing[Z\mapsto U]} \\ & = & \|\langle a\rangle Z\|^T_{\varnothing[Z\mapsto U]} \cup \|[a]\mathsf{f}\| \\ & = & U \cup \varnothing \end{array}$$

which is the identity map on $\mathcal{P}(S)$. Hence $\|\mu Z.(\langle a\rangle Z \vee [a]\mathsf{f})\| = \mathit{lfp}(f) = \varnothing$.

[1]Because S is finite, the closure ordinal of any monotone function on $\mathcal{P}(S)$ is finite.

Example 4.7. Take the transition system

$$T_2 \;=\; \begin{tikzcd} 0 \arrow[r, bend left, "a"] & 1 \arrow[l, bend left, "a"] \arrow[r, "b"] & 2 \end{tikzcd}$$

with state-set $S = \{0,1,2\}$. We show that $0 \vDash \nu Z.\mu Y.[a]((\langle b\rangle \mathsf{t} \wedge Z) \vee Y)$. First note $\|\langle b\rangle \mathsf{t}\| = \{1\}$. Set

$$F(U) \;=\; \|\mu Y.[a]((\langle b\rangle \mathsf{t} \wedge Z) \vee Y)\|_{[Z\mapsto U]}.$$

W.t.p. $0 \in \mathit{gfp}(F)$. By abuse of notation, we have

$$F\{0,1,2\} = F\{0,1\} = F\{1\} = \|\mu Y.[a](\{1\} \vee Y)\|.$$

Intuitively $s \in \|\mu Y.[a](\{1\} \vee Y)\|$ if and only if every a-labelled path from s reaches 1. What is $\|\mu Y.[a](\{1\} \vee Y)\|$? We have

$$\begin{aligned} \|\mu^1 Y.[a](\{1\} \vee Y)\| &= \{0,2\} \\ \|\mu^2 Y.[a](\{1\} \vee Y)\| &= [a](\{1\} \vee \{0,2\}) = \{0,1,2\} \\ \|\mu^3 Y.[a](\{1\} \vee Y)\| &= [a](\{1\} \vee \{0,1,2\}) = \{0,1,2\}. \end{aligned}$$

I.e. $F\{0,1,2\} = \{0,1,2\}$. Hence $\mathit{gfp}(F) = \{0,1,2\}$.

4.6 Alternation Depth Hierarchy

Consider the following modal mu-calculus formulas.

- $\mu X.\varphi \vee [-]X$: all paths eventually satisfy φ
- $\nu Y.[\mu Z.P \vee \langle -\rangle Z] \wedge \langle -\rangle Y$: there exists an infinite path along which P is always reachable
- $\nu Y \mu Z.(P \vee \langle -\rangle Z) \wedge \langle -\rangle Y$: there exists a path along which P holds infinitely often

The fixed point quantifiers provide great expressive power. Liveness and safety can be expressed readily and by allowing more nested fixpoint quantifiers (depth) we can express complex fairness constraints. Thus it natural to define a measure of expressiveness in this term.

The *alternation depth* of a formula is the maximum number of μ/ν alternations in a chain of nested fixed points. The *simple hierarchy* counts the syntactic alternation. The *Niwiński hierarchy* considers genuine dependency between the fixed points.

Formally the Niwiński hierarchy is defined as follows. A formula φ is in the classes Π^μ_0 and Σ^μ_0 if it contains no fixpoint operators i.e. it is a formula of modal logic. The class Σ^μ_{n+1} is the closure of $\Sigma^\mu_n \cup \Pi^\mu_n$ under the following rules.

- If $\varphi, \psi \in \Sigma^\mu_{n+1}$, then $\varphi \wedge \psi, \varphi \vee \psi, [a]\varphi, \langle a\rangle\varphi \in \Sigma^\mu_{n+1}$.
- If $\varphi \in \Sigma^\mu_{n+1}$ and X positive in φ, then $\mu X.\varphi \in \Sigma^\mu_{n+1}$.
- If $\varphi(X), \psi \in \Sigma^\mu_{n+1}$, then $\varphi(\psi) \in \Sigma^\mu_{n+1}$, provided no free variable of ψ becomes bound by a fixpoint quantifier in φ.

$$\begin{tikzcd}
 & \Sigma_1^\mu \arrow[r, dash] \arrow[ddr, dash] & \Sigma_2^\mu \arrow[r, dash] \arrow[ddr, dash] & \Sigma_3^\mu \arrow[r, dotted, dash] & \Sigma_k^\mu \arrow[r, dash] \arrow[ddr, dash] & \Sigma_{k+1}^\mu \arrow[r, dotted, dash] & {} \\
\Sigma_0^\mu = \Pi_0^\mu \arrow[ur, dash] \arrow[dr, dash] & & & & & & \\
 & \Pi_1^\mu \arrow[r, dash] \arrow[uur, dash] & \Pi_2^\mu \arrow[r, dash] \arrow[uur, dash] & \Pi_3^\mu \arrow[r, dotted, dash] & \Pi_k^\mu \arrow[r, dash] \arrow[uur, dash] & \Pi_{k+1}^\mu \arrow[r, dotted, dash] & {}
\end{tikzcd}$$

Figure 4.1: Alternation-depth hierarchy.

The class Π^μ_{n+1} is defined analogously.

It is known that indeed arbitrary alternation is necessary to capture all expressible properties, in other words the modal mu-calculus alternation hierarchy is strict (Bradfield 1996).

There is a price to pay for this expressive power and that is complexity. The most notable open problem in modal mu-calculus is the complexity of model checking.

Model Checking Problem: *Given an LTS T, a state s, and a closed modal mu-calculus formula φ, does $s \in \|\varphi\|^T_\varnothing$ hold?*

The best algorithms to date are all essentially exponential in the depth of the formula φ and the best known bound is **NP** $\cap$ **co-NP**. The problem is conjectured to be in **P**. The harder problem of satisfiability is known to be **EXPTIME**-complete.

Satisfiability Problem: *Given a closed modal mu-calculus formula φ, is it satisfiable?* I.e. is there an LTS T with a state s such that $s \in \|\varphi\|^T_\varnothing$?

Although modal mu-calculus is a decidable logic, it is too expressive for real use. Its importance comes mainly from providing a meta-language to establish results about other temporal logics. Most temporal logics such as LTL and CTL and their extension can be interpreted into modal mu-calculus using two nested quantifiers.

4.7 An Interlude: Computational Tree Logic (CTL)

$$\varphi \ ::= \ P \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid [K]\varphi \mid \mathbf{A}(\varphi\,\mathbf{U}\,\psi) \mid \mathbf{E}(\varphi\,\mathbf{U}\,\psi)$$

where $P \in \mathit{Prop}$ and $K \subseteq \mathcal{L}$.

CTL is a *branching time* logic; LTL is a linear-time logic.

Semantics of CTL We interpret CTL formulas in the same structures as modal mu-calculus. Let $T = \langle\, S, \longrightarrow\, \subseteq S \times \mathcal{L} \times S, \rho : \mathit{Prop} \longrightarrow 2^S \,\rangle$ be an LTS.

$$\begin{array}{rcl}
s \vDash P & := & s \in \rho(P) \\
s \vDash \neg\varphi & := & s \nvDash \varphi \\
s \vDash \varphi_1 \wedge \varphi_2 & := & s \vDash \varphi_1 \text{ and } s \vDash \varphi_2 \\
s \vDash [K]\varphi & := & \text{for all finite or infinite runs } s \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \cdots, \\
 & & \text{if } a_1 \in K, \text{ then } s_1 \vDash \varphi. \\
s \vDash \mathbf{A}(\varphi\,\mathbf{U}\,\psi) & := & \text{for all finite or infinite runs } s = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \cdots, \\
 & & \text{there is } i \geq 0 \text{ with } s_i \vDash \psi \text{ and for all } 0 \leq j < i,\ s_j \vDash \varphi \\
s \vDash \mathbf{E}(\varphi\,\mathbf{U}\,\psi) & := & \text{for some finite or infinite runs } s = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \cdots, \\
 & & \text{there is } i \geq 0 \text{ with } s_i \vDash \psi \text{ and for all } 0 \leq j < i,\ s_j \vDash \varphi
\end{array}$$

In LTL, we write $\mathbf{F}\,\varphi := \mathsf{t}\,\mathbf{U}\,\varphi$ and $\mathbf{G}\,\varphi := \neg(\mathbf{F}\,\neg\varphi)$. In CTL, we write $\mathbf{A}\,\mathbf{F}\,\varphi := \mathbf{A}(\mathsf{t}\,\mathbf{U}\,\varphi)$ and $\mathbf{A}\,\mathbf{G}\,\varphi := \neg\mathbf{E}(\mathsf{t}\,\mathbf{U}\,\neg\varphi)$; similarly for $\mathbf{E}\,\mathbf{F}\,\varphi$ and $\mathbf{E}\,\mathbf{G}\,\varphi$.

Example 4.8. (i) $s \vDash \mathbf{E}\,\mathbf{F}\,P$. There is a path from s on which P is eventually true.

(ii) $s \vDash \mathbf{A}\,\mathbf{F}\,\mathbf{E}\,\mathbf{G}\,P$. On every path from s, there is a state from which there is a path where P holds everywhere.

(iii) $s \vDash \mathbf{E}\,\mathbf{G}\,\mathbf{A}\,\mathbf{F}\,P$. There is a path from s such that every state t on it satisfies the property that on every path from t, P is eventually true.

Example 4.9. What does the mu-formula $\mu Z.[\mathcal{L}]Z$ mean? We use approximants to analyse the formula.

- $\mu^0 Z.[\mathcal{L}]Z := \varnothing$
- $\mu^1 Z.[\mathcal{L}]Z := [\mathcal{L}]\varnothing$, which is the set of terminal or "deadlocked" states.
- $\cdots$
- $\mu^{i+1} Z.[\mathcal{L}]Z := [\mathcal{L}]\cdots[\mathcal{L}]\varnothing$ ($i+1$ nested boxes), which is the set of states s such that every path from s has length at most i and necessarily ends at a "deadlocked" state.

Hence $\mu Z.[\mathcal{L}]Z$ describes the set of states s of an LTS such that every path from s necessarily ends in a deadlocked state. In fact we have $\mu Z.[\mathcal{L}]Z \equiv \mathbf{A}\,\mathbf{F}\,([\mathcal{L}]\mathsf{f})$.

Example 4.10. *CTL formula* $\forall\,\boldsymbol{G}\,\varphi$: "for every path (or run), φ always holds on it". It is the property X *such that* φ *is true now, and for every successor,* X *remains true.* I.e. X satisfies the modal equation:

$$X \;=\; \varphi \wedge [-]X$$

where $[-]X$ means "X is true at every successor".

Which fixpoint? If a state satisfies any solution of the equation then surely it satisfies $\forall\,\boldsymbol{G}\,\varphi$. So the "equation" should be

$$X \;\Rightarrow\; \varphi \wedge [-]X$$

or more precisely, $\|X\| \;\subseteq\; \|\varphi \wedge [-]X\|$. Thus the meaning is the greatest postfixed point $\nu X.\varphi \wedge [-]X$.

Example 4.11. *CTL formula* $\exists\,\boldsymbol{F}\varphi$: "there exists a path on which φ eventually holds". It is the property Y *such that either* φ *holds now, or there is some successor on which* Y *is true.* I.e. Y satisfies the modal equation:

$$Y \;=\; \varphi \vee \langle - \rangle Y$$

where $\langle - \rangle Y$ means "Y is true at some successor".

We can argue: if a state satisfies $\exists\,\boldsymbol{F}\varphi$, then it surely satisfies any solution of the equation; and so, we want the least solution of

$$Y \;\Leftarrow\; \varphi \vee \langle - \rangle Y$$

or more precisely $\|Y\| \;\supseteq\; \|\varphi \vee \langle - \rangle Y\|$. I.e. the meaning is the least prefixed point $\mu Y.\varphi \vee \langle - \rangle Y$.

Problems

4.1 Let φ be a monotone function on the powerset of S. Let $U \subseteq S$. Prove

$$U \subseteq gfp(\varphi) \quad \Leftrightarrow \quad U \subseteq \varphi(gfp(\varphi^U))$$

where $\varphi^U : \mathcal{P}(S) \to \mathcal{P}(S)$ is the function given by $V \mapsto U \cup \varphi(V)$.

4.2 Prove the following:

(a) If $s \in \|\mu Z.\varphi\|_V^T$ then there is a least ordinal α such that $s \in \|\mu^\alpha Z.\varphi\|_V^T$, and for all $\beta < \alpha$, $s \notin \|\mu^\beta Z.\varphi\|_V^T$.

(b) If $s \notin \|\nu Z.\varphi\|_V^T$ then there is a least ordinal α such that $s \notin \|\nu^\alpha Z.\varphi\|_V^T$, and for all $\beta < \alpha$, $s \in \|\nu^\beta Z.\varphi\|_V^T$.

4.3 Prove that if the state-set S has n elements, then for any $s \in S$

(a) $s \models_V \nu Z.\varphi$ iff $s \in \|\nu^n Z.\varphi\|_V^T$

(b) $s \models_V \mu Z.\varphi$ iff $s \in \|\mu^n Z.\varphi\|_V^T$

4.4 We say that φ and ψ are *equivalent* just if for any T and for any V, we have $\|\varphi\|_V^T = \|\psi\|_V^T$. Prove

(a) $\mu Z.\varphi(Z)$ and $\varphi(\mu Z.\varphi)$ are equivalent (similarly $\nu Z.\varphi$ and $\varphi(\nu Z.\varphi)$ are equivalent).

(b) $\mu Z.\varphi(Z)$ and $\neg\nu Z.\neg\varphi(\neg Z)$ are equivalent.

4.5 Is there a labelled transition system T and a valuation V such that for every modal mu-calculus formula φ, $\|\mu Z.\varphi\|_V^T = \|\nu Z.\varphi\|_V^T$?

4.6

(a) Show that $s_0 \in \|\mu Z.[a]Z\|_V^T$ iff there is no infinite transition sequence

$$s_0 \xrightarrow{a} s_1 \xrightarrow{a} s_2 \xrightarrow{a} \cdots$$

(b) What is the meaning of $\nu Z.[a]Z$?

4.7 Consider the transition system

$$T_2 \;=\; \begin{tikzcd} 0 \arrow[r, bend left, "a"] & 1 \arrow[l, bend left, "a"] \arrow[r, "b"] & 2 \end{tikzcd}$$

with state-set $S = \{0, 1, 2\}$.

(a) Compute $\|\mu Y.\nu Z.[a]((\langle b\rangle \mathsf{t} \vee Y) \wedge Z)\|$.

(b) Prove or disprove the following by drawing the game graph and argue by the existence (or not) of a winning strategy.

   i. $0 \models \mu Z.\nu Y.[a]((\langle b\rangle \mathsf{t} \vee Y) \wedge Z)$.
   ii. $2 \models \mu Z.\nu Y.[a]((\langle b\rangle \mathsf{t} \vee Y) \wedge Z)$.

4.8 We say that variable Z in a formula φ is *guarded* if every occurrence of Z in φ is in the scope of some modal operator $[a]-$ or $\langle a\rangle-$. We say that a formula is *guarded* if for every subformula $\sigma Z.\psi$ of φ, Z is guarded in ψ (where $\sigma = \mu, \nu$). Prove *Kozen's Lemma*: Every modal mu-calculus formula is equivalent to some positive guarded formula.

4.9 What properties are expressed by the following modal mu-calculus formulas?

(a) $\mu Z.[\mathcal{L}]Z$

(b) $\nu Z.\langle a\rangle Z \wedge \langle b\rangle Z$

4.10 Fix a set $\mathcal{L}$ of labels. Express the following in modal mu-calculus formulas.

(a) Along some path P is always true

(b) There is a path along which P holds continuously and Q holds infinitely often.

Chapter 5

Games and Tableaux for Modal Mu-Calculus

[*We gratefully acknowledge Bahareh Afshari's contribution to the section on tableaux for modal mu-calculus.*]

Synopsis

Modal mu-calculus model checking games. Streett and Emerson's Fundamental Semantic Theorem. 2-person infinite games. Memoryless winning strategies. Signature and signature decrease lemma. Tableaux. Tree model property. Finite model property. Parity games. Computing winning regions. PARITY is in $\mathbf{NP} \cap \text{co-}\mathbf{NP}$. Determinacy for finite parity games.

References (Streett and Emerson, 1989; Bradfield and Stirling, 2007; Stirling, 1997, 2001; Niwinski and Walukiewicz, 1997)

5.1 Game Characterisation of Model Checking

Modal Mu-Calculus Model Checking Problem Given a state s_0 of a labelled transition system T, a valuation V, and a mu-calculus formula φ, does $s_0 \vDash_V^T \varphi$ hold?.

We aim to give a game characterisation of the problem. We shall consider games played by players V (Verifier) and R (Refuter) on directed graphs of a certain kind. Given a state s_0 of a transition system T, a valuation V and a modal mu-calculus formula φ, we define a game between V and R, $\mathcal{G}_V^T(s_0, \varphi)$, such that $s_0 \vDash_V^T \varphi$ if, and only if, there is a winning (memoryless) strategy for V in $\mathcal{G}_V^T(s_0, \varphi)$.

The characterisation may be viewed as a version of the Fundamental Semantic Theorem of Streett and Emerson (Streett and Emerson, 1989).

Some preliminaries on syntax We assume that φ is *positive normal* i.e.

(i) Negation is only applied to atomic propositions and free variables.

(ii) If $\sigma_1 Z_1$ and $\sigma_2 Z_2$ are two different occurrences of binders in φ, then $Z_1 \neq Z_2$.

(iii) Free variables are disjoint from bound variables.

