Diploma Thesis

Games for the Linear Time μ-Calculus

Christian Dax

Supervisor

Dr Martin Lange

Submitted to

Prof Dr Martin Hofmann
Chair of Theoretical Computer Science
Ludwig-Maximilians-University Munich

1.10.2005–31.3.2006

Set in LaTeX

Abstract

The Linear Time μ-Calculus (μTL) is a temporal logic for specifying ω-regular properties of a system. In this work, we consider game-theoretic characterizations of the model-checking, satisfiability and validity problem for the μTL logic. Using an automaton based approach to encode the winning conditions of the games, a decision procedure is developed which solves these problems in PSPACE.

Acknowledgements

I would like to thank my supervisor Dr Martin Lange for his excellent support during the last years. He helped me to organize my stay abroad at the University of Edinburgh, he replied to all of my numerous (sometimes stupid) email-questions at once, he showed the patience of a saint whenever I got private lessons at his white board and furthermore, he was the key for my future position at the ETH in Zurich.

Further thanks go to Prof Martin Hofmann whose idea of using automata for detecting ν-lines belongs to the main results of this work. With his constant "background-support" he provided us with several proof sketches. At this point, I want to thank him for all the letters of recommendation he wrote for me and for the acceptance of all the unconventional subjects I choosed for the diploma examination.

Then, I would also like to thank Dr Jan Johannsen for his advice and his offer to examine this work.

Finally, I would like to express my thanks to my parents for their constant support. They gave me the opportunity to concentrate on my study and kept almost all daily-life problems away from me.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Christian Dax*)

Contents

1 Introduction

> There are sadistic scientists who hurry to hunt down errors instead of establishing the truth.
>
> *(Marie Curie)*

Formal Verification

In the modern world, computer systems play an important part in our lives and it seems that their significance in daily technology around us will increase steadily. Simultaneously, we are becoming more and more dependent on these electronic devices in e.g. transportation systems, medical applications or banking. While loss of money due to software errors might be unfortunate for people involved, the cost of failure in a medical operation for example can become unacceptably high.

Several incidents in recent history tell us that fatal design errors occur once in a while. In 1994 Thomas Nicely discovered a bug in the Pentium floating point unit which causes wrong values at certain division operations. Two years later the maiden flight of unmanned Ariane 5 rocket ended in a firework about forty seconds after its lift-off because of a malfunction in the control software. In 2000, nearly thirty cancer patients at the National Cancer Institute in Panama City received overdoses of radiation due to miscalculation by the software. Eight patients died and their physicians were indicated for murder. For more details on these examples see [Klo05] or [Gar05].

To avoid troubles in critical applications, developers commonly try to ensure correctness of their work through simulation or testing. However, in practice, systems tend to be large and too complex to be thoroughly tested and therefore these methods are especially used to detect only well-defined types of faults. So, as McFarland writes in [McF93], subtle design errors resulting in unexpected behaviour might be missed.

Another approach to guarantee that software functions correctly is formal verification. As depicted in Table 1.1 its idea is to model behaviour of complex systems by simple, abstract mathematical structures, e.g. words, trees or labelled transition systems (LTS). The specification is translated (from English) into a simple but mathematical precise formal specification language s.t. properties required according to the specification can be concisely represented by automata, logical formulas, etc. Having lifted up both the behaviour of the system and the specification to an abstract level algorithmic methods can be applied to find errors.

Besides, formal verification may allow designers to reduce developing cost. Beginning at the abstract level algorithms and specifications can be checked before they turn into real products. Hence, this approach can effectively reduce the number of updates for repairing faulty versions of the software.

real world:	complex system	fulfils	specification
	↑↓		↑↓
abstraction:	math. structure	fulfils	formal spec. (automaton/ logical formula)

Table 1.1: Idea behind formal verification

Temporal Logics

Formal verification goes back to the 1960s where computer programs were viewed as computing functions of sequential input-output models. Floyd-Hoare logics [Hoa83] provide a framework to make assertions about the inputs of such programs and to verify these by a proof system.

In contrast, different theories have been developed to model concurrent, reactive systems (e.g. operating systems, protocols or applications with user interaction). The behaviour of these programs is a possibly non-terminating computation with interaction between the system and its environment. Computer scientists represent these types of programs by infinite linear or branching mathematical structures like words or trees. In terms of time, these structures can be seen as time-lines with states and properties holding at these states.

Temporal logics have been proven to be suitable to describe properties of linear and branching time-lines. Pnueli introduced Linear Time Temporal Logic (LTL)[Pnu77] for specifying and verifying concurrent systems. Properties of a system are turned into questions of satisfiability or validity in temporal logic. This approach is called *model checking*. LTL extends an underlying propositional logic by temporal operators next, until and release and is interpreted over a linear time structure. A counterpart for describing branching time structures is for example Computation Tree Logic (CTL) which has been presented in [EH81] and [EH85].

If we extend LTL by adding minimal and maximal fixed points, we obtain the Linear Time μ-Calculus (μTL), the logic of this work. It has been introduced in [Var88] (with past operators) and [BKP86] and like LTL, it describes properties of linear time structures. Kozen's Modal μ-Calculus [Koz83] where the next operator of μTL is replaced by two different modal operators is interpreted over branching time structures. This logic has become widely investigated since though its syntax and semantics are simple it has enhanced expressive power compared to LTL.

Several formal languages have been examined in which properties of mathematical structures can be more or less defined. Generally speaking, algorithms can handle languages which are less expressive more easily than complex ones. On the other hand, the language for the formal specification must be expressive enough to be able to describe properties which are required by the specification.