Every formula can be converted into positive normal form using *de Morgan laws* and by renaming bound variables. For example

$$\mu Z.\langle J\rangle Y \vee (\langle b\rangle Z \wedge \mu Y.\nu Z.([b]Y \wedge [K]Z))$$

can be rewritten

$$\mu Z.\langle J\rangle Y \vee (\langle b\rangle Z \wedge \mu X.\nu U.([b]X \wedge [K]U))$$

Let $Sub(\varphi)$ be the set of subformulas of φ. For example, $Sub(\mu X.\nu Y.([b]X \wedge [K]Y))$ is the set

$$\{\ \mu X.\nu Y.([b]X \wedge [K]Y),\ \nu Y.[b]X \wedge [K]Y,\ [b]X \wedge [K]Y,\ [b]X,\ [K]Y,\ X,\ Y\ \}$$

If φ is (positive) normal and $\sigma Z.\psi \in Sub(\varphi)$, then the (bound) variable Z can be used to identify this subformula.

For $\sigma_1 X_1.\psi_1, \sigma_2 X_2.\psi_2 \in Sub(\varphi)$, we say that X_1 *subsumes* X_2, written $X_1 \succcurlyeq X_2$, just if $\sigma_2 X_2.\psi_2 \in Sub(\sigma_1 X_1.\psi_1)$. In the example above, X subsumes Y (but not vice versa).

Lemma 5.1. *Assume a positive normal φ.*

(i) If X subsumes Y and Y subsumes Z, then X subsumes Z.

(ii) If X subsumes Y and $X \neq Y$, then it is not the case that Y subsumes X.

Definition 5.1 (Model Checking Game $\mathcal{G}^T_V(s,\varphi)$)**.** Fix a transition system $T = \langle S, \to \subseteq S \times \mathcal{L} \times S, \rho : Prop \to \mathcal{P}(S)\rangle$, a state $s_0 \in S$, a valuation V, and a positive normal φ. The two players are Refuter (R) and Verifier (V).

- R attempts to show $s_0 \not\models^T_V \varphi$, whereas
- V attempts to show $s_0 \models^T_V \varphi$.

We define the graph underlying $\mathcal{G}^T_V(s_0,\varphi)$. The vertices (also called *positions*, or *moves*) are pairs (t,ψ) where $t \in S$ and $\psi \in Sub(\varphi)$. The initial vertex is (s_0,φ). Edges are organised into three groups according to the shape of the subformula.

- *Boolean subformulas.* For each $\psi_1 \vee \psi_2, \psi_1 \wedge \psi_2 \in Sub(\varphi)$, $s \in S$, $i \in \{1,2\}$, the following are edges:

$$(s,\psi_1 \vee \psi_2) \to (s,\psi_i), \qquad (s,\psi_1 \wedge \psi_2) \to (s,\psi_i)$$

- *Modal subformulas.* For each $K \subseteq \mathcal{L}$, $a \in K$, and each state t where $s \xrightarrow{a} t$, the following are edges:

$$(s,[K]\psi) \to (t,\psi), \qquad (s,\langle K\rangle\psi) \to (t,\psi)$$

- *Fixpoint subformulas.* For each $\sigma Z.\psi \in Sub(\varphi)$, each $s \in S$, the following are edges:

$$(s,\sigma Z.\psi) \to (s,Z), \qquad (s,Z) \to (s,\psi)$$

Observe that the size of ψ decreases in $(s,\psi) \to (s',\psi')$ in all cases except when ψ is a fixpoint variable.

Plays There are no out-going edges from (s,ψ) where ψ is an atomic proposition (P or $\neg P$), or *free* variable Z (i.e. does not identify any fixpoint subformula). Certain vertices are "owned" by one of the two players; as for the rest of the variables (all of which have out-degree 1), it is unimportant who owns them.

V	R	Ownership is irrelevant (say, V)
$(s, \psi_1 \vee \psi_2)$	$(s, \psi_1 \wedge \psi_2)$	
$(s, \langle K \rangle \psi)$	$(s, [K]\psi)$	(s, Z) for bound Z
(s, P), $s \notin \rho(P)$	(s, P), $s \in \rho(P)$	$(s, \sigma Z.\psi)$
(s, Z), $s \notin V(Z)$	(s, Z), $s \in V(Z)$	

A *play* of $\mathcal{G}^T_V(s_0, \varphi)$ is a path in the game graph that starts from the initial vertex (s_0, φ), namely, $(s_0, \varphi), (s_1, \varphi_1), (s_2, \varphi_2), \cdots$, such that for each i, if (s_i, φ_i) has label V (resp. R), then V (resp. R) chooses (s_{i+1}, φ_{i+1}).

Example 5.1. Take the transition system with state-set $S = \{ s \}$ with $\mathcal{L} = \{ a, b \}$, and a transition $s \xrightarrow{a} s$.

$$T_1 \;=\; \begin{tikzcd} s \arrow[loop below, "a"] \end{tikzcd}$$

The game $\mathcal{G}^{T_1}_{\varnothing}(s, \nu Z.[b]\mathsf{f} \wedge \langle a \rangle Z)$ has the following game graph:

$$\begin{tikzcd}
 & & (s, \nu Z.[b]\mathsf{f} \wedge \langle a \rangle Z) \arrow[d] & & \\
 & & (s, Z) \arrow[d] & & \\
 & & (s, [b]\mathsf{f} \wedge \langle a \rangle Z) \arrow[dl] \arrow[dr] & \mathrm{R} & \\
\mathrm{R} & (s, [b]\mathsf{f}) & & (s, \langle a \rangle Z) \arrow[uul, bend right] & \mathrm{V}
\end{tikzcd}$$

Winning conditions R wins a play just if

(i) The play is $(s_0, \varphi_0), (s_1, \varphi_1), \cdots, (s_n, \varphi_n)$ and

a. $\varphi_n = P$ and $s_n \notin \rho(P)$ or

b. $\varphi_n = Z$ and Z is free in φ_0 and $s_n \notin V(Z)$, or

c. $\varphi_n = \langle K \rangle \psi$ and $\{ t : s_n \xrightarrow{a} t \text{ and } a \in K \} = \varnothing$

(ii) The play $(s_0, \varphi_0), (s_1, \varphi_1), \cdots, (s_n, \varphi_n), \cdots$ is infinite, and the *unique* fixed point variable X, which occurs infinitely often and which subsumes all other variables occuring infinitely often, identifies a least fixpoint subformula of φ.

V wins a play just if

(i) The play is $(s_0, \varphi_0), (s_1, \varphi_1), \cdots, (s_n, \varphi_n)$ and

a. $\varphi_n = P$ and $s_n \in \rho(P)$, or

b. $\varphi_n = Z$ and Z is free in φ_0 and $s_n \in V(Z)$, or

c. $\varphi_n = [K]\psi$ and $\{ t : s_n \xrightarrow{a} t \text{ and } a \in K \} = \varnothing$

(ii) The play $(s_0, \varphi_0), (s_1, \varphi_1), \cdots, (s_n, \varphi_n), \cdots$ is infinite, and the *unique* fixed point variable X, which occurs infinitely often and which subsumes all other variables occuring infinitely often, identifies a greatest fixpoint subformula of φ.

It follows from the winning condition that given a maximal play (either finite and terminal, or infinite), exactly one of V and R wins.

Proposition 4. *If $(s_0, \varphi_0), (s_1, \varphi_1), \cdots, (s_n, \varphi_n), \cdots$ is an infinite play of the game $\mathcal{G}_V^T(s_0, \varphi)$, then there is a unique variable X that*

(i) occurs infinitely often (i.e. $\varphi_j = X$ for infinitely many j), and

(ii) if Y also occurs infinitely often, then X subsumes Y.

Proof. Because φ_i decreases in size except when it is a fixpoint variable, there are infinite variable occurrences in an infinite play. Suppose, for a contradiction, X and Y are maximal among the infinitely occurring variables and none subsumes the other. It follows that:

(i) The respective fixpoint subformulas named by X and Y occur in different branches of a conjunction or disjunction.

(ii) Further the conjunction (say) is in the scope of some fixpoint subformula named by Z (say), for otherwise, at most one of X and Y can occur infinitely often.

Hence Z subsumes both X and Y and occurs infinitely often in the play, which is a contradiction. □

Example 5.2 (T_1 revisited)**.**

$$\begin{tikzcd}
 & (s, \nu Z.[b]\mathsf{f} \wedge \langle a \rangle Z) \arrow[d] & \\
 & (s, Z) \arrow[d] & \\
 & (s, [b]\mathsf{f} \wedge \langle a \rangle Z)\quad \mathrm{R} \arrow[dl] \arrow[dr] & \\
\mathrm{R}\quad (s, [b]\mathsf{f}) & & (s, \langle a \rangle Z)\quad \mathrm{V} \arrow[uul, bend right]
\end{tikzcd}$$

Who wins the game? Two cases according to R's move:

- R chooses the left branch: V wins.
- R chooses the right branch: If an infinite play should arise, since Z identifies a ν-fixpoint, V wins.

Hence V wins.

Is it always the case that exactly one of R and V has a winning strategy?

Example 5.3 (T_2 revisited)**.** $\mathcal{G}_{\varnothing}^{T_2}(0, \mu Y.\nu Z.[a]((\langle b \rangle \mathsf{t} \vee Y) \wedge Z))$ where

$$\begin{tikzcd}
T_2 \;=\; 0 \arrow[r, bend left, "a"] & 1 \arrow[l, bend left, "a"] \arrow[r, "b"] & 2
\end{tikzcd}$$

with state-set $S = \{0, 1, 2\}$. *Who wins the game?* (See the picture on the next page)

R has a winning strategy:

- At $(1, (\langle b \rangle \mathsf{t} \vee Y) \wedge Z)$: choose right branch
- At $(0, (\langle b \rangle \mathsf{t} \vee Y) \wedge Z)$: choose right branch

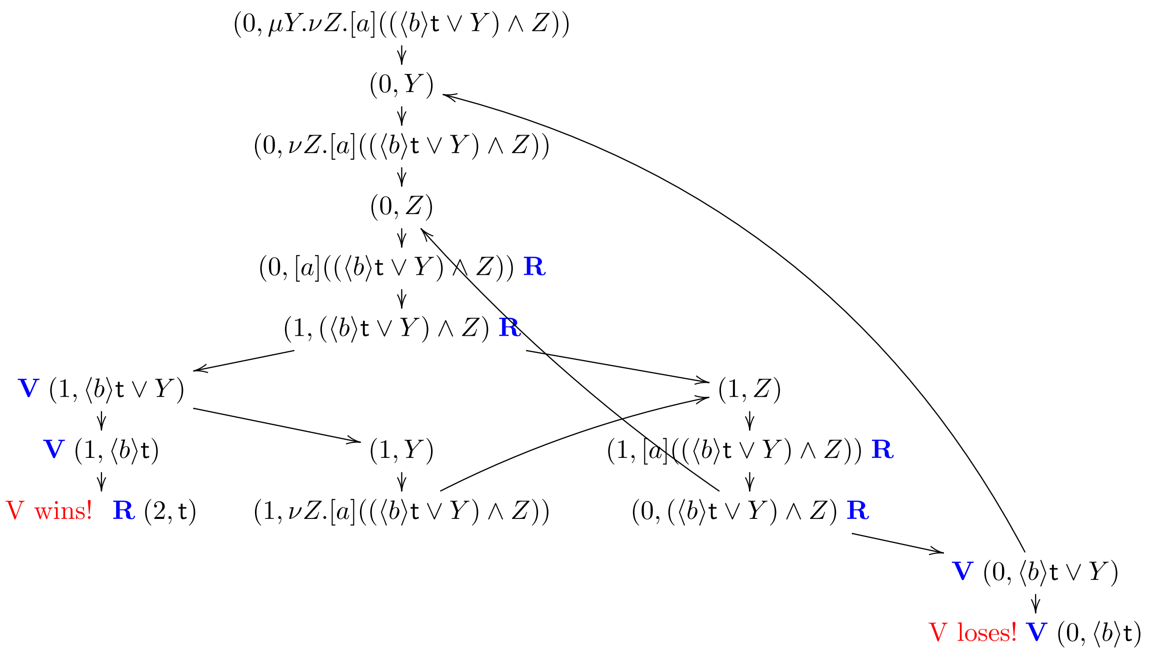


Figure 5.1: Game graph of T_2

At $(0, \langle b\rangle \mathsf{t} \vee Y)$:

- V loses at once if he chooses the left disjunct.
- V also loses if he chooses the right disjunct all the time, since the infinitely occurring and subsuming variable identifies a μ-fixpoint.

Hence R has a winning strategy.

Example 5.4. We present the game graph $\mathcal{G}_{\varnothing}^{T_2}(0, \mu Y.\nu Z.[a](((\langle b\rangle \mathsf{t} \vee Y) \wedge Z))$ in Figure 5.1.

A *strategy* for a player is a set of rules telling the player how to move. A player *uses strategy* π in a play provided all his moves in the play obey the rules in π. *Memoryless strategies* (often called *history-free strategies* in the literature) are strategies that depend only on the last move or position of the play. It follows that for player R, rules have the form:

- at position $(s, \varphi_1 \wedge \varphi_2)$ choose (s, φ_i) where $i = 1$ or $i = 2$
- at position $(s, [K]\varphi)$ such that $\{\, t : s \xrightarrow{a} t \text{ and } a \in K \,\} \neq \varnothing$, choose (t, φ) where $s \xrightarrow{a} t$ and $a \in K$.

Similarly for player V, rules have the form:

- at position $(s, \varphi_1 \vee \varphi_2)$ choose (s, φ_i) where $i = 1$ or $i = 2$
- at position $(s, \langle K\rangle\varphi)$ such that $\{\, t : s \xrightarrow{a} t \text{ and } a \in K \,\} \neq \varnothing$, choose (t, φ) where $s \xrightarrow{a} t$ and $a \in K$.

A strategy π for a player is *winning* if the player wins every play in which he uses π (regardless of how his opponent plays). We aim to prove:

Theorem 5.1 (Fundamental Semantic Theorem). *$s_0 \models_V^T \varphi$ if and only if player V has a memoryless winning strategy for $\mathcal{G}_V^T(s_0, \varphi)$.*

We shall establish the Theorem by proving the following:

(i) If $s_0 \models_V^T \varphi$ then player V has a memoryless winning strategy for $\mathcal{G}_V^T(s_0, \varphi)$.

(ii) If $s_0 \not\models_V^T \varphi$ then player R has a (memoryless) winning strategy for $\mathcal{G}_V^T(s_0, \varphi)$.

It follows from the definition that if one player has a winning strategy, then the other does *not*. I.e. *at most* one player has a winning strategy for a given game $\mathcal{G}_V^T(s_0, \varphi)$. Thus it follows from statements (i) and (ii) in the preceding that *at least* one player has a winning strategy for a given game.

A class of games is said to be *determined* if for every game, there is a winning strategy for exactly one of the players. Thus model checking games of the kind, $\mathcal{G}_V^T(s_0, \varphi)$, are determined. We shall see shortly that these games are *parity games*. In 1975, Donald A. Martin proved that all *Borel games* (which include parity games) are determined.

5.2 Proof of the Fundamental Semantic Theorem

Assume $s_0 \models_V^T \varphi$. We aim to show that V is always able to preserve the "truth of game positions" by making judicious choices, so winning every play.

We list all the fixpoint subformulas of φ, in decreasing order of size, as follows:

$$\sigma_1 Z_1.\psi_1,\ \sigma_2 Z_2.\psi_2,\ \cdots,\ \sigma_n Z_n.\psi_n$$

It follows that:

(i) if $i < j$ then it is impossible for Z_j to subsume Z_i

(ii) if Z_i subsumes Z_j then $i \leq j$.

A *position* in the game $\mathcal{G}_V^T(s, \varphi_0)$ has the form (t, ψ) where ψ may contain free variables in $\{ Z_1, \cdots, Z_n \}$. Our strategy is to show that V can play in such a way that if it reaches (s', ψ) then $s' \models_V^T \psi$. An immediate problem is that since some of the Z_is may occur free in ψ, $\|\psi\|_V^T$ may not be defined!

True positions and valuations We first define valuations $V_0, \cdots, V_n$ by induction on n:

$$\begin{aligned} V_0 &:= V \\ V_{i+1} &:= V_i[Z_{i+1} \mapsto \|\sigma_{i+1} Z_{i+1}.\psi_{i+1}\|_{V_i}^T] \end{aligned}$$

The idea is V_i maps $Z_1, \cdots, Z_i$ to their respective (correct) denotations. E.g. $V_1 : Z_1 \mapsto \|\sigma_1 Z_1.\psi_1\|_{V_0}^T$. Note that $V = V_0$ is not well-defined on $Z_1, Z_2, \cdots, Z_n$, but this is ok since $\sigma_1 Z_1.\psi_1$—as Z_1 is the most subsuming—does not contain any *free* occurrence of $Z_2, \cdots, Z_n$. Similarly, $\|\sigma_2 Z_2.\psi_2\|_{V_1}^T$ is well-defined since it does not contain free occurrences of $Z_3, \cdots, Z_n$, and so on. Thus V_n captures the semantics of all bound variables Z_is i.e. $\|\psi\|_{V_n}^T$ is well-defined for every $\psi \in \mathit{Sub}(\varphi)$.

We say that (t, ψ) is a *true position* just if $t \models_{V_n}^T \psi$. Note that (s_0, φ_0) is a true position, by assumption.

Next we give a more refined valuation that identifies the smallest *least fixpoint approximant* making a given position true. Let

$$\mu Y_1.\chi_1,\ \mu Y_2.\chi_2,\ \cdots,\ \mu Y_m.\chi_m$$

be the set of least fixpoint subformulas of φ, again in decreasing order of size.

A *signature* is just an m-long vector of ordinals. We say that signatures $r < r'$ if r *lexicographically precedes*[1] r'.

Given a signature $r = \alpha_1 \cdots \alpha_m$ and a valuation V, we define valuations $V_0^r, \cdots, V_n^r$ by induction:

$$\begin{array}{rcl} V_0^r & := & V \\ V_{i+1}^r & := & V_i^r \left[Z_{i+1} \mapsto \left\{ \begin{array}{ll} \| \sigma_{i+1} Z_{i+1}.\psi_{i+1} \|_{V_i^r}^T & \text{if } \sigma_{i+1} = \nu \\ \| \mu Y_j^{\alpha_j}.\chi_j \|_{V_i^r}^T & \text{else if } \sigma_{i+1} Z_{i+1} = \mu Y_j \end{array} \right. \right] \end{array}$$

Thus we use a signature $r = \alpha_1 \cdots \alpha_m$ to interpret the *least* fixpoint subformulas in φ so that the j-th such is interpreted by its α_j-th approximant.