A formal specification given by an LTL formula belongs to the class of star-free languages which is less than the class of ω-regular languages. For more detail on star-free languages and LTL see [Kam68, GPSS80, GPSS53, Tho79]. In comparison to LTL, the fixed point logic μTL is capable of expressing ω-regular properties, see [Lan05, JW96].

Games

The main topic in this thesis is to find game-theoretic characterizations of model-checking, satisfiability and validity problem for the μTL logic. We use linear time structures to represent programs and μTL formulas to encode specifications of such programs.

The first approach is to check whether a single *run* of a program – represented by an infinite word w – fulfils the μTL formula φ. We will write

$$w \overset{?}{\models} \varphi.$$

If for all possible runs w' of the program the relationship $w' \models \varphi$ holds, we assume the program to be correct.

Suppose our algorithm tells us that the specification φ does not hold for some run w'. That is, there must be an error in the system which has to be repaired. Hence, it might be helpful if our algorithm could report where this error occurs and why.

Games provide a natural framework to fulfil this feature. The idea of using games to characterize model checking problems is due to Stirling. These kind of games consist of two players, namely *Eliza* and *Albert*, competing each other. Player *Eliza* tries to show that a formula holds whereas her opponent wants the opposite. Winning a game means having a winning strategy which can be used to isolate an error of the system by an interactive play against the designer, for example.

Synopsis

In chapter 2 we summarize necessary preliminaries of mathematical structures and lemmas to provide a fundamental background to understand the following chapters.

Chapter 3 introduces model checking for linear infinite structures such as infinite words. The games for this task have been developed by Stirling [Sti95, Sti97] and a closer look into that subject will support comprehension of games for tree structures.

The heart of this work is contained in chapter 4. In [SW91] Stirling and Walker examined tableaux for solving the model checking problem for trees. Bradfield, Esparza and Mader continued their work and presented a tableau for satisfiability checking [BEM96]. The games of this thesis are based on their work and improve the winning conditions (which are the equivalent to the abortion conditions in tableaux).

Kaivola investigated the satisfiability problem for the μ-calculus using tableaux [Kai95, Kai97]. His approach is related to the games we used for model checking for trees. In chapter 5, we simply extract satisfiability and validity checking games from the tree games using a "universal tree".

In chapter 6 an automata based algorithm is presented for deciding the winner of a tree game. Then we will estimate the complexity of all games discussed in this work.

In mathematics you don't understand things. You just get used to them.

(von Neumann)

2 Preliminaries

2.1 Infinite Words and Infinite Trees

In formal verification Kripke structures or Labeled Transition Systems (LTS) are used to abstract the behaviour of non-terminating systems. Infinite words and infinite trees provide a similar mathematical structure and since they are simple and directly connected to language classes which have been widely examined, we choose them to be our mathematical interpretations of real systems.

Definition 2.1 (Infinite Words) Let $\Sigma = \{a, b, c, ...\}$ be a finite non-empty alphabet. An *infinite word (or ω-word) over* Σ is a total map $w : \mathbb{N} \to \Sigma$.

We will usually write $w := w(0)\, w(1)\, w(2) \dots$ for such a word and $w^{[i} := w(i)\, w(i+1)\, w(i+2) \dots$ for its suffix beginning at position $i \in \mathbb{N}$. Its prefix ending at position i is denoted by $w^{i]}$.

The *set of all infinite words over* Σ is denoted by Σ^ω.

Let $v \in \Sigma^*$ be a finite word. To represent the infinite sequence $vvvv\dots$ we write $v^\omega \in \Sigma^\omega$. □

Example 2.2

- $w = abaabaaabaaab\dots$ is an infinite word.
- Let $w = abcdefg(abc)^\omega$. Then $w^{[8} = bc(abc)^\omega$ and $w^{8]} = abcdefgab$.

An infinite word can be seen as a labeling of the linear and infinite sequence $\mathbb{N} = 0\,1\,2\,3\,4\,5\dots$ and if we represent $\mathbb{N}$ by a unary numeral system using one symbol, e.g. 0, this sequence is of the form $\mathbb{N} = 0\,00\,000\,0000\dots$ where each position in N is a word in $\{0\}^*$.

A tree can be defined as a labeling of linear (and infinite) branches beginning at the root of the tree. In a binary tree such a branch w can be specified by a sequence of $w = \varepsilon\,0\,1\,1\dots$ where ε is the root of the tree and 0 means left and 1 means right.

Definition 2.3 (Infinite Trees) A set of finite words $\mathbb{D} \subseteq \mathbb{N}^*$ is called *tree domain* if it satisfies

- $\mathbb{D}$ is prefix closed (in particular: $\varepsilon \in \mathbb{D}$),
- $\forall d \in \mathbb{D} : d0 \in \mathbb{D}$,

- $\forall d \in \mathbb{D}, \forall i+1 \in \mathbb{N} : d(i+1) \in \mathbb{D} \Rightarrow di \in \mathbb{D}$.

Let $\Sigma = \{a, b, c, ...\}$ be a finite non-empty alphabet. An *infinite tree over* Σ is a map $t : \mathbb{D} \to \Sigma$.

Each element of $\mathbb{D}$ is called *node* and $\varepsilon \in \mathbb{D}$ is called *root* of the tree. Furthermore, $di \in \mathbb{D}$ is a *child* of its *parent* $d \in \mathbb{D}$.