μ-signature of a true position

Lemma 5.2. *If $t \models_{V_n}^T \psi$ then there is a smallest signature r such that $t \models_{V_n^r}^T \psi$.*

Thus given a true position (t, ψ), we define its μ-*signature* (or simply *signature*), written $sig^\mu(t, \psi)$, to be the least r such that $t \models_{V_n^r}^T \psi$ holds.

Notation Let r and r' be signatures. We write

- $r(k)$ to mean the kth component of the signature r, and
- $r =_k r'$ to mean that the first k components of the signatures r and r' are identical.

The μ-signature is *unchanged or decreases* when passing through boolean, modal or ν-variable dependencies / edges, and when passing through $(-, Y_j)$, it strictly decreases in the jth-component and is unchanged in each of the $1, \cdots, j-1$ component.

Lemma 5.3 (Signature Decrease)**.** *Whenever the LHS of the relation in question is defined, we have*

(i) $sig^\mu(s, \varphi_1 \vee \varphi_2) = sig^\mu(s, \varphi_i)$, *for some* $i \in \{1, 2\}$.

(ii) $sig^\mu(s, \varphi_1 \wedge \varphi_2) \geq \max(sig^\mu(s, \varphi_1), sig^\mu(s, \varphi_2))$

(iii) $sig^\mu(s, \langle a \rangle \varphi_1) = sig^\mu(t, \varphi_1)$, *for some* t *such that* $s \xrightarrow{a} t$

(iv) $sig^\mu(s, [a]\varphi_1) \geq sig^\mu(t, \varphi_1)$, *for all* t *such that* $s \xrightarrow{a} t$

(v) *Assuming* Z_i *is a* ν-*variable*, $sig^\mu(s, \nu Z_i.\psi_i) = sig^\mu(s, Z_i) = sig^\mu(s, \psi_i)$.

(vi) *Assuming* $Z_i = Y_j$ *is a* μ-*variable*

  (a) $sig^\mu(s, \mu Y_j.\chi_j) =_{j-1} sig^\mu(s, Y_j)$

  (b) $sig^\mu(s, Y_j)(j) > sig^\mu(s, \chi_j)(j)$ *and* $sig^\mu(s, Y_j) =_{j-1} sig^\mu(s, \chi_j)$.

Proof. Exercise. □

[1] We define $a_1 \cdots a_m < b_1 \cdots b_m$ iff $a_1 < b_1$, or $(a_1 = b_1$ and $a_2 \cdots a_m < b_2 \cdots b_m)$.

Winning memoryless stratregy for V Finally we simultaneously give the memoryless strategy for V and prove that it is winning, by appealing to the *Signature Decrease Lemma.*

Suppose (s_m, φ_m) is the current position in the play. Assume that it is a true position, and so, $s_m \vDash_{V_n^{r_m}} \varphi_m$ where $r_m = sig^{\mu}(s_m, \varphi_m)$. If (s_m, φ_m) is a final position, then V is the winner. Therefore either V wins or the play is not yet complete. In the latter, we show how the play is extended to (s_{m+1}, φ_{m+1}) by a case analysis on φ_m.

- $\varphi_m = \psi_1 \wedge \psi_2$: Then R chooses ψ_i for some $i \in \{1, 2\}$ as the next move, and the next position $(s_{m+1}, \varphi_{m+1}) = (s_m, \psi_i)$ remains true, and $s_{m+1} \vDash_{V_n^{r_{m+1}}} \varphi_{m+1}$ with $r_{m+1} \leq r_m$, by the Lemma.
- $\varphi_m = [K]\psi$: Then R chooses as the next move $(s_{m+1}, \varphi_{m+1}) = (t, \psi)$, for some t and some $a \in K$ such that $s_m \xrightarrow{a} t$. Since $s_m \in \|[K]\psi\|^T_{V_n^{r_m}}$, we have $s_{m+1} \in \|\psi\|^T_{V_n^{r_m}}$, so (s_{m+1}, φ_{m+1}) is true and $r_{m+1} \leq r_m$ by the Lemma.

Applying the Signature Decrease Lemma

- $\varphi_m = \psi_1 \vee \psi_2$: V chooses one of ψ_1 and ψ_2 that holds; if both hold, then she chooses the one (say, ψ_1) with the least signature. The next position $(s_{m+1}, \varphi_{m+1}) = (s_m, \psi_1)$ remains true. By the Lemma, $r_{m+1} = r_m$.
- $\varphi_m = \langle K\rangle\psi$: similar to above.
- $\varphi_m = \sigma_i Z_i.\psi_i$. Then $(s_{m+1}, \varphi_{m+1}) = (s_m, Z_i)$. Two cases:
  - $\sigma_i = \nu$: (s_{m+1}, φ_{m+1}) is a true position, and $r_{m+1} = r_m$.
  - $\sigma_i = \mu$: (s_{m+1}, φ_{m+1}) is a true position, but $r_{m+1} =_{i-1} r_m$.
- $\varphi_m = Z_i$. Then $(s_{m+1}, \varphi_{m+1}) = (s_m, \psi_i)$. Two cases:
  - $\sigma_i = \nu$: (s_{m+1}, φ_{m+1}) is a true position, and $r_{m+1} = r_m$.
  - $\sigma_i = \mu$: (s_{m+1}, φ_{m+1}) is a true position, with $r_{m+1} < r_m$ and $r_{m+1} =_{i-1} r_m$.

The case of infinite plays Take an infinite play $(s_0, \varphi_0), (s_1, \varphi_1), \cdots$ in which after (say) (s_k, φ_k), every occurrence of a fixpoint variable is subsumed by Z_i. Suppose, for a contradiction, $Z_i = Y_j$ which identifies a least fixpoint subformula $\mu Y_j.\chi_j$.

Let $k_1, k_2, \cdots$ be the positions in the play where Y_j occurs. The move from (s_{k_i}, Y_j) to (s_{k_i+1}, χ_j) causes a strict decrease of the μ-signature. We show that the remaining moves between k_i and k_{i+1} cannot cancel this decrease out.

We only need to worry about $(s, \mu Y_l.\chi_l) \to (s, Y_l)$, as this is the only case which may cause an increase in the μ-signature. By assumption Y_l is subsumed by Y_j; it follows that $l > j$. Note that the transition $(t, \mu Y_j.\chi_j)$ to (t, Y_j) is not possible in the segment from k_i to k_{i+1} because Y_j is assumed to be subsuming. Hence the increase in μ-signature occurs after the l-th component, and so, there is still an overall decrease in the respective j-prefix of r_z as z progresses from k_i to k_{i+1}. This contradicts the well-foundedness of (fixed-length vectors of) ordinals.

A duality We use a "symmetric" argument to establish the other direction: *If $s \nvDash^T_V \varphi$ then player R has a memoryless winning strategy for $\mathcal{G}^T_V(s, \varphi)$.*

Specifically

- For a position (t, ψ) that is *false* (i.e. $s \notin \|\psi\|_V^T$), define its *ν-signature*, written $sig^\nu(t, \psi)$, to be the least r such that $t \nvDash^T_{V^r_n} \psi$ holds.
- Establish a corresponding *Signature Decrease Lemma* for ν-signatures.
- Define a memoryless winning strategy for R.

Hence V has a memoryless winning strategy for $\mathcal{G}_V^T(s, \varphi)$ iff $s \vDash_V^T \varphi$.

Lemma 5.4. *A player does not have a memoryless winning strategy for $\mathcal{G}_V^T(s, \varphi)$ if and only if he has a memoryless strategy for $\mathcal{G}_V^T(s, \widetilde{\varphi})$, where $\widetilde{\varphi}$ is the positive normal form of $\neg\varphi$.*

Proof. We consider V; the situation for R is exactly dual.

V has no memoryless winning strategy for $\mathcal{G}_V^T(s, \varphi)$
iff $s \nvDash_V^T \varphi$
iff $s \vDash_V^T \neg\varphi$
iff V has a memoryless winning strategy for $\mathcal{G}_V^T(s, \widetilde{\varphi})$ □

5.3 Tableaux for modal mu-calculus

Preliminaries We will not distinguish between different labels namely only consider formulas $[\mathcal{L}]\varphi$ and $\langle\mathcal{L}\rangle\varphi$ which we will denote with $\Box\varphi$ and $\Diamond\varphi$. Note that this is not a necessary condition for the following but makes for concise presentation. From now on we will only consider closed formulas. Since every formula can be written in positive normal form, we will work with *unlabelled mu-formulas* defined inductively by:

$$\varphi ::= P \mid \neg P \mid Z \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \Box\varphi \mid \Diamond\varphi \mid \mu Z.\varphi \mid \nu Z.\varphi$$

All formulas considered are also assumed to be *guarded.* We say that variable Z in a formula φ is *guarded* if every occurrence of Z in φ is in some modal subformula $[a]\chi$ or $\langle a\rangle\chi$ of φ. We say that a formula is *guarded* if for every subformula $\sigma Z.\psi$ of φ, Z is guarded in ψ (where $\sigma = \mu, \nu$). For example $\mu Z.Z \vee \varphi$ is equivalent to the guarded formula $\mu Z.\varphi$ and $\nu Y.(Q \wedge (\mu Z.Y \wedge \Diamond Z)) \vee \Diamond Y$ is equivalent to the guarded formula $\nu Y.(Q \wedge \mu Z.\Diamond Z) \vee \Diamond Y$.

Satisfaction and Models We say a transition system $T = (S, \rightarrow, \lambda)$ with a distinguished node s_0 *satisfies* the formula φ of modal mu-calculus, if $s_0 \models^T \varphi$ (i.e. $s_0 \in \|\varphi\|^T$). In this case we call T a *model* of φ and say that φ is *satisfiable.*

Tableaux: an informal view Consider the formula $\varphi = \nu Y.\mu X.(\Box\Box P \wedge \Diamond X) \vee (\neg P \wedge \Diamond Y)$. We want to ask if φ is satisfiable. We aim to find a model of φ namely a transition system

$T = (S, \to, \lambda)$ and $s_0 \in S$ such that $s_0 \models^T \varphi$. We can proceed as follows.

$$\begin{array}{ll}
 & s_0 \models \nu Y.\mu X.(\Box\Box P \wedge \Diamond X) \vee (\neg P \wedge \Diamond Y) \\
 & s_0 \models (\Box\Box P \wedge \Diamond X) \vee (\neg P \wedge \Diamond Y) \\
 & s_0 \models \Box\Box P \wedge \Diamond X \\
 & s_0 \models \{\Box\Box P, \Diamond X\} \\
\exists s_1 : s_0 \to s_1 & s_1 \models \{\Box P, X\} \\
 & s_1 \models \{\Box P, (\Box\Box P \wedge \Diamond X) \vee (\neg P \wedge \Diamond Y)\} \\
 & s_1 \models \{\Box P, \neg P \wedge \Diamond Y\} \\
 & s_1 \models \{\Box P, \neg P, \Diamond Y\} \\
\exists s_2 : s_1 \to s_2 & s_2 \models \{P, Y\} \\
 & s_2 \models \{P, (\Box\Box P \wedge \Diamond X) \vee (\neg P \wedge \Diamond Y)\} \\
 & s_2 \models \{P, \Box\Box P \wedge \Diamond X\} \\
 & s_2 \models \{P, \Box\Box P, \Diamond X\} \\
\exists s_3 : s_2 \to s_3 & s_3 \models \{\Box P, X\} \\
 & \quad\vdots
\end{array}$$

This "model search" naturally gives rise to the transition system illustrated in Figure 5.2. This search for satisfiability (or soundness) can be formalised using tableaux. The idea is to encapsulate all possible plays on an arbitrary transition system as a tree where each node is labelled by a subset of $Sub(\varphi)$.

$$\begin{tikzcd}
s_0 \arrow[r] & s_1 \arrow[r] & s_2 \arrow[r] & \cdots
\end{tikzcd}$$

where $s_{2i} \in \rho(P)$ and $s_{2i+1} \notin \rho(P)$

Figure 5.2: A model of $\nu Y.\mu X.(\Box\Box P \wedge \Diamond X) \vee (\neg P \wedge \Diamond Y)$.

Trees and Paths

Definition 5.2. A *tree* (over Σ) is a triple $t = (V, \to, \lambda)$ with a distinguished node $r_t \in V$ which satisfies the following conditions.

- $(V, \to)$ is a connected directed graph.
- There are no transitions into r_t.
- For every $v \in V \setminus \{r_t\}$ there is exactly one $v_0 \in V$ such that $v_0 \to v$.
- $\lambda \colon V \to \mathcal{P}(\Sigma)$ is called the *labelling function* of t.

The node r_t is referred to as the *root* of the tree and any node without outgoing transitions is a *leaf*. We use subscripts to emphasise which tree they refer to.

Note that a tree can be viewed as a special case of a transition system.

If $t = (V, \to, \lambda)$ is a tree then a *path* through t is an enumerable set $\mathbb{P} \subseteq V$ such that $r_t \in \mathbb{P}$, if $v_0 \to v \in \mathbb{P}$ then $v_0 \in \mathbb{P}$, and for every $v \in \mathbb{P}$ either v is a leaf or there exists exactly one $v' \in V$ such that $v \rightarrow v'$ and $v' \in \mathbb{P}$. Hence a path can be visualised as a sequence $r_t = v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_n \rightarrow \ldots$ and we write $\mathbb{P}(n)$ to denote v_n.

Tableaux

Recall *Prop* is a (possibly infinite) set of propositions from which the syntax of μ-calculus is defined. Let $Prop^{\neg} = \{\neg P \colon P \in Prop\}$. Uppercase Greek letters such as Γ and Δ denote *sequents*, finite sets of formulas. $\Box\Gamma$ abbreviates the set $\{\Box\varphi : \varphi \in \Gamma\}$ and $\Diamond\Gamma$ is defined analogously. We write Γ, φ to mean $\Gamma \cup \{\varphi\}$, and Γ, Δ to denote $\Gamma \cup \Delta$.

$$(\wedge)\,\frac{\Gamma, \varphi_0 \wedge \varphi_1}{\Gamma, \varphi_0, \varphi_1} \qquad (\vee_0)\,\frac{\Gamma, \varphi_0 \vee \varphi_1}{\Gamma, \varphi_0} \qquad (\vee_1)\,\frac{\Gamma, \varphi_0 \vee \varphi_1}{\Gamma, \varphi_1}$$

$$(\mu)\,\frac{\Gamma, \mu Z.\varphi}{\Gamma, Z} \qquad (\nu)\,\frac{\Gamma, \nu Z.\varphi}{\Gamma, Z} \qquad (Z)\,\frac{\Gamma, Z}{\Gamma, \varphi}\,(Z \text{ identifies } \sigma Z.\varphi)$$

$$(\text{mod})\,\frac{\Box\Gamma, \Diamond\{\delta_1, \cdots, \delta_n\}, \Theta}{\Gamma, \delta_1 \qquad \Gamma, \delta_2 \qquad \cdots \qquad \Gamma, \delta_n}\,(\Theta \subseteq Prop \cup Prop^{\neg} \text{ is consistent})$$

Table 5.1: Rules for generating pre-tableaux.

Definition 5.3. A *pre-tableau* for Ξ is a tree $t = (V, \rightarrow, \lambda)$ over $\bigcup_{\varphi \in \Xi} Sub(\varphi)$ generated by the rules given below from the sequent Ξ, in which every finite branch terminates by reaching a sequent of the form $\Box\Gamma, \Diamond\Delta, \Theta$ where Δ is empty or $\Theta \subseteq Prop \cup Prop^{\neg}$ is inconsistent.

Notice that a pre-tableau is a finitely branching tree and branching only occurs at a (mod)-rule. For each rule in Table 5.1 that is not the mod rule, the distinguished formulas in the upper and lower sequents are called respectively the *principal* and *residual* formulas of the rule.

The tableau rules are read top-down thus our trees grow downwards. For example, in the $(\vee)$-rule from the sequent $\Gamma, \varphi_0 \vee \varphi_1$ we may proceed to either Γ, φ_0 or Γ, φ_1. If the (mod)-rule can be applied to a sequent, then no other rule is applicable to that sequent.

Proposition 5. *Every infinite path in a pre-tableau passes through a (mod)-rule infinitely often.*

Proof. Exercise 5.3. □

Definition 5.4. Fix a pre-tableau $t = (V, \rightarrow, \lambda)$ for Γ and a path $\mathbb{P}$ through t. A finite *trace* (through $\mathbb{P}$) is a sequence of formulas $\varphi_0, \varphi_1, \ldots, \varphi_n$, where, for each i, r_i is the rule that applies at $\mathbb{P}(i)$, satisfying the following:

(i) $\varphi_i \in \lambda(\mathbb{P}(i))$ for each $i \leq n$;

(ii) If $i < n$ and r_i is not the mod rule, then

  (a) if φ_i is the principal formula of the rule r_i, then φ_{i+1} is a residual formula of r_i

  (b) if φ_i is not the principal formula of the rule r_i, then $\varphi_{i+1} = \varphi_i$

(iii) If $i < n$ and r_i is a mod rule instance with premise

$$\Box\gamma_1, \cdots, \Box\gamma_l, \Diamond\delta_1, \cdots, \Diamond\delta_m, \Theta$$

then

(a) if $\varphi_i = \Box\gamma_j$ then $m \geq 1$ and $\varphi_{i+1} = \gamma_j$

(b) if $m \geq 1$ and $\varphi_i = \Diamond\delta_j$ then $\varphi_{i+1} = \delta_j$

In (iii)(a), φ_{i+1} refers to the occurrence of γ_j in the child-sequent of r_i chosen by the path $\mathbb{P}$; similarly for (b). Note that the case of $\varphi_i = P$ is not possible (because $i < n$).

An infinite sequence of formulas $\varphi_0, \varphi_1, \ldots$ is a trace if every finite initial sequence is a trace.

Lemma 5.5. *For each infinite trace there exists a variable that appears infinitely often in the trace and subsumes all other infinitely occurring variables.*

Proof. Exercise. □

In each infinite trace the unique variable identified by Lemma 5.5 will be referred to as the *most significant variable* of the trace. We call an infinite trace a *μ-trace* if its most significant variable is a μ-variable; otherwise it is a *ν-trace.*

Definition 5.5. A pre-tableau is a *tableau* if

(i) the sequent at the leaf of each finite branch is of the form $\Box\Gamma, \Theta$ where Θ is a consistent set of propositions, and

(ii) every infinite trace though every path in the pre-tableau is a ν-trace.

Lemma 5.6. *Let $t = (V, \rightarrow, \lambda)$ be a tableau. Then for every $v \in V$ we have $\lambda(v) \cap (Prop \cup Prop^{\neg})$ is consistent.*

Proof. Exercise 5.3. □

Example 5.5. Of the four pre-tableaux given below for the formula $\mu Z.Q \vee (P \wedge \Diamond Z)$ only the first three are tableaux. The fourth one which is assumed to always pick the right disjunct in $Q \vee (P \wedge \Diamond Z)$ is not a tableau.

$$\begin{array}{rc}
 & \mu Z.Q \vee (P \wedge \Diamond Z) \\ \cline{2-2}
(\mu) & Z \\ \cline{2-2}
(Z) & Q \vee (P \wedge \Diamond Z) \\ \cline{2-2}
(\vee) & Q
\end{array}
\qquad
\begin{array}{rc}
 & \mu Z.Q \vee (P \wedge \Diamond Z) \\ \cline{2-2}
(\mu) & Z \\ \cline{2-2}
(Z) & Q \vee (P \wedge \Diamond Z) \\ \cline{2-2}
(\vee) & P \wedge \Diamond Z \\ \cline{2-2}
(\wedge) & P, \Diamond Z \\ \cline{2-2}
(\text{mod}) & Z \\ \cline{2-2}
(Z) & Q \vee (P \wedge \Diamond Z) \\ \cline{2-2}
(\vee) & Q
\end{array}
\qquad
\begin{array}{rc}
 & \mu Z.Q \vee (P \wedge \Diamond Z) \\ \cline{2-2}
(\mu) & Z \\ \cline{2-2}
(Z) & Q \vee (P \wedge \Diamond Z) \\ \cline{2-2}
(\vee) & P \wedge \Diamond Z \\ \cline{2-2}
(\wedge) & P, \Diamond Z \\ \cline{2-2}
(\text{mod}) & Z \\ \cline{2-2}
(Z) & Q \vee (P \wedge \Diamond Z) \\ \cline{2-2}
(\vee) & P \wedge \Diamond Z \\ \cline{2-2}
(\wedge) & P, \Diamond Z \\ \cline{2-2}
(\text{mod}) & Z \\ \cline{2-2}
(Z) & Q \vee (P \wedge \Diamond Z) \\ \cline{2-2}
(\vee) & Q
\end{array}
\qquad
\begin{array}{rc}
 & \mu Z.Q \vee (P \wedge \Diamond Z) \\ \cline{2-2}
(\mu) & Z \\ \cline{2-2}
(Z) & Q \vee (P \wedge \Diamond Z) \\ \cline{2-2}
(\vee) & P \wedge \Diamond Z \\ \cline{2-2}
(\wedge) & P, \Diamond Z \\ \cline{2-2}
(\text{mod}) & Z \\ \cline{2-2}
(Z) & Q \vee (P \wedge \Diamond Z) \\ \cline{2-2}
(\vee) & P \wedge \Diamond Z \\ \cline{2-2}
 & \vdots \\
 & Z \\
 & \vdots
\end{array}$$

Soundness and Completeness

Lemma 5.7 (Soundness)**.** *If φ has a tableau then φ is satisfiable.*

Proof. Let $t = (V, \rightarrow_t, \lambda)$ be a tableau for φ. Define a transition system $T = (S, \rightarrow_T, \rho)$ and a map $\tau \colon V \rightarrow S$ such that

(i) $\tau(r_t) = s_0 \in S$

(ii) If $v \to_t u$ and the tableau rule applied at v is (mod) then $\tau(v) \to_T \tau(u)$, otherwise $\tau(u) = \tau(v)$.

(iii) $s \in \rho(P)$ if and only if there exists $v \in V$ such that $\tau(v) = s$ and $P \in \lambda(v)$.

We will use t to define a winning strategy for Verifier in the model checking game $\mathcal{G}^T(s_0, \varphi)$, whence the Fundamental Semantic Theorem yields $s_0 \models^T \varphi$.

Suppose we have a play, namely, (s_0, φ_0), (s_1, φ_1), $\ldots$, (s_n, φ_n), in the game $\mathcal{G}^T(s_0, \varphi)$, and a corresponding trace φ'_0, φ'_1, $\ldots$, φ'_m in the tableau t such that

- there exists $i_0 = 0 < i_1 < \cdots < i_n < i_{n+1} = m + 1$ so that for every $j \leq n$ and every $k \in [i_j, i_{j+1})$, $\varphi'_k = \varphi_j$, and
- φ'_m is principal in the node v of t associated to it, or a box or diamond formula in case of mod rule, and $\tau(v) = s_n$.

Suppose $\varphi_n \notin \mathit{Prop} \cup \mathit{Prop}^\neg$. We show how the play and the trace can be extended.

Case I. It is Verifier's move. Then either $\varphi_n = \psi_0 \vee \psi_1$ or $\Diamond\psi_0$ respectively. As t is a tableau and $\varphi'_m = \varphi_n$ the trace can be extended by ψ where $\psi \in \{\psi_0, \psi_1\}$ or $\psi = \psi_0$ respectively. If u denotes the node associated to ψ in this trace, set Verifier's choice to be $(s_{n+1}, \varphi_{n+1}) = (\tau(u), \psi)$.

Case II It is Refuter's move. Then for every choice of (s_{n+1}, φ_{n+1}) by Refuter, the trace can be extended accordingly.

This is in fact a winning strategy for Verifier. Suppose $(s_0, \varphi_0), (s_1, \varphi_1), \ldots, (s_n, \varphi_n)$ is a finite play using the above strategy. It follows that there exists a node $v \in V$ such that $\tau(v) = s_n$ and $\varphi_n \in \lambda(v)$ is principal at v. If $\varphi_n \in \mathit{Prop}$, this play is winning for Verifier by definition. If $\varphi_n = \neg P$ then since $\lambda(v) \cap (\mathit{Prop} \cup \mathit{Prop}^\neg)$ is consistent, $s_n \notin \rho(P)$ and Verifier wins this play too. Furthermore, an infinite play corresponds to an infinite trace in the tableau. Since every infinite trace is a ν-trace this implies that the play is winning for Verifier. □

Lemma 5.8 (Completeness). *If φ is satisfiable then φ has a tableau.*

Proof. Let T be a transition system with distinguished node s_0 such that $s_0 \models^T \varphi$. Use Verifier's winning strategy in $\mathcal{G}^T(s_0, \varphi)$ (given by the Fundamental Semantic Theorem) to make your choices in defining a pre-tableau for φ. Then every infinite trace in the tableau corresponds to an infinite play in the game and hence is a ν-trace. Moreover, every finite branch in the pre-tableau will be of the form $\Box\Gamma, \Theta$ where Θ is consistent as otherwise there will be maximal plays that end in a position (s, P) with $s \not\models P$ or $(s, \Diamond\psi)$ which would be losing for Verifier. □

The Soundness and Completeness Lemmas give us a characterisation of satisfaction in terms of the existence of tableaux:

Theorem 5.2. *A formula is satisfiable iff it has a tableau.*

Corollary 5.1 (Tree Model Property). *If a formula φ of modal mu-calculus has a model then it has a tree model.*

Proof. Suppose φ is satisfiable. The Completeness Lemma 5.8 implies that φ has a tableau. The proof of the Soundness Lemma 5.7 then shows how to extract a tree model for φ from this tableau. □

5.4 Parity Games

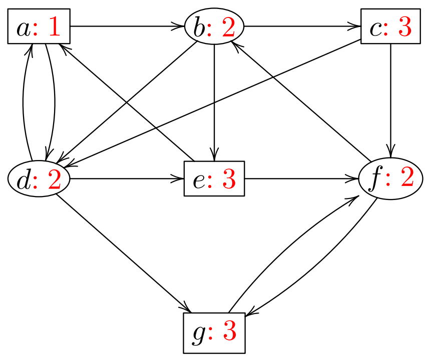
Definition 5.6. A *parity game* is a tuple $G = \langle N, E, v_I, \lambda, \Omega \rangle$ where

- $\langle N, E \subseteq N \times N \rangle$ is a directed graph such that E is *total* (i.e. for each u, there is some v, such that $(u, v) \in E$), and $v_I \in N$ is the *start vertex*
- $\lambda : N \longrightarrow \{ V, R \}$ labels each vertex with R (Refuter) or V (Verifier); $\lambda(v)$ indicates which player is responsible for moving from v
- $\Omega : N \longrightarrow \{ 0, \cdots, p \}$ assigns a *priority* (or *colour*) to each vertex v

A play begins with a token on the start vertex v_I. When the token is on u and $\lambda(u) = P$, player P moves it along an outgoing edge (u, v) to v. A *play* is an infinite path $v_I\, v_1 \cdots v_i \cdots$ in the graph visited by the token.

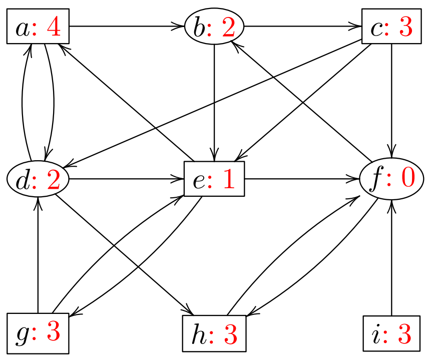
The winner of a play is determined by the *min-parity* condition: if the least priority that occurs infinitely often in the sequence $\Omega(v_I)\, \Omega(v_1)\, \Omega(v_2) \cdots$ is even then V wins; otherwise R wins. There is an equivalent *max-parity* condition.

Example 5.6 (A parity game)**.** V-moves are circled; R-moves are boxed. Prioities are indicated in red after the colon. Who has a winning strategy starting from a?



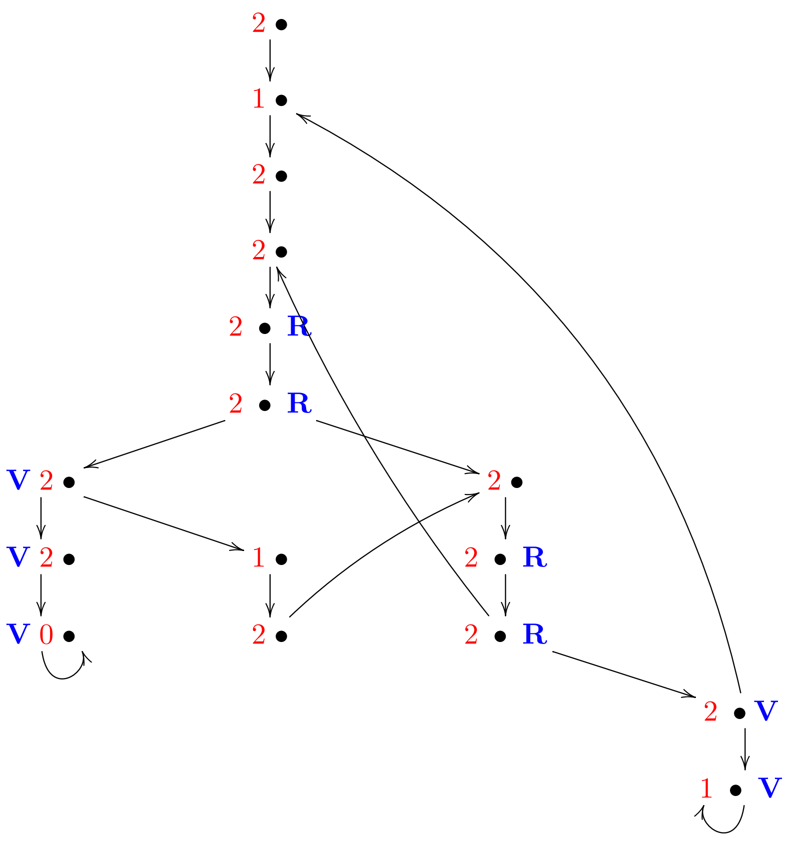
Recall: V wins an infinite play just if the least infinitely occurring priority is even. Ans: V has a winning strategy: $b \mapsto c$, $f \mapsto g$, $d \mapsto g$.

Example 5.7.



From which vertices does V have a winning strategy? Equivalently, what is the *winning region* of V?

Example 5.8. Parity Game $\mathcal{G}_{\varnothing}^{T_2}[0, \mu Y.\nu Z.[a]((\langle b \rangle \mathtt{t} \vee Y) \wedge Z)]$ where T_2 is defined in Example 5.3.



Every modal mu-calculus model checking instance determines a parity game. Precisely,

Proposition 6. *The mu-calculus model checking problem* reduces *to the decision problem* Parity*: given a parity game, does V have a winning solution?*

5.5 Solvability and Determinacy for Finite Parity Games

Let $G = \langle N, \to, v_0, \lambda, \Omega \rangle$ be a parity game, and $v \in N$. Write $G(v) := \langle N, \to, v, \lambda, \Omega \rangle$. Let $P \in \{ V, R \}$. The *winning region of* P, written W_P, is the subset of G-vertices v such that P has a winning strategy starting from v (i.e. in the game $G(v)$). It follows from the definition that $W_R \cap W_V = \varnothing$. Trivially $W_R \cup W_V \subseteq N$.

Theorem 5.3 (Solvability)**.** *Let $G = \langle N, \to, v_0, \lambda, \Omega \rangle$ be a finite parity game. Both the winning regions, W_V and W_R, and the corresponding memoryless winning strategies for V and R are computable.*

Intuitively $\mathsf{Force}^i_P(X)$ is the set of vertices from which player P can force play to enter the set X of vertices in at most i moves.

Definition 5.7 (Force Sets)**.** Let $X \subseteq N$.

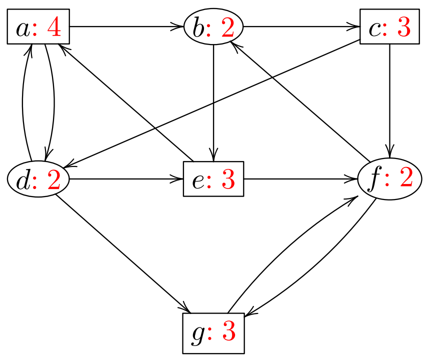
$$\begin{array}{rcl}
\mathsf{Force}_P^0(X) & := & X \quad \text{for } P \in \{V, R\} \\
\mathsf{Force}_R^{i+1}(X) & := & \mathsf{Force}_R^i(X) \\
& \cup & \{ j : \lambda(j) = R \wedge \exists k \in \mathsf{Force}_R^i(X). j \to k \} \\
& \cup & \{ j : \lambda(j) = V \wedge \forall k. j \to k \Rightarrow k \in \mathsf{Force}_R^i(X) \} \\
\mathsf{Force}_V^{i+1}(X) & := & \mathsf{Force}_V^i(X) \\
& \cup & \{ j : \lambda(j) = V \wedge \exists k \in \mathsf{Force}_V^i(X). j \to k \} \\
& \cup & \{ j : \lambda(j) = R \wedge \forall k. j \to k \Rightarrow k \in \mathsf{Force}_V^i(X) \} \\
\mathsf{Force}_P(X) & := & \bigcup_{i \geq 0} \mathsf{Force}_P^i(X)
\end{array}$$

Rank and Forcing Strategy The definition of force set provides a method for computing it. As i increases, we calculate $\mathsf{Force}_P^i(X)$ until it is the same as $\mathsf{Force}_P^{i-1}(X)$. Clearly this must hold when $i \leq (|N| - |X|) + 1$.

If $j \in \mathsf{Force}_P(X)$ and current position is j, then P can force play from j into X, regardless of how the opponent moves. (Vertex j itself need not belong to X.) The *rank* of such a vertex j is the least index i such that $j \in \mathsf{Force}_P^i(X)$.

For each $i \in \mathsf{Force}_P(X)$ belonging to P, either $i \in X$ or there is $i \to k$ and $k \in \mathsf{Force}_P(X)$, so the *forcing strategy* for P is to choose a k with the least rank.

Example 5.9 (Force Sets of a Parity Game)**.** V-moves are circled; R-moves are boxed. Prioities are indicated in red.



i	$\mathsf{Force}_V^i(\{ f \})$
0	$\{ f \}$
1	$\{ f, g \}$
2	$\{ f, g, d \}$
3	$\{ f, g, d, c \}$
4	$\{ f, g, d, c, b \}$
5	$\{ f, g, d, c, b, a \}$
6	$\{ f, g, d, c, b, a, e \}$

Since $\mathsf{Force}_V(\{ f \})$ is the entire vertex set, and f has the least priority which is even, V has a winning strategy from every vertex. V's forcing strategy: $f \mapsto g,\ d \mapsto g,\ b \mapsto c$

Subgames If $X \subseteq N$, then $G - X$ is the result of removing all vertices in X from G, all edges from vertices in X, all edges into vertices in X.

The subgraph $G - X$ may or may not be a parity game (because not all vertices may be total). If it is, then $G - X$ is a *subgame* of G.

Proposition 7. *If G is a game and X is a subset of vertices, then the subgraph $G - \mathsf{Force}_P(X)$ is a subgame.*

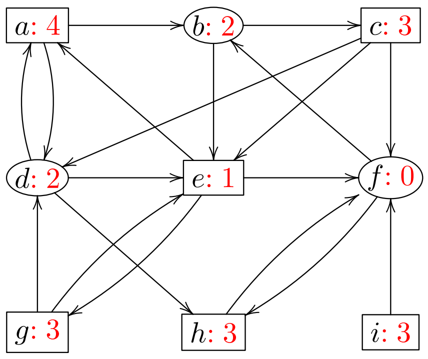
Exercise 5.1. Prove the proposition.