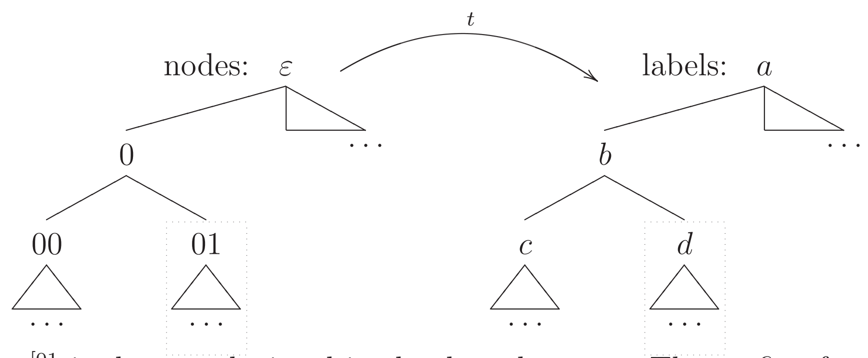
A *branch in* t is an infinite word $p \in \mathbb{D}^\omega$ s.t. for every $i \in \mathbb{N} : p(i+1)$ is a child of $p(i)$. We write $w \in t$ and call it a *path* in t iff there is a branch p in t s.t. $w(i) = t(p(i))$ for all $i \in \mathbb{N}$.

Each node $n \in \mathbb{D}$ defines a subtree $t^{[n} : \mathbb{D}^{[n} \to \Sigma$ of t which begins at node n. Formally, $\mathbb{D}^{[n} := \{d \mid nd \in \mathbb{D}\}$ and $t^{[n}(d) := t(nd)$ for all $d \in \mathbb{D}^{[n}$. The prefix of t which ends in node n is a word $w \in \Sigma^*$ where $w(i) := t(n^{i]})$ for all $i = 0, 1, \ldots, |n|$. It is denoted by $t^{n]}$.

The *set of all infinite trees over* Σ is denoted by $\mathcal{T}_\Sigma$. $\square$

Example 2.4

- Every infinite word is a tree with only one branch $w \in \mathbb{D} = \{0\}^*$, e.g. words of example 2.2.
- An illustration of a binary tree $t : \mathbb{D} \to \Sigma$ where $\mathbb{D} = \{0, 1\}^*$ is



where $t^{[01}$ is the tree depicted in the dotted square. The prefix of t which ends in node 01 is $t^{01]} = abd$.

2.2 Definition of μTL

Definition 2.5 (Syntax) Let $\Sigma = \{a, b, c, \ldots\}$ be a finite non-empty alphabet and $\mathcal{V} = \{X, Y, Z, ...\}$ be a set of variables. A μTL formula in *positive normal form* is defined by the following grammar:

$$\varphi ::= a \mid X \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid O\varphi \mid \mu X.\varphi \mid \nu X.\varphi$$

where $a \in \Sigma$ and $X \in \mathcal{V}$. The connectives $\wedge$ and $\vee$ are called conjunctions and disjunction. The operator O is called *next* and the binders $\mu X.\psi$ and $\nu X.\psi$ denote *least/greatest fixed points*.

We will often refer to the μ- or ν-binder by σ. $\square$

Definition 2.6 The formulas *true* and *false* are abbreviated by $\mathtt{tt} := \nu X.OX$ und $\mathtt{ff} := \mu X.OX$, where $X \in \mathcal{V}$. $\square$

Definition 2.7 (Subformulas) The set of subformulas $Sub(\varphi)$ of a μTL formula φ is inductively defined as

$$\begin{aligned}
Sub(a) &:= a, \quad \text{for all } a \in \Sigma \\
Sub(X) &:= X, \quad \text{for all } X \in \mathcal{V} \\
Sub(\varphi_1 \wedge \varphi_2) &:= \{\varphi_1 \wedge \varphi_2\} \cup Sub(\varphi_1) \cup Sub(\varphi_2) \\
Sub(\varphi_1 \vee \varphi_2) &:= \{\varphi_1 \vee \varphi_2\} \cup Sub(\varphi_1) \cup Sub(\varphi_2) \\
Sub(\mu X.\varphi) &:= \{\mu X.\varphi\} \cup Sub(\varphi) \\
Sub(\nu X.\varphi) &:= \{\nu X.\varphi\} \cup Sub(\varphi).
\end{aligned}$$ $\square$

Definition 2.8 An occurrence of a variable X in a μTL formula φ is *bound* iff there is a $\sigma X.\psi \in Sub(\varphi)$ s.t. $X \in Sub(\psi)$. Otherwise it is *free*. A formula is *closed* iff it has no free variables.

A bound variable is of *type* μ iff its binder is $\mu X.\psi$. It is also called μ-variable. Otherwise it is of type ν.

For two variables $X, Y \in Sub(\varphi)$ we write $X <_\varphi Y$ iff Y is free in some $\sigma X.\psi \in Sub(\varphi)$. $\square$

Example 2.9

- In $\varphi := \mu X.\nu Y.X \wedge O(Y) \vee Z$ the variables X and Y are bound and variable Z is free. X is of type μ, Y is of type ν and the type of Z is unknown.
- The following table

	V	X	Y	Z
V				
X				
Y		$<_\varphi$		
Z	$<_\varphi$			

depicts the $<_\varphi$ relation for all variables in $\varphi := (\mu X.\nu Y.X \wedge Y) \vee (\mu Z.V \wedge Z)$.

Definition 2.10 (Substitution) Formula $\varphi[\psi/X]$ is defined as the formula, where all free occurrences of the variable X are simultaneously substituted by ψ.