Algorithmic Solution of Finite Parity Games **WReg**(G) computes G's winning regions, W_V and W_R. (The respective memoryless winning strategies can be extracted from the correctness proof.) Assume the least priority is even; otherwise swap players in the algorithm.

Algorithm WReg(G). Output: W_V and W_R

1. Let v be a G-vertex of the least priority (assumed even). Set $X := \mathsf{Force}_V(\{ v \})$.
2. If $X = N$ then return $W_V := N$ and $W_R := \varnothing$.
3. Else run **WReg**$(G - X)$, and let W'_R and W'_V be the winning regions.
   (a) If V can guarantee transition from v to $W'_V \cup X$, i.e.,
   $$\left\{ \begin{array}{l} (a)\ \lambda(v) = V \ \wedge\ \exists v' \,.\, v \to v' \ \wedge\ v' \in (W'_V \cup X), \text{ or} \\ (b)\ \lambda(v) = R \ \wedge\ \forall v' \,.\, v \to v' \ \Rightarrow\ v' \in (W'_V \cup X) \end{array} \right\} \text{then}$$
   $$\text{return} \left\{ \begin{array}{lcl} W_V & := & W'_V \cup X \\ W_R & := & W'_R. \end{array} \right.$$
   (b) Else set $X' := \mathsf{Force}_R(W'_R)$ in G. Run **WReg**$(G - X')$, and let W''_V and W''_R be the winning regions. Return $\left\{ \begin{array}{lcl} W_R & := & W''_R \cup X' \\ W_V & := & W''_V. \end{array} \right.$

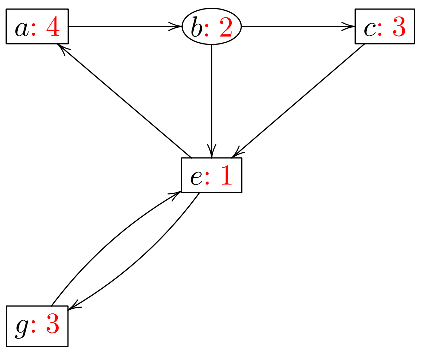
Example 5.10 (Computing Winning Regions)**.** Parity game G:



Run **WReg**(G): let winning regions be W_V and W_R. Set $X = \mathsf{Force}_V(\{ f \}) = \{ d, f, h, i \}$. **WReg**$(G - X)$ returns regions $W'_V = \varnothing$ and $W'_R = (G - X)$.

Since $f \to h$ and $h \in X$, return $W_V = X$ and $W_R = W'_R$.

Parity game $G - X$:



In $G - X$, $\mathsf{Force}_R(\{ e \}) = (G - X)$. Hence $\mathbf{WReg}(G - X)$ returns winning regions $W'_V = \varnothing$ and $W'_R = (G - X)$.

Proof of Theorem 5.3: Correctness of WReg(G)

Proof. By induction on $n = |N|$. Base case $n = 1$ is trivial. Inductive case. $G - X$ has less than n vertices. By the induction hypothesis $\mathbf{WReg}(G - X)$ computes winning regions W'_V and W'_R.

Case 1. V can guarantee transition from v to $W'_V \cup X$ iff (a) or (b). Claim: (i) $W'_V \cup X \subseteq W_V$; and (ii) $W'_R \subseteq W_R$; this is sufficient since $W'_R \cup W'_V \cup X = N$. Thus the memoryless V-strategy on $W'_V \cup X$ is: 1. On W'_V, play the winning strategy (thanks to the induction hypothesis). 2. On X, play the "forcing" strategy, eventually reaching v. 3. From v, move back to $W'_V \cup X$. For R, use the memoryless winning strategy given by the induction hypothesis Proof of Claim (i): If the play eventually remains in W'_V then V wins by the induction hypothesis; otherwise the play passes through v infinitely often, then V wins since the priority of v, which is the least, is even. (ii) holds because, starting in W'_R, R can guarantee that the play remains in W'_R (because no V-move in $G - X$ can transition to X).

Case 2. R can guarantee transition from v to W'_R. Thus $v \in X'$. By the induction hypothesis $\mathbf{WReg}(G - X')$ computes the winning regions W''_V and W''_R. Claim: (i) $W''_R \cup X' \subseteq W_R$ (ii) $W''_V \subseteq W_V$. Proof of (i): R can move to W'_R from any move in X', and there he can guarantee that the play remains in W'_R. From a move in W''_R, V can choose to move to either W''_R or X'. In both cases, R wins the play. (ii) is clear since V can guarantee that the play remains in W''_V.

In the worst case, $\mathbf{WReg}$ is called $2^{|N|}$ times; thus running time is exponential in $|N|$. □

Uniformly Winning Memoryless Strategies Let $X \subseteq W_V$. A V-strategy is *uniformly winning on* X just if it is winning for V from every $v \in X$.

Lemma 5.9. *Given a parity game, if $W_V \neq \varnothing$ then V has a memoryless strategy that is uniformly winning on W_V.*

Proof. Exercise □

PARITY is the decision problem: Given a parity game $G = \langle N, \rightarrow, v_0, \lambda, \Omega \rangle$, is $v_0 \in W_V$?

Proposition 8. PARITY $\in$ **NP** $\cap$ **co-NP**

Proof. We first show that Parity is in **NP**. Guess a uniformly winning V-strategy, which is succinct i.e. its size is $O(|N|)$. It can be verified in time polynomial in $|G|$ whether $v_0 \in W_V$. Claim: $v_0 \in W_V$ iff in the *strategy transition graph* (i.e. one edge from V-vertices and all edges from R-vertices), V cannot enter a loop from v_0 such that the least priority is odd. Suppose (w.l.o.g.) that the least odd and the largest priorities are 1 and p (even) respectively: just verify for the subgraphs over

$$\bigcup_{i=1}^{p} \Omega^{-1}(i), \bigcup_{i=3}^{p} \Omega^{-1}(i), \cdots, \bigcup_{i=p-1}^{p} \Omega^{-1}(i)$$

whether they contain a strongly connected component reachable from v_0 which meets the priorities $1, 3, \cdots, p$ respectively.

The complementary problem $v_0 \in (N - W_V)$ is just $v_0 \in W_R$, which is in **NP** using the same argument but with the players swapped. □

At the moment, the best deterministic algorithms for solving Parity run essentially in time $O(|N|^d)$ where d is the number of priorities (see Grädel et al. (2002) for an overview).

One of the best known open problem in the foundations of verification:

Conjecture 5.1. Parity $\in$ **P**

Determinacy for Finite Parity Games

Theorem 5.4 (Determinacy). *Let $G = \langle N, \rightarrow, v_0, \lambda, \Omega \rangle$ be a finite parity game. Then $N \subseteq W_V \cup W_R$. Further if $v \in W_V$ (resp. W_R) then V (resp. R) has a memoryless winning strategy from v.*

Proof. Suppose the image of Ω is $\{0, 1, \cdots, p-1\}$. Proof is by induction on p. Base case is trivial. Inductive case: if least priority is odd, swap players. Let HF_R be the set of vertices from which R has a memoryless winning strategy, and let ρ be a memoryless R-strategy that is *uniformly winning* on HF_R. It suffices to show that V has a memoryless winning strategy from every vertex in $N - HF_R$.

Case 1. No vertex in $G - HF_R$ has the least priority, 0. Apply the induction hypothesis to the subgame $G - HF_R$, since no vertex in it has priority 0. I.e. $G - HF_R$ can be partitioned into winning regions W'_V and W'_R in which memoryless winning strategies exist for the respective players. Now $W_R = HF_R \cup W'_R$ and $W_V = W'_V$; G is thus partitioned and the respective players have the appropriate memoryless winning strategies.

Case 2. $G - HF_R$ contains a vertex with priority 0. Claim: V can guarantee that, starting from a vertex in $G - HF_R$, the play remains there. Now either the play stays in $(G - HF_R) - \mathsf{Force}_V(\Omega^{-1}(0) - HF_R)$ or it visits $\mathsf{Force}_V(\Omega^{-1}(0) - HF_R)$ infinitely often. In the former case, V wins by the induction hypothesis with a memoryless strategy; in the latter, V wins by infinitely many visits to a vertex with priority 0, also with a memoryless strategy. □

5.6 Muller Games

We can define other types of graph games by changing the winning condition. For instance, a *Muller game* is a tuple $G = \langle N, E, v_0, \lambda, \mathit{Win} \rangle$ such that

- N is a finite set of nodes with an initial vertex $v_0 \in N$,
- $\langle N, E \subseteq N \times N \rangle$ is a directed graph such that E is total,
- $\lambda : N \to \{ \mathrm{V}, \mathrm{R} \}$ labels each vertex with V (Verifier) or R (Refuter) to indicate which player is responsible for moving from each position,
- $\mathit{Win} = \{ F_1, \ldots, F_k \}$ where $F_i \subseteq N$.

A play $\rho \in N^\omega$ is winning if $\inf(\rho) \in \mathit{Win}$ (i.e. if the play satisfies the Muller condition specified by $\{ F_1, \ldots, F_k \}$).

We outline an approach to proving the following result.

Theorem 5.5 (Buchi and Landweber (1969))**.** *Let G be a finite Muller game. Then the winning regions W_V and W_R in G for each player are computable, and only finite memory is required to implement the winning strategies.*

Similar results hold for other games with ω-regular winning conditions.

Solving Muller games by reduction to parity games Fix some game $G = \langle N, E, v_0, \lambda, \mathit{Win} \rangle$. Given some parity automaton $A = \langle Q, \Sigma, q_0, \Delta, \Omega \rangle$ over the alphabet $\Sigma = N$, the *composition game* $A \circ G$ is the parity game $\langle Q \times N, E', q_0 \times v_0, \lambda', \Omega' \rangle$ where

- $((q, v), (r, w)) \in E'$ iff both $(q, v, r) \in \Delta$ and $(v, w) \in E$,
- $\lambda'((q, v)) = \lambda(v)$, and
- $\Omega'((q, v)) = \Omega(q)$.

Proposition 9. *Let A be a deterministic parity automaton recognizing precisely the plays in Win. Then Verifier has a winning strategy in the parity game $A \circ G$ iff Verifier has a winning strategy in G.*

Proof. Exercise. □

Hence, in order to show that Muller games are solvable, we must show that deterministic Muller automata are equivalent to deterministic parity automata. This was shown by Gurevich and Harrington using a data structure known as the *latest appearance record*.

Theorem 5.6 (Gurevich and Harrington (1982))**.** *Let A' be a deterministic Muller automaton. Then there is a deterministic parity automaton A with $L(A') = L(A)$.*

The solvability of Muller games follows then from Proposition 9, Theorem 5.6, and the fact that parity games are solvable.

Memory required in Muller games Like parity games, it can be shown that Muller games are determined. Unlike parity games, Muller games do not always have memoryless strategies.

Proposition 10. *Muller games do not always have memoryless strategies.*

Proof. Exercise 5.6. □

However, it can be shown that Muller games have finite memory strategies. Roughly speaking, a *finite memory strategy* is a strategy that depends only on a finite summary of the history of the play, rather than the entire history of the play.

We describe this formally using a memory structure. A *memory structure* $\mathcal{M}$ using a finite set M of memory states has a

- next move function $\mathit{next} : M \times N_{\mathrm{V}} \to N$ (where N_{V} is $\{\, v \in N : \lambda(v) = \mathrm{V} \,\}$),
- memory update function: $\mathit{update} : M \times N \to M$,
- initial memory state $m_0 \in M$.

A play $\rho = v_0 v_1 v_2 \ldots$ in the game is consistent with $\mathcal{M}$ if there is a sequence of memory states $m_0 m_1 \ldots$ such that $m_i = \mathit{update}(m_{i-1}, v_i)$ and if $v_i \in N_{\mathrm{V}}$, then $v_{i+1} = \mathit{next}(m_i, v_i)$. We say a strategy σ for Verifier *uses finite memory of size* n if there is some memory structure $\mathcal{M}$ with $\mathsf{size}(M) = n$ such that every play that is possible using the strategy is consistent with $\mathcal{M}$. Note that a memoryless strategy is a special case when $M = \varnothing$.

The fact that Muller games have finite memory strategies can be shown using an extension of Proposition 9 and the fact that parity games have memoryless strategies. The idea is that we can convert a memoryless strategy in the parity game $A \circ G$ into a finite memory strategy in the Muller game G, where the memory structure is based on A. We leave the details of this conversion as an exercise.

Problems

5.1 This question proves the basic result: *modal mu-calculus is bisimulation invariant.*

First a definition. We say that a relation $B \subseteq S_1 \times S_2$ between the states of labelled transition systems $\mathcal{T}_1$ and $\mathcal{T}_2$, where $\mathcal{T}_i = \langle S_i, \longrightarrow_i, \rho_i \rangle$ $(i = 1, 2)$, is a *bisimulation* just if whenever $(s, t) \in B$

- for all $P \in \mathit{Prop}$, $s \in \rho_1(P)$ iff $t \in \rho_2(P)$
- for all $a \in \mathcal{L}$
  - for all $s' \in S_1$, if $s \xrightarrow{a}_1 s'$ then there exists a state $t' \in S_2$ such that $t \xrightarrow{a}_2 t'$ and $(s', t') \in B$, and
  - for all $t' \in S_2$, if $t \xrightarrow{a}_2 t'$ then there exists a state $s' \in S_1$ such that $s \xrightarrow{a}_1 s'$ and $(s', t') \in B$.

Two states s and t are *bisimulation equivalent*, written $s \sim_{\mathcal{T}_1,\mathcal{T}_2} t$, just if there is a bisimulation relation B such that $(s, t) \in B$.

Prove that if $s \sim_{\mathcal{T}_1,\mathcal{T}_2} t$ then for all modal mu-calculus sentences φ, we have $s \models^{\mathcal{T}_1} \varphi$ iff $t \models^{\mathcal{T}_2} \varphi$.

5.2 Prove the *Signature Decrease Lemma*: Whenever the left-hand side of the relation in question is defined, we have:

(a) $\mathit{sig}^{\mu}(s, \varphi_1 \vee \varphi_2) = \mathit{sig}^{\mu}(s, \varphi_i)$, for some $i \in \{1, 2\}$.

(b) $\mathit{sig}^{\mu}(s, \varphi_1 \wedge \varphi_2) \geq \max(\mathit{sig}^{\mu}(s, \varphi_1), \mathit{sig}^{\mu}(s, \varphi_2))$

(c) $\mathit{sig}^{\mu}(s, \langle a \rangle \varphi_1) = \mathit{sig}^{\mu}(t, \varphi_1)$, for some t such that $s \xrightarrow{a} t$

(d) $\mathit{sig}^{\mu}(s, [a]\varphi_1) \geq \mathit{sig}^{\mu}(t, \varphi_1)$, for all t such that $s \xrightarrow{a} t$

(e) Assuming Z_i is a ν-variable, $\mathit{sig}^{\mu}(s, \nu Z_i.\psi_i) = \mathit{sig}^{\mu}(s, Z_i) = \mathit{sig}^{\mu}(s, \psi_i)$.

(f) Assuming $Z_i = Y_j$ is a μ-variable

  (i) $\mathit{sig}^{\mu}(s, \mu Y_j.\chi_j) =_{j-1} \mathit{sig}^{\mu}(s, Y_j)$

  (ii) $\mathit{sig}^{\mu}(s, Y_j)(j) > \mathit{sig}^{\mu}(s, \chi_j)(j)$ and $\mathit{sig}^{\mu}(s, Y_j) =_{j-1} \mathit{sig}^{\mu}(s, \chi_j)$ (and so $\mathit{sig}^{\mu}(s, Y_j) > \mathit{sig}^{\mu}(s, \chi_j)$).

5.3 Write down a replacement (mod)-rule that captures satisfiability for $\mathcal{L}_{\mu}$-formulas with labels.

5.4 Prove the following.

(i) Every infinite path in a pre-tableau passes through a (mod)-rule infinitely often.

(ii) Let $t = (V, \rightarrow, \lambda)$ be a tableau and suppose $v \in V$. Then for all $P \in \mathit{Prop}$ we have $\{P, \neg P\} \not\subseteq \lambda(v)$.

5.5 Show in the proof of the Soundness Lemma 5.7 there is in fact a memoryless strategy for verifier.

Hint: you need to address the following issue. If between two (mod)-rules a disjunction, say $\psi_0 \vee \psi_1$, is broken down more than once with a different disjunct chosen, what will a successful strategy for Verifier pick without referring to the history of the play so far?

5.6 Prove that every mu-calculus model checking game $\mathcal{G}_V^T(s, \varphi)$ determines an equivalent parity game namely define a parity game $\mathcal{G}_V^T[s, \varphi]$ such that *Player P has a memoryless winning strategy for $\mathcal{G}_V^T(s, \varphi)$ if and only if Player P has a memoryless winning strategy for $\mathcal{G}_V^T[s, \varphi]$.*

5.7 Assume the notations of the lecture course. Let s_0 be a state of a finite labelled transition system T. WLOG let φ be a closed formula in positive normal form with no occurrences of atomic propositions. Suppose φ has n fixpoint variables; and m least fixpoint variables $Y_1, \cdots, Y_m$, naming subformulas $\mu Y_1.\chi_1, \cdots, \mu Y_m.\chi_m$ in decreasing order of size.

Fix a memoryless V-strategy σ for the game $\mathcal{G}_\varnothing^T(s_0, \varphi)$. A *signature assignment* is an assignment $\mathcal{S}$ of signatures (of length m) to each position (t, ψ). We say that a signature assignment is *σ-consistent* just if for each position u

- if $u = (s, Y_j)$ then $\mathcal{S}((s, \chi_j)) <_j \mathcal{S}(u)$ and $\mathcal{S}((s, \chi_j)) =_{j-1} \mathcal{S}(u)$
- if $u = (s, \mu Y_j.\chi_j)$ then $\mathcal{S}((s, Y_j)) =_{j-1} \mathcal{S}(u)$
- if u is a V-position then $\mathcal{S}(\sigma(u)) \le \mathcal{S}(u)$
- if u is a R-position then for all successor vertices v we have $\mathcal{S}(v) \le \mathcal{S}(u)$.

(i) Prove that $s \models_\varnothing^T \varphi$ if, and only if, there is a memoryless V-strategy σ and a σ-consistent signature assignment $\mathcal{S}$.

(ii) Hence prove that the modal mu-calculus model checking problem is in **NP** $\cap$ **co-NP**.

5.8 The model checking problem of the modal mu-calculus can be given a game characterization:

(a) $s \models_V^T \varphi$ iff player V has a memoryless winning strategy for $\mathcal{G}_V^T(s, \varphi)$.

(b) $s \not\models_V^T \varphi$ iff player R has a memoryless winning strategy for $\mathcal{G}_V^T(s, \varphi)$.

In the lectures, we proved (i). This question proves (ii).

Henceforth we fix a transition system T and a normal formula φ. Let

$$\nu Y_1.\chi_1,\ \nu Y_2.\chi_2,\ \cdots,\ \nu Y_m.\chi_m$$

be the set of greatest fixpoint formulas in φ, again in decreasing order of size.
We define valuations $V_0, \cdots, V_n$ by induction:

$$\begin{array}{rcl} V_0 & = & V \\ V_{i+1} & = & V_i[Z_{i+1} \mapsto \|\sigma_{i+1} Z_{i+1}.\psi_{i+1}\|_{V_i}^T] \end{array}$$

Thus we can make sense of $\|\psi\|^T_{V_n} \subseteq S$, for any $\psi \in \mathit{Sub}(\varphi)$.

Given a signature $r = \alpha_1, \cdots, \alpha_m$ and a valuation V, we define *ν-valuations*[2] $V^r_0, \cdots, V^r_n$ by induction:

$$\begin{array}{rcl} V^r_0 & = & V \\ V^r_{i+1} & = & V^r_i[E_{i+1}/Z_{i+1}] \end{array}$$

where

$$E_{i+1} = \begin{cases} \|\sigma_{i+1} Z_{i+1}.\psi_{i+1}\|^T_{V^r_i} & \text{if } \sigma_{i+1} = \mu \\ \|\nu Y_j^{\alpha_j}.\chi_j\|^T_{V^r_i} & \text{if } \sigma_{i+1} Z_{i+1} = \nu Y_j \end{cases}$$

Recall that if $s \notin \|\nu Z.\psi\|^T_V$ then there is an ordinal α such that $s \notin \|\nu Z^\alpha.\psi\|^T_V$ and for all $\beta < \alpha$, we have $s \in \|\nu Z^\beta.\psi\|^T_V$.

Let $\psi \in \mathit{Sub}(\varphi)$. If $s \notin \|\psi\|^T_V$, we define the *ν-signature* of (s, ψ), written $\mathit{sig}^\nu(s, \psi)$, to be the $<$-least signature $r = \alpha_1 \cdots \alpha_m$ such that $s \notin \|\psi\|^T_{V^r_n}$.

(a) Prove the *Signature Decrease Lemma*: Whenever the left-hand side of the equation or inequality is defined:

i. $\mathit{sig}^\nu(s, \varphi_1 \wedge \varphi_2) = \mathit{sig}^\nu(s, \varphi_1)$ or $\mathit{sig}^\nu(s, \varphi_1 \wedge \varphi_2) = \mathit{sig}^\nu(s, \varphi_2)$

ii. $\mathit{sig}^\nu(s, \varphi_1 \vee \varphi_2) \geq \max(\mathit{sig}^\nu(s, \varphi_1), \mathit{sig}^\nu(s, \varphi_2))$

iii. $\mathit{sig}^\nu(s, [a]\varphi_1) \geq \mathit{sig}^\nu(t, \varphi_1)$, for some t such that $s \xrightarrow{a} t$

iv. $\mathit{sig}^\nu(s, \langle a\rangle\varphi_1) \geq \mathit{sig}^\nu(t, \varphi_1)$ for all t such that $s \xrightarrow{a} t$, or s does not have an a-transition

v. Assuming Z_i is a μ-variable, $\mathit{sig}^\nu(s, \mu Z_i.\psi_i) = \mathit{sig}^\nu(s, Z_i) = \mathit{sig}^\nu(s, \psi_i)$

vi. Assuming $Z_i = Y_j$ is a ν-variable, $\mathit{sig}^\nu(s, \nu Y_j.\chi_j)$ is the same as $\mathit{sig}^\nu(s, Y)$ on the first $j - 1$ components

vii. Assuming $Z_i = Y_j$ is a ν-variable, $\mathit{sig}^\nu(s, Y_j) > \mathit{sig}^\nu(s, \chi_j)$, and the first $j - 1$ components of the two signatures are the same but not the j-component.

(b) Hence prove that *if $s \not\models^T_V \varphi$ then R has a memoryless winning strategy.*

5.9 Let φ be a monotonic function on a powerset 2^S. Prove that for every $U \subseteq S$,

$$U \subseteq \nu X.\varphi(X) \quad \Leftrightarrow \quad U \subseteq \varphi(\nu X.(U \cup \varphi(X))).$$

5.10 Give a procedure to translate CTL-formulas (defined in Section 4.7) into equivalent modal μ-calculus formulas. You may assume that the transition systems are total (i.e. every node has at least one outgoing edge).

Based on this translation, prove that CTL model checking is decidable, and state an upper bound on the complexity of CTL model checking.

5.11 Prove that Muller games do not always have memoryless strategies.

[Hint: Construct a particular Muller game for which Verifier has a finite memory winning strategy, but any memoryless strategy for Verifier loses.]

[2]This is dual to, and not to be confused with, the μ-valuations as defined in the lectures.

Chapter 6

Tree Automata, Rabin's Theorems and S2S

[*We gratefully acknowledge Michael Vanden Boom's contribution to this chapter.*]

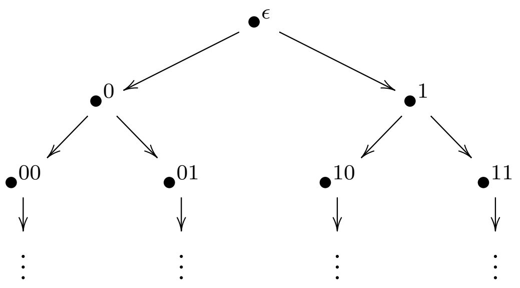
Synopsis

Non-deterministic and alternating tree automata examples. Non-emptiness and Rabin's Basis Theorem. Complementation is straightforward for alternating automata. Alternating tree automata can be simulated by non-deterministic tree automata. S2S/WS2S: syntax and semantics. Expressivity of S2S/WS2S. Rabin's Tree Theorem: S2S and WS2S are decidable.

References (Löding, 2011; Rabin, 1969, 1970; Muller and Schupp, 1987; Muller et al., 1986)

6.1 Trees and Non-deterministic Tree Automata

We are interested in infinite (full) binary trees whose nodes are labelled by letters of an alphabet Σ. Formally a Σ-*labelled binary tree* is a function $t : \{0,1\}^* \to \Sigma$ i.e. the label of the tree t at node $u \in \{0,1\}^*$ is $t(u)$.



We write $\mathfrak{T}^\omega_\Sigma$ for the collection of Σ-labelled binary trees. Henceforth by a tree, we mean an element of $\mathfrak{T}^\omega_\Sigma$. A *tree language* is just a subset $T \subseteq \mathfrak{T}^\omega_\Sigma$. A *maximal path* (or, simply, *path*) of a tree t is a sequence $\pi = u_0\,u_1\,u_2 \cdots$ of tree nodes whereby $u_0 = \epsilon$ (the root of the tree) and $u_{i+1} = u_i\,0$ or $u_{i+1} = u_i\,1$, for every $i \geq 0$. The ω-*word determined by* π is $t(u_0)\,t(u_1)\,t(u_2) \cdots \in \Sigma^\omega$.

Definition 6.1. A *tree automaton* A (for Σ-labelled binary trees) is a tuple $(Q, \Sigma, q_0, \Delta, Acc)$, where

- Q is the finite set of states, q_0 is the initial state
- $\Delta \subseteq Q \times \Sigma \times Q \times Q$ is the transition relation, and
- Acc is the acceptance condition (such as Büchi, Muller, Rabin and Parity).

The automaton is *deterministic* just if for every q and a, there is at most one transition (or quadruple) in Δ the first two components of which are q and a.

A *run-tree* of a tree automaton A over a tree t is an assignment of states to tree nodes i.e. a function $\rho : \{0,1\}^* \to Q$ such that

- $\rho(\epsilon) = q_0$, and
- for all $u \in \{0,1\}^*$, $(\rho(u), t(u), \rho(u\,0), \rho(u\,1)) \in \Delta$.

Note that a run-tree is a Q-labelled binary tree.

Let Acc be an acceptance condition for automata over ω-words. A run-tree ρ is accepting w.r.t. condition Acc just if every path of ρ is accepting w.r.t. Acc; we say that a tree automaton $A = (Q, \Sigma, q_0, \Delta, Acc)$ accepts a given tree t just if there is run-tree of A over t which is accepting w.r.t. Acc. Thus we have *Büchi tree automata, Muller tree automata, Rabin tree automata, Parity tree automata*, etc. For example, a Büchi tree automaton $A = (Q, \Sigma, q_0, \Delta, F)$ *accepts* a tree t just if there exists a run-tree ρ of A over t such that in every path of ρ, a final state from F occurs infinitely often.

The *tree language* recognised by the tree automaton A, denoted $L(A)$, is the set of trees accepted by A.

Example 6.1. Consider the language T_1 of $\{a, b\}$-labelled binary trees t such that t has a path with infinitely many a's. The Büchi tree automaton $(\{q_a, q_b, \top\}, \{a, b\}, q_a, \Delta, \{q_a, \top\})$ where

$$\Delta : \begin{cases} (q_*, a) & \mapsto & \{(q_a, \top), (\top, q_a)\} \\ (q_*, b) & \mapsto & \{(q_b, \top), (\top, q_b)\} \\ (\top, *) & \mapsto & \{(\top, \top)\} \end{cases}$$

recognises the language T_1 (where $*$ mean a or b). Clearly the automaton accepts every tree from the state $\top$. Observe that every run-tree has a single path labelled with states q_a (and possibly q_b), and the rest of the run-tree is labelled with $\top$. There are infinitely many states q_a on this path if and only if there are infinitely many vertices labelled by a on the path of the input tree. Thus the non-deterministic automaton descends the input tree t guessing such a path.

Example 6.2. Consider the language $T_2 := \mathfrak{T}^{\omega}_{\{a,b\}} \setminus T_1$. I.e. T_2 consists of $\{a, b\}$-labelled binary trees t such that every path of t has only finitely many a. Define a deterministic Muller tree automaton $(\{q_a, q_b\}, \{a, b\}, q_a, \Delta, \{\{q_b\}\})$ where

$$\Delta : \begin{cases} (q_*, a) & \mapsto & \{(q_a, q_a)\} \\ (q_*, b) & \mapsto & \{(q_b, q_b)\} \end{cases}$$

Then, given a tree t, for each path π of t, there are infinitely many occurrences of b (respectively a) on π if and only if the corresponding path of the unique run-tree ρ over t has infinitely many occurrences of q_b (respectively q_a); it follows that $t \in T_2$ if and only if for every path in the unique run-tree, the set of infinitely occurring states is $\{ q_b \}$. Hence the automaton recognises T_2.

In constrast to automata over ω-words, deterministic Muller tree automata and non-deterministic Büchi tree automata are not equivalent.

Theorem 6.1. *The language T_2 (of Example 6.2) is not recognisable by any Büchi tree automaton, whether deterministic or not.*

Proof. Assume, for a contradiction, that the Büchi tree automaton $A = (Q, \{ a, b \}, q_0, \Delta, F)$ recognises T_2. Let $n = |F| + 1$. Consider the following $\{ a, b \}$-labelled binary tree t:

$$t \ : \ u \mapsto \begin{cases} a & u \in (1^+ 0)^i \text{ for } i \in \{ 1, \cdots, n \} \\ b & \text{otherwise} \end{cases}$$

Since $t \in T_2$, there is a accepting run-tree ρ of A on t. On the path 1^ω, a final state is visited, say, at $v_0 = 1^{m_0}$. On the path $1^{m_0} 0 1^\omega$, final states are visited infinitly often. Suppose a final state is reached, for the first time after 0, at $v_1 = 1^{m_0} 0 1^{m_1}$. By repeated the argument, we obtain visits to final states at the nodes $v_0 = 1^{m_1}, v_1 = 1^{m_0} 0 1^{m_1}, \cdots, v_n = 1^{m_0} 0 1^{m_1} 0 \cdots 0 1^{m_n}$. Now, there exist $i < j$ such that the same final state appears at v_i and v_j. It follows from the definition of t that between v_i and v_j, at least one label a occurs (at the node $v_i 0$).

Construct a new tree t' by copying the (respective labels of the) part between v_i and v_j repeatedly. Similarly we construct the corresponding run-tree ρ' form ρ. Thus A also accepts t' i.e. $t' \in T_2$, but in t', infinitely many a occur on the new path π, which is a contradiction. □

Notation Following Vardi and Kupferman, we use acronym XYZ where

- X ranges over automaton modes: deterministic, non-deterministic and alternating,
- Y ranges over acceptance / winning conditions: Büchi, Muller, Rabin, Streett, parity, and weak,
- Z ranges over input structures: words and trees.

For example, DMW and NPT are shorthand for deterministic Muller word automaton and non-deterministic parity tree automaton respectively.

6.2 Non-deterministic Parity Tree Automata

A *parity tree automaton* is a tuple $A = (Q, \Sigma, q_I, \Delta, \Omega)$ with priority map $\Omega : Q \to \{ 0, \cdots, k \}$. It accepts a tree t just if there is a run-tree ρ of A over t such that for every path of ρ, the least priority that occurs infinitely often is even.

Example 6.3. Consider the parity tree automaton $(\{ q_a, q_b \}, \{ a, b \}, q_a, \Delta, \Omega)$ where Δ is as defined in Example 6.2, and $\Omega : q_a \mapsto 1; q_b \mapsto 2$. For every path in a given tree, there are only finitely many occurrences of a if, and only if, the there are finitely many occurrences of q_a in the corresponding path of the (unique) run-tree, which is so if, and only if, the least priority that occurs infinitely often is 2. Thus the automaton recognises T_2.

Theorem 6.2. *(i) Given a DMW, there is an algorithm to construct an equivalent DPW, and vice versa.*

(ii) Given an NMT, there is an algorithm to construct an equivalent NPT, and vice versa.

Lemma 6.1. *Consider the language consisting of $\{a, b\}$-labelled binary trees t such that t has at least one vertex labelled with a. The language is not recognisable by a deterministic tree automaton with any of the acceptance conditions we have considered.*

Acceptance Parity Game Given a NPT $A = (Q, \Sigma, q_I, \Delta, \Omega)$ and a tree t, we define a parity game, called the *acceptance parity game*, $\mathcal{G}_{A,t} = (N, E, (\epsilon, q_I), \lambda, \Omega')$ as follows. Writing $N_V := \lambda^{-1}(V)$ and $N_R := \lambda^{-1}(R)$

- $N_V = \{0, 1\}^* \times Q$
- $N_R = \{0, 1\}^* \times (Q \times Q)$
- for each vertex $(v, q) \in N_V$, for each transition $(q, a, q_0, q_1) \in \Delta$ with $t(v) = a$, we have

$$((v, q), (v, (q_0, q_1)) \in E$$

- for each vertex $(v, (q_0, q_1)) \in N_R$, we have

$$((v, (q_0, q_1)), (v\,0, q_0)), ((v, (q_0, q_1)), (v\,1, q_1)) \in E.$$

- $\Omega' : (v, q) \mapsto \Omega(q)$ and $(v, (q_0, q_1)) \mapsto \max(\Omega(q_0), \Omega(q_1))$.

It follows from the definition of the game $\mathcal{G}_{A,t}$ that there is a one-one correspondence between accepting run-trees of A over t and winning strategies for Verifier in $\mathcal{G}_{A,t}$.

Lemma 6.2. *Verifier has a winning strategy in $\mathcal{G}_{A,t}$ from vertex (ϵ, q_I) if and only if $t \in L(A)$.*

The Non-emptiness Parity Game Given a NPT $A = (Q, \Sigma, q_I, \Delta, \Omega)$, we define a parity game, called the *non-emptiness parity game*, $\mathcal{G}_A = (N, E, q_I, \lambda, \Omega')$ as follows.

- $N_V = Q$
- $N_R = \Delta$
- for each vertex $q \in N_V$, and for each transition $(q, a, q_0, q_1) \in \Delta$, we have

$$(q, (q, a, q_0, q_1)) \in \ E$$

- for each vertex $(q, a, q_0, q_1) \in N_R$, we have

$$((q, a, q_0, q_1), q_0), ((q, a, q_0, q_1), q_1) \in E$$

- $\Omega' : q \mapsto \Omega(q)$ and $(q, a, q_0, q_1) \mapsto \Omega(q)$.

Lemma 6.3. *Verifier has a winning strategy in $\mathcal{G}_A$ from vertex q_I if and only if $L(A) \neq \varnothing$.*

One nice thing about viewing acceptance and non-emptiness of an NPT as a game is that we can take advantage of the key result about memoryless determinacy of parity games.

Theorem 6.3 (Memoryless Determinacy)**.** *Let G be a parity game. From every vertex of G, one of the two players has a memoryless winning strategy.*

Proof. In descriptive set theory, the *Borel determinacy theorem* (Martin, 1975) states that every Gale-Stewart game whose winning set is a Borel set is determined. Parity games lie in the third level of the Borel Hierarchy, hence they are determined. Proofs of memoryless determinacy of parity games can be found in (Emerson and Jutla, 1991; Mostowski, 1991; Zielonka, 1998). □

Recall that a non-empty regular ω-language contains an ultimately periodic ω-word. We show a corresponding result for non-empty tree languages recognisable by parity tree automata.