Definition 2.11 (well-named) A μTL formula φ consisting of m μ-variables and n ν-variables is *well-named* if

- every variable in φ is bound at most once, and

- all μ-variables are renamed to $X_1, X_2, \dots, X_m$ s.t. $\forall i, j \in \{1, 2, \dots, m\} : X_i <_\varphi X_j \Rightarrow i > j$, and
- all ν-variables are renamed to $Y_1, Y_2, \dots, Y_n$ s.t. $\forall i, j \in \{1, 2, \dots, n\} : Y_i <_\varphi Y_j \Rightarrow i > j$.

For every formula φ in normal form we can define a mapping $\mathit{fp}_\varphi : \mathcal{V} \cap \mathit{Sub}(\varphi) \to \mathit{Sub}(\varphi)$, where fp_φ maps each X to its unique binder $\sigma X.\psi$. Analogously, $\mathit{fb}_\varphi : \mathcal{V} \cap \mathit{Sub}(\varphi) \to \mathit{Sub}(\varphi)$ maps each variable X to its unique fixed point body ψ. $\Box$

Any μTL formula can be transformed into a well-named form by renaming variables. This is possible since by definition of $<_\varphi$ for any two distinct variables $X <_\varphi Y$ and $Y <_\varphi X$ cannot hold simultaneously. Otherwise X would be in $\mathit{Sub}(\mathit{fp}_\varphi(Y))$ and $Y \in \mathit{Sub}(\mathit{fp}_\varphi(X))$. By now, we may assume that all formulas in this work are well-named.

Definition 2.12 (guarded form) A μTL formula φ is in *guarded form* iff every occurrence of a bound variable $X \in \mathit{Sub}(\varphi)$ is in the scope of an O operator which itself is in $\mathit{fb}_\varphi(X)$. $\Box$

Every μTL formula can be translated into guarded form by only a quadratic blow-up. See for example [Mat02] and [Wal00]. From now on, we assume that all μTL formulas are in guarded form.

Definition 2.13 (Semantics) The semantics of a μTL formula is inductively defined over infinite words in Σ^ω. Formulas with free variables are interpreted with respect to an *environment* $\rho : \mathcal{V} \to 2^{\mathbb{N}}$ which maps all free variables to positions $S \subseteq \mathbb{N}$. Besides, we write $\rho[Y \mapsto S]$ meaning that only the mapping for variable Y is changed to S.

$$\begin{aligned}
[\![a]\!]^w_\rho &:= \{i \in \mathbb{N} \mid w(i) = a\} \\
[\![X]\!]^w_\rho &:= \rho(X) \\
[\![\varphi_1 \vee \varphi_2]\!]^w_\rho &:= [\![\varphi_1]\!]^w_\rho \cup [\![\varphi_2]\!]^w_\rho \\
[\![\varphi_1 \wedge \varphi_2]\!]^w_\rho &:= [\![\varphi_1]\!]^w_\rho \cap [\![\varphi_2]\!]^w_\rho \\
[\![O\varphi]\!]^w_\rho &:= \{i \in \mathbb{N} \mid i + 1 \in [\![\varphi]\!]^w_\rho\} \\
[\![\mu X.\varphi]\!]^w_\rho &:= \bigcap \{S \subseteq \mathbb{N} \mid [\![\varphi]\!]^w_{\rho[X \mapsto S]} \subseteq S\} \\
[\![\nu X.\varphi]\!]^w_\rho &:= \bigcup \{S \subseteq \mathbb{N} \mid S \subseteq [\![\varphi]\!]^w_{\rho[X \mapsto S]}\}
\end{aligned}$$ $\Box$

Definition 2.14 (Model) An infinite word $w \in \Sigma^\omega$ together with an environment ρ is a *model* of a μTL formula φ iff the formula holds at position 0. We write

$$w \models_\rho \varphi \quad :\Leftrightarrow \quad 0 \in [\![\varphi]\!]^w_\rho$$

A tree $t \in \mathcal{T}_\Sigma$ is a model of φ iff all paths in t are models of that formula, i.e.

$$t \models_\rho \varphi \quad :\Leftrightarrow \quad \forall w \in t : w \models_\rho \varphi$$

Two formulas are *equivalent* iff they have exactly the same word-models. We write

$$\varphi \equiv \psi \quad :\Leftrightarrow \quad \forall w \in \Sigma^\omega : w \models_\rho \varphi \Leftrightarrow w \models_\rho \psi.$$

Notice that two equivalent formulas have exactly the same tree-models, as well.
For closed formulas we commonly drop ρ. □

Definition 2.15 A closed μTL formula φ is called

- *satisfiable* $\quad :\Leftrightarrow \quad \exists w \in \Sigma^\omega : w \models \varphi$
- *valid* $\quad :\Leftrightarrow \quad \forall w \in \Sigma^\omega : w \models \varphi$. □

EXAMPLE 2.16

- The μTL formula $\varphi := a \wedge Ob \wedge O(Oc \vee Od)$ specifies the following property: At the first position a holds, one step later b holds and at position four either c or d holds. Another way to understand this formula is: φ shall hold at the first position.
- A property φ which shall hold at every position (in LTL "generally φ" or $G(\varphi)$) can be described by the μTL formula $\nu Y.\varphi \wedge O(Y)$.

  Intuitively, we can interpret this formula by substituting variable Y by $fb_\varphi(Y)$ *infinitely* often. The resulting formula looks like

  $$\varphi \wedge O(\varphi \wedge O(\varphi \wedge O(\varphi \wedge O(\varphi \wedge O(\varphi \wedge O(\ldots))))))$$

  and can be read step by step.
- The existence of a position satisfying φ (in LTL "finally φ" or $F(\varphi)$) can be expressed by $\mu X.\varphi \vee O(X)$.