Definition 6.2. A tree $t \in \mathfrak{T}^{\omega}_{\Sigma}$ is said to be *regular* just if there is a deterministic finite automaton (DFA) equipped with output, which gives, for every $w \in \{0, 1\}^*$, the label $t(w)$ at node w. The automaton has the form $B = (Q_B, \{0, 1\}, q^B_0, \delta_B, f_B)$ where $\delta_B : Q \times \{0, 1\} \to Q$ is the transition function, and $f_B : Q_B \to \Sigma$ is the output function.

Theorem 6.4 (Rabin Basis Theorem)**.** *(i) The emptiness problem for NPT is decidable.*

(ii) If a parity tree automaton accepts some tree then it accepts a regular tree.

Proof. (i) Consider the non-emptiness game $\mathcal{G}_A$. This is a parity game on a finite graph, so using results from the previous chapter, there is an algorithm to determine the winning region for each player. By the previous lemma, Verifier has a winning strategy from q_I iff $L(A) \neq \varnothing$.

(ii) Suppose $L(A) \neq \varnothing$. It follows that in the parity game $\mathcal{G}_A$, Verifier has a *memoryless* winning strategy from q_I. Construct a DFA B with output, using states Q. The memoryless strategy in $\mathcal{G}_A$ chooses for each state q a transition (q, a, q_0, q_1), so we set the output function $f_B(q) = a$ and transition function $\delta_B(q, d) = q_d$. This DFA B generates a regular tree accepted by A.

□

Closure properties of parity tree automata We would like to show that these non-deterministic tree automata have good closure properties, like non-deterministic automata on ω-words.

A major theorem of this chapter is the closure of NPT under complementation. The result was first obtained by Michael Rabin in a landmark paper (Rabin, 1969). Gurevich and Harrington (1982) used games to simplify the proof; they proved a bounded memory theorem for the Rabin condition. A further simplification for the parity condition was proved independently by Emerson and Jutla (1991) and by Mostowski (1991). These approaches stayed always within non-deterministic forms of automata.

We take a different approach, following (Löding, 2011). In order to help us prove closure of non-deterministic parity tree automata under union, intersection, complementation, and projection, we take a detour through alternating tree automata.

6.3 Alternating Parity Tree Automata

Alternating tree automata are a generalisation of non-deterministic tree automata introduced by Muller and Schupp (1987). Running an alternating automaton on a tree can be viewed as a game (described below) between Refuter and Verifier.

The transition function of the alternating automaton describes which moves are controlled by each player. Formally, the transition function δ maps a state and input letter to a *positive boolean formula* $\mathcal{B}^+(\{0,1\}\times Q)$ built up from atoms in $\{0,1\}\times Q$ using $\wedge$ and $\vee$. For instance, if $q, q_b \in Q$, then one possible positive boolean formula is $(0,q)\wedge(1,q)\wedge\big((0,q_b)\vee(1,q_b)\big)$.

The idea is that each atom $(d,r)\in\{0,1\}\times Q$ describes a direction d to go in the tree, and the next state r of the automaton when it moves in that direction. The particular atom from the formula is chosen by playing a game, with Verifier resolving disjunctions and Refuter resolving conjunctions.

For instance, assume we are currently in node x in the tree with $t(x)=a$ and the automaton in state q. If $\delta(q,a) := (0,q)\wedge(1,q)\wedge\big((0,q_b)\vee(1,q_b)\big)$, then one possible move would be for Refuter to choose the conjunct $(0,q_b)\vee(1,q_b)$, and then Verifier to choose the atom $(1,q_b)$. The automaton would then move to the right successor of x (the node $x1$), change to state q_b, and continue the computation from there.

A play is a sequence of moves like this, and Verifier's goal is to ensure that the acceptance condition of the automaton is satisfied, regardless of how Refuter plays.

Before we describe formally the semantics in terms of the acceptance game, we give an example (taken from Löding (2011)).

Example 6.4. Consider $T_3 := \{t \in \mathfrak{T}^\omega_{\{a,b\}} : \text{below every } a\text{-node there is a } b\text{-node}\}$ (by this, we mean for every a-node u, there is a b in the subtree rooted at u; we do not require that there is an immediate successor of u that is labelled b). Define an APT $A := (\{q, q_b, \top\}, \{a,b\}, q, \delta, \Omega)$ where

$$\delta : \begin{cases} (q,a) & \mapsto & (0,q)\wedge(1,q)\wedge\big((0,q_b)\vee(1,q_b)\big) \\ (q,b) & \mapsto & (0,q)\wedge(1,q) \\ (q_b,a) & \mapsto & (0,q_b)\vee(1,q_b) \\ (q_b,b) & \mapsto & (0,\top) \\ (\top,*) & \mapsto & (0,\top) \end{cases}$$

and $\Omega : q,\top \mapsto 0; q_b \mapsto 1$.

Consider some tree $t \in \mathfrak{T}^\omega_{\{a,b\}}$.

In state q, Refuter chooses a path in t. Any time he sees an a on this path, he has the option of switching to state q_b, which represents a challenge to Verifier to witness a b below the current a-node.

In state q_b, Verifier chooses a path. If Verifier can witness a b, then she moves to a sink state $\top$ with priority 0, so Verifier wins. If not, then Verifier remains forever in state q_b with priority 1, so Verifier loses.

We now proceed to a formal definition of an alternating automaton. Although we give the definitions only for alternating parity automata, the definitions can be adapted for other acceptance conditions.

Definition 6.3. *An* alternating parity tree automaton (APT) A *is a tuple* $(Q,\Sigma,q_0,\delta,\Omega)$ *where*

- *Q is the finite set of states, q_0 is the initial state*
- *$\delta : Q\times\Sigma \to \mathcal{B}^+(\{0,1\}\times Q)$ is the transition function, and*
- *$\Omega : Q \to \{0,\dots,k\}$ is the priority function.*

Acceptance Game for Alternating Parity Automata Given APT $A = (Q, \Sigma, q_I, \delta, \Omega)$ and tree t, we define the *acceptance parity game*, $\mathcal{G}_{A,t} = (N, E, (\epsilon, q_I), \lambda, \Omega')$ as follows.

Let $M := \{0,1\}^* \times Q$.

We set $N := M \cup (\{0,1\}^* \times \mathcal{B}^+(\{0,1\} \times Q))$.

Writing $N_V := \lambda^{-1}(V)$ and $N_R := \lambda^{-1}(R)$

- N_V is the set of nodes of the form $(x,q) \in M$, or nodes of the form (x,ψ) for ψ a disjunction or a single atom;
- N_R is the set of nodes of the form (x,ψ) for ψ a conjunction;
- for each vertex $q \in Q$, $x \in \{0,1\}^*$, $d \in \{0,1\}$, and $\psi \in \mathcal{B}^+(\{0,1\} \times Q)$, we have the following edges in the game graph:

$$\begin{aligned} ((x,q),(x,\delta(q,t(x))) &\in E \\ ((x,\psi_1 \wedge \psi_2),(x,\psi_i)) &\in E \quad \text{for } i \in \{1,2\} \\ ((x,\psi_1 \vee \psi_2),(x,\psi_i)) &\in E \quad \text{for } i \in \{1,2\} \\ ((x,(d,q)),(xd,q)) &\in E \end{aligned}$$

- the priority function defining the winning condition is:

$$\Omega' : \begin{cases} (x,q) & \mapsto & \Omega(q) \\ (x,\psi) & \mapsto & \max \Omega(Q) \end{cases}$$

The main positions in the game are of the form $(x,q) \in M$. In order to decide on the next position of the form $(xd,r) \in M$, Verifier and Refuter play a game based on $\delta(q,t(x))$, with Verifier controlling disjunctions and Refuter controlling conjunctions. A play is winning for Verifier if it satisfies the parity condition.

We define the *language* $L(A)$ *recognised by* A to be the set of trees t such that Verifier has a winning strategy in $\mathcal{G}_{A,t}$ starting from vertex (ϵ, q_I).

Strategies Fix a strategy σ for Verifier in $\mathcal{G}_{A,t}$. Given a path $\pi \in M^*$ in $\mathcal{G}_{A,t}$ ending in some position of the form $(x,q) \in M$, we write *next-positions*(σ,π) to denote the set of possible positions of the form $(xd,r) \in M$ that are consistent with Verifier's choices in σ when the history of the play is π.

We can picture a strategy σ as a *run tree* or *strategy tree* where the root is labelled with (ϵ, q_I), and each node v labelled (x,q) has a children *next-positions*(σ,π) where π is the sequence of labels from the root of the strategy tree to v. Note that the strategy tree can have a different degree (i.e. even if the input is a binary tree, the number of children of each node in the strategy tree need not be two).

Figure 6.1 shows a possible strategy tree from the acceptance game $\mathcal{G}_{A,t}$, for the pictured tree t and the automaton in Example 6.4.

Special Types of Alternating Tree Automata Non-deterministic tree automata are a special type of alternating automata where every transition is of the form $\bigvee_i (0,q_0^i) \wedge (1,q_1^i)$. Likewise, deterministic tree automata are a special type of alternating automata where every transition is of the form $(0,q_0) \wedge (1,q_1)$.

An *alternating weak tree automaton* (AWT) $A = (Q, \Sigma, q_1, \delta, \Omega)$ is a special type of APT such that

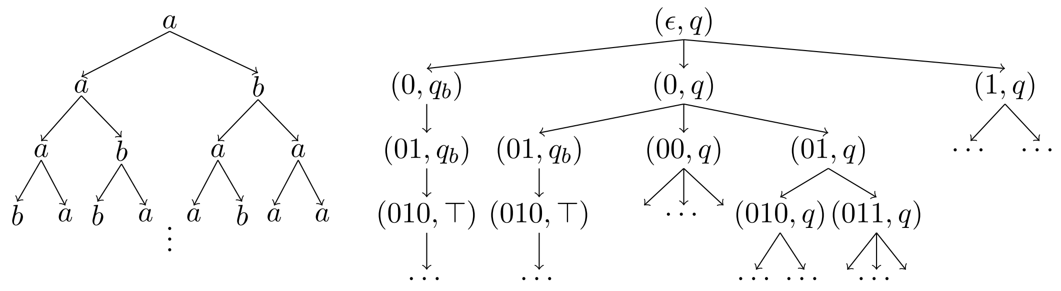


Figure 6.1: Input tree t, and possible strategy tree in $\mathcal{G}_{A,t}$ for A as in Example 6.4.

- the range of Ω consists of one even priority called the *accepting priority*, and one odd priority called the *non-accepting priority*,

and there is a partition of Q into $Q_1, \ldots, Q_k$ such that

- for all i, either $\Omega(q)$ is accepting for all $q \in Q_i$, or $\Omega(q)$ is non-accepting for all $q \in Q_i$, and
- for all $q \in Q_i$ and $a \in \Sigma$, if (d, r) appears in $\delta(q, a)$, then $r \in Q_j$ for $j \geq i$.

In other words, Ω classifies each partition Q_i as accepting or non-accepting, and any move consistent with δ must remain in the same partition, or move to a "higher" partition. This means that any play in an acceptance game $\mathcal{G}_{A,t}$ for an AWT A must either stabilise in an accepting partition (winning for Verifier) or a non-accepting partition (winning for Refuter); no play can switch infinitely many times between accepting and non-accepting priorities.

Note that Example 6.4 is actually an AWT, with partitions $Q_1 = \{ q \}$ and $Q_2 = \{ q_b \}$.

6.4 Closure Properties

We now seek to prove that APT and NPT are closed under the operations of union, intersection, complementation, and projection.

Closure under union and intersection is straightforward for alternating parity automata.

Lemma 6.4 (Closure under Union and Intersection)**.** *Given APT A_1 and A_2, there is an algorithm to construct an APT for $L(A_1) \cup L(A_2)$ and an APT for $L(A_1) \cap L(A_2)$.*

Proof. Exercise. □

Complementation is also straightforward for alternating automata. This is one advantage of using alternating automata instead of non-deterministic automata.

Theorem 6.5 (Closure under Complementation)**.** *Given an APT A, there is an algorithm to construct an APT for $\mathfrak{T}^{\omega}_{\{a,b\}} \setminus L(A)$.*

Proof (Sketch). The idea is that we *dualise* the transition function and acceptance condition of $A = (Q, \Sigma, q_0, \delta, \Omega)$.

The dual of a transition function $\delta : Q \times \Sigma \to (\mathcal{B}^+(\{ 0, 1 \} \times Q)$ is the transition function $\widetilde{\delta} : Q \times \Sigma \to (\mathcal{B}^+(\{ 0, 1 \} \times Q)$ obtained by exchanging $\vee$ and $\wedge$ in all of the formulas in δ.

The dual of the parity acceptance condition given by the priority function Ω, is the parity acceptance condition given by $\widetilde{\Omega}(q) := \Omega(q) + 1$.

Dualising like this switches the roles of Refuter and Verifier in the corresponding acceptance games. By determinacy of parity games (Theorem 6.3), this means $\widetilde{A} := (Q, \Sigma, q_0, \widetilde{\delta}, \widetilde{\Omega})$ recognises $\mathfrak{T}^{\omega}_{\{a,b\}} \setminus L(A)$. □

Showing closure under projection is more challenging for alternating automata than it is for non-deterministic automata. Recall that given $s \times t \in \mathfrak{T}^{\omega}_{\Gamma\times\Sigma}$, the Σ-projection of $s \times t$ is the tree t, and the Σ-projection of the language $T \subseteq \mathfrak{T}^{\omega}_{\Gamma\times\Sigma}$ is denoted $\pi_\Sigma(T)$.

Projection is straightforward for non-deterministic automata, since the non-deterministic automaton can guess the information that was removed during projection. Using this construction directly on an alternating automaton does not work because there is no way to ensure that Verifier guesses the same information in different "copies" of the alternating automaton that are processing the same part of the tree.

However, we can show that any APT can be *simulated* by an NPT, and then apply the projection construction from before to the NPT.

That is, we first show that alternating parity automata are no more powerful than non-deterministic parity automata on infinite trees. The proof relies on two fundamental results:

(i) memoryless determinacy of parity games (Theorem 6.3), and

(ii) complementation and determinisation of NBW.

Theorem 6.6 (Simulation of APT by NPT)**.** *Given an APT A, there is an algorithm to construct an NPT B such that $L(A) = L(B)$.*

Proof. Let $A = (Q, \Sigma, \delta, q_I, \Omega)$ be an APT on Σ-labelled binary trees. We aim to construct an equivalent NPT B. Informally speaking, on input t, B will "guess" a memoryless strategy for Verifier in the acceptance game $\mathcal{G}_{A,t}$, and simultaneously check that this strategy is winning. Correctness will follow from memoryless determinacy of parity games (Theorem 6.3).

A memoryless strategy σ for Verifier must specify how Verifier should move when in a position (x, ψ) for $x \in \{0,1\}^*$ and $\psi \in \mathcal{B}^+(\{0,1\} \times Q)$ such that $\psi = \psi_1 \vee \psi_2$ (note that the other positions controlled by Verifier have only one outgoing edge, so there is nothing for the strategy to specify).

A key observation is that such a memoryless strategy can be represented as a *LocStr*-labelled binary tree

$$f : \{0,1\}^* \to \mathit{LocStr}$$

where *LocStr* is the finite set of partial functions $\hat{\sigma} : \mathcal{B}^+(\{0,1\} \times Q) \to \mathcal{B}^+(\{0,1\} \times Q)$ describing *local strategies* for Verifier such that for each $\psi = \psi_1 \vee \psi_2$, either $\hat{\sigma}(\psi) = \psi_1$ or $\hat{\sigma}(\psi) = \psi_2$.

Take such a strategy function f and consider the tree $t \times f : \{0,1\}^* \to \Sigma \times \mathit{LocStr}$, in which the node u has label $(t(u), f(u))$. This is a tree annotated with the memoryless strategy described by f.

Let $\Sigma' = \Sigma \times \mathit{LocStr} \times \{0,1\}$. Consider a word $\alpha = (a_0, f_0, d_0)\,(a_1, f_1, d_1) \cdots$ over Σ', such that $a_0 = t(\epsilon)$ and $f_0 = f(\epsilon)$, and $a_{i+1} = t(d_0 \cdots d_i)$ and $f_{i+1} = f(d_0 \cdots d_i)$. The word α specifies a path $d_0 d_1 \cdots$, as well as the local strategies on this path. Formally, we say that a *play over* α is a word $(v_0, q_0)\,(v_1, q_1) \cdots \in M^\omega$ such that

- $v_0 = \epsilon$ and $q_0 = q_I$
- $v_{i+1} = d_0 \cdots d_i$
- $(v_{i+1}, q_{i+1}) \in$ *next-positions*$(f_i, (v_i, q_i))$

Thus a play over α is a play in the acceptance game $\mathcal{G}_{A,t}$ on $d_0 d_1 \cdots$ when Verifier makes her choices according to f (technically, it is the restriction of a play to the positions in M).

It is straightforward to construct a NBW C that accepts exactly those words α over Σ' such that there is a play over α that does not satisfy the parity condition. The idea is that C guesses a play consistent with f over the branch described by α, and checks that the parity condition is not satisfied on this play. By McNaughton's Theorem, there is a DPW D that is equivalent to the complement of C, i.e. D accepts those words $\alpha \in (\Sigma')^\omega$ for which all plays over α satisfy the original parity condition from A.

A strategy function f is winning for Verifier in $\mathcal{G}_{A,t}$ if, and only if, every play is winning for Verifier when she plays according to f, regardless of Refuter's choices. Observe that a path in the tree $t \times f$ specifies a word $\alpha \in (\Sigma')^\omega$. It follows that f is a winning strategy for Verifier if, and only if, each α specified by a path in the tree $t \times f$ is accepted by D.

Thus the NPT tree automaton B recognising A operates as follows: given an input tree t, it guesses an annotation of t with a memoryless strategy function f, and runs D on the annotated tree $t \times f$. By memoryless determinacy of parity games (Theorem 6.3), this is enough to capture precisely the language of A. This completes our proof of Theorem 6.5. □

We can now prove closure under projection.