  Intuitively, φ can be transformed to formula

  $$\varphi \vee O(\varphi \vee O(\varphi \vee O(\varphi \vee O(\ldots(\varphi \vee O(X)))))$$

  where variable X is substituted by $fb_\varphi(X)$ *finitely* many times.
- Property φ holds until property ψ is satisfied (in LTL $\varphi U \psi$) is usually denoted by formula $\mu X.\psi \vee (\varphi \wedge O(X))$.

Negated formulas

Definition 2.17 We extend μTL by introducing a new syntactical construct $\neg\varphi$ and interpret it as $[\![\neg\varphi]\!]^w_\rho := \mathbb{N} \setminus [\![\varphi]\!]^w_\rho$. Let $\mu TL^\neg$ denote this class of formulas which uses negation. □

Lemma 2.18 *For any ω-word $w \in \Sigma^\omega$, any ρ and any $\varphi \in \mu TL^\neg$:*

$$w \models_\rho \varphi \quad \Leftrightarrow \quad w \not\models_\rho \neg\varphi.$$

Proof

$$\begin{aligned} w \models_\rho \varphi \quad &\Leftrightarrow \quad 0 \in [\![\varphi]\!]^w_\rho \\ &\Leftrightarrow \quad 0 \notin [\![\neg\varphi]\!]^w_\rho \\ &\Leftrightarrow \quad w \not\models_\rho \neg\varphi \end{aligned}$$

■

Lemma 2.19 (Equivalences) *Let φ, ψ be $\mu TL^\neg$ formulas and a be a letter in the finite alphabet Σ. Then the following holds:*

a) $\neg a \equiv \bigvee_{b\in\Sigma, b\neq a} b$

b) $\neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi$

c) $\neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi$

d) $\neg(O\varphi) \equiv O\neg\varphi$

e) $\neg(\mu X.\varphi) \equiv \nu X.\neg(\varphi[\neg X/X])$

f) $\neg(\nu X.\varphi) \equiv \mu X.\neg(\varphi[\neg X/X])$

Proof Let w be an arbitrary ω-word.

a)

$$\begin{aligned} w \models \neg a &\Leftrightarrow w \not\models a \\ &\Leftrightarrow w(0) \neq a \\ &\Leftrightarrow \text{there is a } b \in \Sigma \text{ with } b \neq a \text{ s.t. } w(0) = b \\ &\Leftrightarrow \text{there is a } b \in \Sigma \text{ with } b \neq a \text{ s.t. } w \models b \\ &\Leftrightarrow w \models \bigvee_{b\in\Sigma, b\neq a} b \end{aligned}$$

b) , c) de Morgan

d)

$$\begin{aligned}
w \models \neg(O\varphi) &\Leftrightarrow 0 \in [\![\neg(O\varphi)]\!]^w_\rho \\
&\Leftrightarrow 0 \in \mathbb{N} \setminus [\![O\varphi]\!]^w_\rho \\
&\Leftrightarrow 0 \in \mathbb{N} \setminus \{i \in \mathbb{N} \mid i+1 \in [\![\varphi]\!]^w_\rho\} \\
&\Leftrightarrow 0 \in \{i \in \mathbb{N} \mid i+1 \notin [\![\varphi]\!]^w_\rho\} \\
&\Leftrightarrow 0 \in \{i \in \mathbb{N} \mid i+1 \in \mathbb{N} \setminus [\![\varphi]\!]^w_\rho\} \\
&\Leftrightarrow 0 \in \{i \in \mathbb{N} \mid i+1 \in [\![\neg\varphi]\!]^w_\rho\} \\
&\Leftrightarrow 0 \in [\![O(\neg\varphi)]\!]^w_\rho \\
&\Leftrightarrow w \models_\rho O(\neg\varphi)
\end{aligned}$$

e)

$$\begin{aligned}
w \models \neg(\mu X.\varphi) &\Leftrightarrow 0 \in [\![\neg(\mu X.\varphi)]\!]^w \\
&\Leftrightarrow 0 \in \mathbb{N} \setminus \bigcap\{S \subseteq \mathbb{N} \mid [\![\varphi]\!]^w_{\rho[X\mapsto S]} \subseteq S\} \\
&\Leftrightarrow 0 \in \bigcup\{\mathbb{N} \setminus S \subseteq \mathbb{N} \mid [\![\varphi]\!]^w_{\rho[X\mapsto S]} \subseteq S\} \\
&\Leftrightarrow 0 \in \bigcup\{\overline{S} \subseteq \mathbb{N} \mid [\![\varphi]\!]^w_{\rho[X\mapsto \mathbb{N}\setminus\overline{S}]} \subseteq \mathbb{N} \setminus \overline{S}\} \\
&\Leftrightarrow 0 \in \bigcup\{\overline{S} \subseteq \mathbb{N} \mid \overline{S} \subseteq \mathbb{N} \setminus [\![\varphi]\!]^w_{\rho[X\mapsto \mathbb{N}\setminus\overline{S}]}\} \\
&\Leftrightarrow 0 \in \bigcup\{\overline{S} \subseteq \mathbb{N} \mid \overline{S} \subseteq [\![\neg\varphi]\!]^w_{\rho[X\mapsto \mathbb{N}\setminus\overline{S}]}\} \\
&\Leftrightarrow 0 \in \bigcup\{\overline{S} \subseteq \mathbb{N} \mid \overline{S} \subseteq [\![\neg(\varphi[\neg X/X])]\!]^w_{\rho[X\mapsto \overline{S}]}\} \\
&\Leftrightarrow 0 \in \bigcup\{S \subseteq \mathbb{N} \mid S \subseteq [\![\neg(\varphi[\neg X/X])]\!]^w_{\rho[X\mapsto S]}\} \\
&\Leftrightarrow 0 \in [\![\nu X.\neg(\varphi[\neg X/X])]\!]^w \\
&\Leftrightarrow w \models \nu X.\neg(\varphi[\neg X/X])
\end{aligned}$$

f)

$$
\begin{aligned}
w \models \neg(\nu X.\varphi) &\Leftrightarrow 0 \in [\![\neg(\nu X.\varphi)]\!]^w \\
&\Leftrightarrow 0 \in \mathbb{N} \setminus \bigcup\{S \subseteq \mathbb{N} \mid S \subseteq [\![\varphi]\!]^w_{\rho[X\mapsto S]}\} \\
&\Leftrightarrow 0 \in \bigcap\{\mathbb{N} \setminus S \subseteq \mathbb{N} \mid S \subseteq [\![\varphi]\!]^w_{\rho[X\mapsto S]}\} \\
&\Leftrightarrow 0 \in \bigcap\{\overline{S} \subseteq \mathbb{N} \mid \mathbb{N} \setminus \overline{S} \subseteq [\![\varphi]\!]^w_{\rho[X\mapsto \mathbb{N}\setminus\overline{S}]}\} \\
&\Leftrightarrow 0 \in \bigcap\{\overline{S} \subseteq \mathbb{N} \mid \mathbb{N} \setminus [\![\varphi]\!]^w_{\rho[X\mapsto \mathbb{N}\setminus\overline{S}]} \subseteq \overline{S}\} \\
&\Leftrightarrow 0 \in \bigcap\{\overline{S} \subseteq \mathbb{N} \mid [\![\neg\varphi]\!]^w_{\rho[X\mapsto \mathbb{N}\setminus\overline{S}]} \subseteq \overline{S}\} \\
&\Leftrightarrow 0 \in \bigcap\{\overline{S} \subseteq \mathbb{N} \mid [\![\neg(\varphi[\neg X/X])]\!]^w_{\rho[X\mapsto \overline{S}]} \subseteq \overline{S}\} \\
&\Leftrightarrow 0 \in \bigcap\{S \subseteq \mathbb{N} \mid [\![\neg(\varphi[\neg X/X])]\!]^w_{\rho[X\mapsto S]} \subseteq S\} \\
&\Leftrightarrow 0 \in [\![\mu X.\neg(\varphi[\neg X/X])]\!]^w \\
&\Leftrightarrow w \models \mu X.\neg(\varphi[\neg X/X])
\end{aligned}
$$

■

Lemma 2.20 *Every μTL formula φ in positive normal form can effectively be transformed into a formula (in positive normal form) which is equivalent to its negation.*

Proof By induction on the structure of the negated formula $\neg\varphi$ where all equivalences of Lemma 2.19 are applied. ■

Definition 2.21 Let $\varphi \in \mu TL$ a formula in positive normal form. The unique formula returned by the procedure described in Lemma 2.20 is denoted by $\overline{\varphi}$. □

Unfortunately, $\overline{\cdot} : \mu TL \to \mu TL$ is not bijective and so $\overline{\overline{\varphi}}$ does not necessarily equal φ. On the other hand it is easy to see that φ can be inferred if its negation $\overline{\varphi}$ is given because $\overline{\cdot} : \mu TL \to \mu TL$ is injective.

2.3 Fixed Points

Definition 2.22 (Lattice) Let S be a *partially ordered set* with respect to $\leq$, i.e. the following holds:

- $\forall x \in S : x \leq x$ (reflexivity)
- $\forall x, y, z \in S : (x \leq y) \wedge (y \leq z) \Rightarrow x \leq z$ (transitivity)
- $\forall x, y \in S : (x \leq y) \wedge (y \leq x) \Rightarrow x = y$ (anti-symmetriy)

Let $A \subseteq S$ be a subset of S. A *supremum* of A, denoted as $\sqcup A$, is the least element $s \in S$ s.t. $\forall a \in A : a \leq s$. Dually, an *infimum* of A, denoted by $\sqcap A$, is the greatest element $s \in S$ s.t. $\forall a \in A : s \leq a$.

A pair $(S, \leq)$ is called *lattice* if $\sqcup\{x, y\}$ and $\sqcap\{x, y\}$ exist in S for all $x, y \in S$. A lattice is *complete* if suprema and infima exist for all subsets of S. In this case, define the two elements *bottom* $\perp := \sqcup S$ and *top* $\top := \sqcap S$. □

EXAMPLE 2.23

- In $\mathbb{R}$ the set of negative real numbers $\mathbb{R}^-$ has no greatest element, but its supremum $\sqcup\mathbb{R}^- = 0$ exists.
- $(\mathbb{N}, \leq)$ is a lattice with $\perp = 0$ and no top element.
- $(2^{\mathbb{N}}, \subseteq)$ is a complete lattice. For every subset $A \subseteq 2^{\mathbb{N}}$, $\bigcup A$ is the supremum and $\bigcap A$ is the infimum of A. The bottom element is $\perp = \emptyset$ and the top element is $\top = \mathbb{N}$.