Lemma 6.5 (Closure under Projection). *Given an APT recognising the language T of* $(\Gamma \times \Sigma)$-*labelled binary trees, there is an algorithm to construct an APT recognising $\pi_\Sigma(T)$.*

Proof. Given an APT A over $\Gamma \times \Sigma$, construct an equivalent NPT B using Theorem 6.6. Then define an NPT C over Σ which does, in any step, the following. For input letter $b \in \Sigma$ guess the Γ-component a and proceed by a B-transition for the input letter (a, b). □

Because we have shown that APT and NPT are equivalent, this means that both APT and NPT are closed under union, intersection, complementation and projection.

6.5 S2S and Rabin's Tree Theorem

The Logical System S2S The logical system *S2S* (monadic second-order logic of 2 successors) is defined over first-order variables $x, y, \cdots$ ranging over $\{0,1\}^*$ (nodes in the full binary tree) and over second-order variables $X, Y, \cdots$ ranging over $2^{\{0,1\}^*}$ (sets of nodes of the full binary tree).

Terms are built up from first-order variables and ϵ by the two successors, represented as concatenation with 0 and 1 respectively.

Let s and t be terms. The *atomic formulas* are

- $X(s)$ "s is in X"
- $s \leq t$ "s is a prefix of t"
- $s = t$ "s is equal to t".

The formulas of S2S are built up from the atomic formulas using the standard boolean connectives, and closed under first- and second-order quantifiers $\exists$ and $\forall$.

The *structure* of the infinite full binary tree is $\mathbf{t}_2 = (\mathbb{B}^*, \epsilon, S_0, S_1)$ where S_i is the i-th successor function: $S_0(u) = u\,0$ and $S_1(u) = u\,1$ for $u \in \mathbb{B}^*$. The *theory S2S* is the set of S2S-sentences that are true in $\mathbf{t}_2$.

Semantics of S2S S2S-formulas $\varphi(X_1, \cdots, X_n)$, with free second-order variables from $X_1, \cdots, X_n$, are interpreted in expanded structures $\widehat{t} = (\mathbf{t}_2, P_1, \cdots, P_n)$. We write

$$\widehat{t} \vDash \varphi(X_1, \cdots, X_n)$$

just if $\widehat{t}$ satisfies $\varphi(\overline{X})$. We identify $\widehat{t}$ with the infinite tree $t \in \mathfrak{T}^{\omega}_{\mathbb{B}^n}$ whereby for each $u \in \mathbb{B}^*$, we have

$$t(u) = (b_1, \cdots, b_n) \quad \text{where } b_i = 1 \leftrightarrow u \in P_i.$$

Given an S2S formula $\varphi(\overline{X})$ the *tree language* defined by $\varphi(\overline{X})$ is the set

$$L(\varphi) := \{ t \in \mathfrak{T}^{\omega}_{\mathbb{B}^n} \mid \widehat{t} \vDash \varphi \}.$$

Example 6.5. Consider the S2S formula $\varphi(X_1) := \exists Y.\mathsf{infinite}(Y) \wedge \forall y.(Y(y) \to X_1(y))$, where $\mathsf{infinite}(Y)$ is an S2S formula expressing that Y is an infinite set of nodes.

Then $L(\varphi) := \{ t \in \mathfrak{T}^{\omega}_{\mathbb{B}} : \widehat{t}$ has infinitely many positions where P_1 holds $\}$.

Rabin (1969) showed a connection between NPT and S2S over infinite trees that parallels the connection between NBW and S1S in the case of infinite words.

Theorem 6.7. *A tree language is S2S-definable if and only if it is recognisable by a NPT.*

The proof proceeds in a similar fashion as in the word case. Given an NPT, we can write an equivalent S2S-formula: the formula asserts the existence of a run of the parity automaton that satisfies the parity condition.

Likewise, given an S2S-formula, we can construct an equivalent automaton by induction on the structure of the formula, taking advantage of the closure properties proven in the previous section to deal with the different operators in the logic.

One important consequence of this connection between the logic S2S and NPT, is the decidability of satisfiability for S2S, first shown by Rabin (1969).

Theorem 6.8 (Rabin Tree Theorem)**.** *The theory S2S is decidable.*

The Logical System WS2S The logical system *WS2S* (weak monadic second-order logic of 2 successors) has the same syntax as S2S, but quantified second-order variables range only over finite sets of positions.

Example 6.6. Consider the WS2S formula $\varphi(X_1) := \forall Y.\exists z.\neg Y(z) \wedge X_1(z)$.

You can read this as saying, for all *finite* sets of nodes Y, there exists a node z outside of Y where P_1 holds.

In other words, this express that there are infinitely many P_1 positions, so this is a way to define the language from Example 6.5 in WS2S.

Every WS2S formula can be expressed in S2S, because it is possible to write an S2S formula $\mathsf{finite}(X)$ that asserts that some set of nodes is finite. However, not every S2S formula can be translated into WS2S.

Indeed, WS2S can be characterized as follows, based on the work of Rabin (1970) and Muller et al. (1986).

Theorem 6.9. *The following are equivalent for a tree language $T \subseteq \mathfrak{T}^{\omega}_{\mathbb{B}^n}$:*

- *T is definable in WS2S;*
- *T is recognisable using AWT;*
- *T and $\mathfrak{T}^{\omega}_{\mathbb{B}^n} \setminus T$ are recognisable using NBT.*

Theorem 6.10. *The theory WS2S is decidable.*

Problems

6.1 Construct a non-deterministic Büchi tree automaton for each of the following languages:

(a) $\{\, t \in \mathfrak{T}^{\omega}_{\{a,b\}} : t \text{ has exactly one } a \,\}$

(b) $\{\, t \in \mathfrak{T}^{\omega}_{\{a,b\}} : \text{every branch in } t \text{ has at most one } a \,\}$

6.2 Prove that every non-deterministic Muller tree automaton can be converted into an equivalent non-deterministic parity tree automaton. [You may use the fact that deterministic parity automata are equivalent to deterministic Muller automata over ω-words.]

6.3 Construct an alternating parity tree automaton for each of the following tree languages:

(a) $\{\, t \in \mathfrak{T}^{\omega}_{\{a,b\}} : \text{there is some branch in } t \text{ with finitely many } a\text{'s} \,\}$.

(b) $\{\, t \in \mathfrak{T}^{\omega}_{\{a,b\}} : t \text{ has finitely many } a\text{'s} \,\}$.

You should formally define the transition function for the automaton, and then explain the role of both players as in Example 6.4.

What would change in order to define alternating automata for the complements of these languages?

6.4 Prove that alternating parity tree automata are closed under union and intersection.

6.5

(a) Write an S2S formula *on-branch*(X) that expresses that X is a set of positions along a single branch in the tree.

(b) Write an S2S formula *finite-on-branch*(X) that expresses that X is a finite set of positions along a single branch in the tree.

(c) Write an S2S formula $\varphi(X_1)$ defining

$$T_2 := \{\, t \in \mathfrak{T}^{\omega}_{\mathbb{B}} : \text{every branch of } t \text{ has finitely many positions where } P_1 \text{ holds} \,\}.$$

6.6 Write a WS2S formula $\varphi(X_1, X_2)$ defining

$$\begin{aligned} T := \{\ t \in \mathfrak{T}^{\omega}_{\mathbb{B}^2} : {} & \text{below every position in } t \text{ where } P_1 \text{ holds,} \\ & \text{there are finitely many positions where } P_2 \text{ holds } \}. \end{aligned}$$

Bibliography

J. Bradfield and C. P. Stirling. Modal logics and mu-calculi. In A. Ponse and S. Smolka, editors, *Handbook of Process Algebra*, pages 293–332. Springer-Verlag, 2001. 4

J. Bradfield and C. P. Stirling. Modal mu-calculi. In A. Ponse and S. Smolka, editors, *Handbook of Modal Logic*, pages 721–756. 2007. Studies in Logic and Practical Reasoning Volume 3. 0.1, 4, 5

J. R. Büchi. Weak second order arithmetic and finite automata. *Zeitschrift für Maths. Logik und Grundlagen Maths.*, 6:66–92, 1960a. 3

J. R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. International Congress on Logic, Methodology and Philosophy of Science*, pages 1–11. Stanford Univ. Press, 1960b. 1.2

J. R. Büchi. Weak second order arithmetic and finite automata. In *Proc. Int. Congr. Logic, Methodology and Philosophy of Science*, pages 1–12. Stanford University Press, 1962. 3

J. Richard Buchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:pp. 295–311, 1969. ISSN 00029947. URL http://www.jstor.org/stable/1994916. 5.5

Thomas Colcombet and Konrad Zdanowski. A tight lower bound for determinization of transition labeled büchi automata. In *ICALP(2)*, pages 151–162, 2009. 5

C. C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–52, 1961. 3

E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *FOCS*, pages 368–377, 1991. 6.2, 6.2

H. B. Enderton. *Elements of Set Theory*. Academic Press, 1977. A

Paul Gastin and Denis Oddoux. Fast LTL to büchi automata translation. In *CAV*, pages 53–65, 2001. 5

Erich Grädel, Wolfgang Thomas, and Thomas Wilke. *Automata, Logics and Infinite Games*. Springer-Verlag, 2002. LNCS Vol. 2500. 0.1, 5.5

Y. Gurevich and L. Harrington. Tree, automata and games. In *STOC*, pages 60–65, 1982. 5.6, 6.2

Moritz Hammer, Alexander Knapp, and Stephan Merz. Truly on-the-fly LTL model checking. In *TACAS*, pages 191–205, 2005. 5

Neil Immerman. *Descriptive Complexity*. Springer, New York, 1999. Graduate Texts in Computer Science. 1.4, 1.2

H. W. Kamp. *The temporal logic of programs*. PhD thesis, University of California, Los Angeles, 1968. 3

B. Khoussainov and A. Nerode. *Automata Theory and its Applications*, volume 21 of *Progress in Computer Science and Applied Logic*. Birkhäuser, 2001. 0.1

D. Kozen. *Theory of Computation*. Springer-Verlag, 2006. 4

Wanwei Liu and Ji Wang. A tighter analysis of piterman's büchi determinization. *Inf. Process. Lett.*, 109:941–945, 2009. 3

Christof Löding. Optimal bounds for transformations of omega-automata. In *FSTTCS*, pages 97–109, 1999. 1.5

Christof Löding. Automata on infinite trees, December 2011. Available at `http://automata.rwth-aachen.de/~loeding/inf-tree-automata.pdf`,. 6, 6.2, 6.3

Christof Löding and Wolfgang Thomas. Alternating automata and logics over infinite words. In *IFIP TCS*, pages 521–535, 2000. 5

D. A. Martin. Borel determinacy. *The Annals of Mathematics*, 102(2):363–371, 1975. 6.2

R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9:521–530, 1966. 1.5

A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential time. In *Proc. 13th IEEE Symp. on Swithcing and Automata Theory*, pages 125–129, 1972. 1.4

M. Michel. Complementation is more difficult with automata on infinite words. Technical report, CNET, Paris, 1988. 1.5

A. W. Mostowski. Games with forbidden positions. Technical Report 78, 1991. 6.2, 6.2

D. Muller. Infinite sequences and finite machines. In *Proc. 4th Ann. IEEE Symp. Switching Circuit Theory and Logical Design*, pages 3–16, 1963. 3

David E. Muller and Paul E. Schupp. Alternating automata on infinite trees. *Theor. Comput. Sci.*, 54:267–276, 1987. 6, 6.3

David E. Muller, Ahmed Saoudi, and Paul E. Schupp. Alternating automata. the weak monadic theory of the tree, and its complexity. In *ICALP*, pages 275–283, 1986. 6, 6.5

D. Niwinski and I. Walukiewicz. Games for the mu-calculus. *Theoretical Computer Science*, 163:99–116, 1997. 5

C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994. 2.5, 2.5

J. P. Pécuchet. On the complementation of büchi automata. *Theoretical Computer Science*, 47:95–98, 1986. 1.4

Nir Piterman. From nondeterministic büchi and streett automata to deterministic parity automata. *Logical Methods in Computer Science*, 3, 2007. 2

Amir Pnueli. The temporal logic of programs. In *Proc. 18th IEEE Symp. Found. of Comp. Sci.*, pages 46–57, 1977. 2.1

M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Maths. Soc*, 141:1–35, 1969. 6, 6.2, 6.5, 6.5

Michael Rabin. Weakly definable relations and special automata. In *Mathematical Logic in Foundations of Set Theory*, pages 1–23. 1970. 6, 6.5

A. Rabinovich. A proof of Kamp's theorem. In *CSL*, pages 516–527, 2012. 3

S. Safra. On the complexity of ω-automaton. In *Proc. 29th IEEE Symp. Foundations of Comp. Sc.*, pages 319–327, 1988. 1.5, 1

Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4:177–192, 1970. 2.5

Sven Schewe. Tighter bounds for the determinisation of büchi automata. In *FoSSaCS*, pages 167–181, 2009. 1.5, 1.5, 1.6, 4

Michael Sipser. *Introduction to the Theory of Computation*. Thomson Course Technology, 2005. 1.4, 2.5

A. P. Sistla, M. Y. Vardi, and P. Wolper. The complementation problem for büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987. 1.4, 1.4

A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985. 2.5

C. P. Stirling. Bisimulation, model checking and other games. Notes for Mathfit instructional meeting on games and computation, Edinburgh, 1997. 4, 5

C. P. Stirling. *Modal and Temporal Properties of Processes*. Springer-Verlag, 2001. Texts in Computer Science. 0.1, 4, 5

Robert S. Streett and E. Allen Emerson. An automata theoretic decision procedure for the propositional mu-calculus. *Inf. Comput.*, 81(3):249–264, 1989. 5, 5.1

R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Computing*, 1:146–160, 1972. 1.4

W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B*, pages 134–191. Elsevier, 1990. 0.1, 1.2

W. Thomas. Languages, automata and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3. Springer-Verlag, 1997. 0.1

M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Proceedings of Banff Higher Order Workshop*, pages 238–266. Springer-Verlag, 1996. LNCS Vol. 1043. 0.1

M. Y. Vardi. Branching *vs* linear time: final showdown. In *ETAPS 2001*. Springer-Verlag, 2001. 1

Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998. 6.2

Appendix A

Ordinals and Transfinite Induction: A Primer

Everybody is familiar with the set $\omega = \{ 0, 1, 2, \cdots \}$ of *finite ordinals* or *natural numbers.* A standard mathematical tool is the *Principle of Mathematical Induction* on this set. How does one count beyond the finite ordinals? Is there an associated induction principle?

Ordinals are certain sets of sets.

- There are two kinds: *successors* and *limits.*
- They are *well-ordered.*
- There are a lot of them.
- We can do induction on them, using the *Principle of Transfinite Induction.*

A good reference is Enderton's book (Enderton, 1977).

We assume the axioms of Zermelo-Fraenkel set theory. It is impossible to understand ordinals and transfinite induction properly outside the context of set theory.

Definition A.1. (i) A set C of sets is said to be *transitive* just if $A \in C$ whenever $A \in B$ and $B \in C$. Equivalently C is transitive if every element of C is also a subset of C i.e. $C \subseteq 2^C$.

(ii) An *ordinal* is defined to be a set A such that A is transitive, and every element of A is transitive. We use $\alpha, \beta, \gamma, \cdots$ to refer to ordinals.

There are several equivalent definitions of ordinals. It follows that

(i) every element of an ordinal is an ordinal

(ii) every transitive set of ordinals is itself an ordinal.

The collection of all ordinals is not a set, but a proper *class.*

Ordinals are well-ordered A binary relation $<$ on a set A is a *linear order* if it satisfies:

(i) Irreflexivity: $\forall x \in A \,.\, \neg(x < x)$

(i) Transitivity: $\forall x, y, z \in A \,.\, x < y \;\wedge\; y < z \;\rightarrow x < z$

(i) Trichotomy: $\forall x, y \in A \,.\, x < y \;\vee\; y < x \;\vee\; x = y$

A binary relation $<$ on a class S is *well-founded* just if every non-empty subset of S has a $<$-minimal element.

For ordinals α, β, define $\alpha < \beta$ just if $\alpha \in \beta$. It follows that every ordinal is equal to the set of all smaller ordinals i.e. $\alpha = \{ \beta : \beta < \alpha \}$.

Proposition 11. *(i) $<$ is a well-founded linear ordering on the class of ordinals.*

(ii) If α is an ordinal, so is $\alpha \cup \{ \alpha \}$, called the successor ordinal *of α, which is denoted $\alpha + 1$.*

(iii) If A is a set of ordinals then $\bigcup A$ is an ordinal; and it is the supremum of the ordinals in A under $\leq$.

An ordinal is called a *successor ordinal* if it is of the form $\alpha + 1$; otherwise it is a *limit ordinal*. For example, the smallest few ordinals are

$$\begin{aligned} 0 &:= \varnothing \\ 1 &:= \{ \varnothing \} \\ 2 &:= \{ \varnothing, \{ \varnothing \} \} \\ 3 &:= \{ \varnothing, \{ \varnothing \}, \{ \varnothing, \{ \varnothing \} \} \} \\ &\vdots \end{aligned}$$

The first infinite ordinal is $\omega := \{ 0, 1, 2, \cdots \}$. The smallest limit ordinal is 0, the next smallest is ω. In fact there are uncountably many countably infinite ordinals:

$$\begin{aligned} &\omega, \omega + 1, \omega + 2, \cdots, \omega + \omega = \omega \cdot 2, \cdots, \omega \cdot 3, \cdots, \\ &\omega \cdot \omega = \omega^2, \cdots, \omega^3, \cdots, \omega^\omega, \cdots, \omega^{\omega^\omega}, \cdots, \omega^{\omega^{\omega^\omega}}, \cdots, \\ &\epsilon_0 = \omega^{\omega^{\omega^{\omega^{\cdots}}}}, \cdots \end{aligned}$$

The set of all countable ordinals is the first uncountable ordinal, written ω_1.

Principle of Transfinite Induction To prove that "for all ordinals α, the property H_α holds", we establish the following:

(i) *Successor ordinal* $\alpha + 1$: Assuming that H_α holds, then $H_{\alpha+1}$ holds.

(ii) *Limit ordinal* λ: Assuming that H_α holds for all $\alpha < \lambda$, then H_λ holds.

The validity of the principle ultimately rests on the well-foundedness of the relation $\in$.