Definition 2.24 (Fixed Points) Let $(S, \leq)$ be a lattice and $f : S \to S$ be a map on S. An element $x \in S$ is called

- a *fixed point* of $f \quad :\Leftrightarrow \quad f(x) = x$
- a *pre-fixed point* of $f \quad :\Leftrightarrow \quad f(x) \leq x$
- a *post-fixed point* of $f \quad :\Leftrightarrow \quad f(x) \geq x$ □

Definition 2.25 (Monotonicity) Let $(S, \leq)$ be a lattice and $f : S \to S$ be a map on S. The map f is called *monotone* if $\forall x, y \in S : x \leq y \Rightarrow f(x) \leq f(y)$. □

Theorem 2.26 (Knaster-Tarski) *[Tar55] Let $(S, \leq)$ be a complete lattice and $f : S \to S$ a monotone map on S. The least fixed point of f, denoted μf, exists uniquely and is the infimum of all pre-fixed points. Dually, the greatest fixed point νf exists uniquely and is the supremum of all post-fixed points.*

$$\mu f := \sqcap\{x \in S \mid f(x) \leq x\}$$
$$\nu f := \sqcup\{x \in S \mid x \leq f(x)\}$$

A short proof of that theorem can be found in [Win93].

Lemma 2.27 *Let $w \in \Sigma^\omega$ be a word and φ a μTL formula. Then $f_\varphi(A) := [\![\varphi]\!]^w_{\rho[X \mapsto A]} : 2^{\mathbb{N}} \to 2^{\mathbb{N}}$ is a monotone function in $(2^{\mathbb{N}}, \subseteq)$, where $X \in \mathcal{V}$.*

Proof We will prove this lemma by induction on the structure of φ. Let $A, B \in 2^{\mathbb{N}}$ be two sets with $A \subseteq B$.

If $\varphi = a$ for some $a \in \Sigma$ then

$$[\![a]\!]^w_{\rho[X \mapsto A]} = \{i \in \mathbb{N} \mid w(i) = a\} = [\![a]\!]^w_{\rho[X \mapsto B]}.$$

If $\varphi = X$ for variable $X \in \mathcal{V}$ then

$$[\![X]\!]^w_{\rho[X \mapsto A]} = A \subseteq B = [\![X]\!]^w_{\rho[X \mapsto B]}.$$

If $\varphi = Y \neq X$ for some $Y \in \mathcal{V}$ then

$$[\![Y]\!]^w_{\rho[X \mapsto A]} = \rho(Y) = [\![Y]\!]^w_{\rho[X \mapsto B]}.$$

If $\varphi = \sigma X.\psi$ then

$$[\![\sigma X.\psi]\!]^w_{\rho[X \mapsto A]} = [\![\sigma X.\psi]\!]^w_{\rho} = [\![\sigma X.\psi]\!]^w_{\rho[X \mapsto B]}.$$

If $\varphi = \psi_1 \vee \psi_2$ then

$$\begin{aligned} f_{\psi_1 \vee \psi_2}(A) &= f_{\psi_1}(A) \cup f_{\psi_2}(A) \\ &\overset{IH}{\subseteq} f_{\psi_1}(B) \cup f_{\psi_2}(B) \\ &= f_{\psi_1 \vee \psi_2}(B) \end{aligned}$$

because of the monotonicity of $\cup$. Dually, we can prove that $f_{\psi_1 \wedge \psi_2}(A) \subseteq f_{\psi_1 \wedge \psi_2}(B)$ due to monotonicity of $\cap$.

If $\varphi = O\psi$ then

$$\begin{aligned} f_{O\psi}(A) &= [\![O\psi]\!]^w_{\rho[X \mapsto A]} \\ &= \{i \in \mathbb{N} \mid i+1 \in [\![\psi]\!]^w_{\rho[X \mapsto A]}\} \\ &\overset{IH}{\subseteq} \{i \in \mathbb{N} \mid i+1 \in [\![\psi]\!]^w_{\rho[X \mapsto B]}\} \\ &= [\![O\psi]\!]^w_{\rho[X \mapsto B]} \\ &= f_{O\psi}(B) \end{aligned}$$

If $\varphi = \nu Y.\psi$ then we have to show that $f_{\nu Y.\psi}(A) \subseteq f_{\nu Y.\psi}(B)$.

$$\begin{aligned} x \in f_{\nu Y.\psi}(A) &\Leftrightarrow x \in \bigcup\{S \subseteq 2^{\mathbb{N}} \mid S \subseteq [\![\psi]\!]^w_{\rho[X \mapsto A, Y \mapsto S]}\} \\ &\Leftrightarrow \exists S \subseteq [\![\psi]\!]^w_{\rho[X \mapsto A, Y \mapsto S]} : x \in S \\ &\overset{IH}{\Rightarrow} \exists S \subseteq [\![\psi]\!]^w_{\rho[X \mapsto B, Y \mapsto S]} : x \in S \\ &\Leftrightarrow x \in \bigcup\{S \subseteq 2^{\mathbb{N}} \mid S \subseteq [\![\psi]\!]^w_{\rho[X \mapsto B, Y \mapsto S]}\} \\ &\Leftrightarrow x \in f_{\nu Y.\psi}(B) \end{aligned}$$

If $\varphi = \mu Y.\psi$ then we have to show that $f_{\mu Y.\psi}(A) \subseteq f_{\mu Y.\psi}(B)$.

$$\begin{aligned} x \in f_{\mu Y.\psi}(A) &\Leftrightarrow x \in \bigcap \{S \subseteq 2^{\mathbb{N}} \mid [\![\psi]\!]^w_{\rho[X\mapsto A, Y\mapsto S]} \subseteq S\} \\ &\Leftrightarrow \forall S \in 2^{\mathbb{N}} : \text{if } [\![\psi]\!]^w_{\rho[X\mapsto A, Y\mapsto S]} \subseteq S \text{ then } x \in S \\ &\stackrel{(*)}{\Rightarrow} \forall S \in 2^{\mathbb{N}} : \text{if } [\![\psi]\!]^w_{\rho[X\mapsto B, Y\mapsto S]} \subseteq S \text{ then } x \in S \\ &\Leftrightarrow x \in \bigcap \{S \subseteq 2^{\mathbb{N}} \mid [\![\psi]\!]^w_{\rho[X\mapsto B, Y\mapsto S]} \subseteq S\} \\ &\Leftrightarrow x \in f_{\mu Y.\psi}(B) \end{aligned}$$

$(*)$ holds because of the following fact: if $[\![\psi]\!]^w_{\rho[X\mapsto B, Y\mapsto S]} \subseteq S$ then by IH and transitivity of the $\subseteq$-relation $[\![\psi]\!]^w_{\rho[X\mapsto A, Y\mapsto S]} \subseteq S$ and therefore $x \in S$. ∎

This lemma together with Theorem 2.26 explain our definition for the least and greatest fixed point in Definition 2.13, namely $[\![\mu X.\psi]\!]^w_\rho$ and $[\![\nu X.\psi]\!]^w_\rho$.

Lemma 2.28 *For any μTL formula φ and any ρ:*

$$[\![\varphi[\mu X.\varphi/X]]\!]^w_\rho = [\![\varphi]\!]^w_{\rho[X\mapsto [\![\mu X.\varphi]\!]^w_\rho]} = [\![\mu X.\varphi]\!]^w_\rho$$

Proof Directly from Lemma 2.27 and the the fixed point theorem of Knaster-Tarski (Theorem 2.26). ∎

2.4 Approximants

Definition 2.29 In *μTL* we have two kinds of approximants for the least and the greatest fixed points. They can be defined in the following way:

$$\mu^0 X.\varphi := \mathtt{ff} \qquad \mu^{k+1} X.\varphi := \varphi[\mu^k X.\varphi/X]$$

$$\nu^0 X.\varphi := \mathtt{tt} \qquad \nu^{k+1} X.\varphi := \varphi[\nu^k X.\varphi/X]$$

where $k \in \mathbb{N}$.

Definition 2.30 A set S together with a binary relation $\leq$ having the following properties is called *directed*.

- $\forall x \in S : x \leq x$ (reflexivity)
- $\forall x, y, z \in S : (x \leq y) \wedge (y \leq z) \Rightarrow x \leq z$ (transitivity)
- $\forall x, y \in S : \exists z \in S : (x \leq z) \wedge (y \leq z)$ (directedness) □

Example 2.31

- $(\mathbb{N}, \leq)$ is directed (and so is any totally ordered set).
- Any lattice is directed because it has a supremum for any two elements.

Lemma 2.32 *$(\{[\![\mu^i X.\varphi]\!]^w_\rho \mid i \in \mathbb{N}\}, \subseteq)$ is directed for any w, ρ.*

PROOF Relation $\subseteq$ ensures reflexivity and transitivity. We show by induction that $[\![\mu^k X.\varphi]\!]^w_\rho \subseteq [\![\mu^{k+1} X.\varphi]\!]^w_\rho$ for all $k \in \mathbb{N}$. Due to transitivity of the relation operator any two elements $[\![\mu^i X.\varphi]\!]^w_\rho$, $[\![\mu^j X.\varphi]\!]^w_\rho$, where $i, j \in \mathbb{N}$, are contained in or equal to the element $[\![\mu^{\max(i,j)} X.\varphi]\!]^w_\rho$.

For $k = 0$:

$$[\![\mu^0 X.\varphi]\!]^w_\rho = \emptyset \subseteq [\![\mu^1 X.\varphi]\!]^w_\rho$$

For $k+1 \to k+2$:

$$\begin{aligned}
[\![\mu^{k+1} X.\varphi]\!]^w_\rho &= [\![\varphi[\mu^k X.\varphi/X]]\!]^w_\rho \\
&= [\![\varphi]\!]^w_{\rho[X \mapsto [\![\mu^k X.\varphi]\!]^w_\rho]} \\
&\overset{IH}{\subseteq} [\![\varphi]\!]^w_{\rho[X \mapsto [\![\mu^{k+1} X.\varphi]\!]^w_\rho]} \\
&= [\![\varphi[\mu^{k+1} X.\varphi/X]]\!]^w_\rho \\
&= [\![\mu^{k+2} X.\varphi]\!]^w_\rho
\end{aligned}$$

since the map $\lambda S.[\![\varphi]\!]^w_{\rho[X \mapsto S]}$ is monotone by Lemma 2.27. ∎

Lemma 2.33 *If $(S, \leq)$ is directed and $f : S \to S$ is a monotone map then $S' := (\{f(s) \mid s \in S\}, \leq)$ is directed.*

PROOF Let $f(x)$ and $f(y)$ be two elements of S', where $x, y \in S$. Since S is directed there is a z s.t. $x \leq z$ and $y \leq z$. Therefore there is a $f(z) \in S'$ s.t. $f(x) \leq f(z)$ and $f(y) \leq f(z)$ because f is monotone. ∎

Lemma 2.34 *For every word $w \in \Sigma^\omega$ and every environment $\rho : \mathcal{V} \to 2^N$:*

a) $w \models_\rho \mu X.\psi \quad \Leftrightarrow \quad \exists k \in \mathbb{N} : w \models_\rho \mu^k X.\psi$

b) $w \not\models_\rho \nu X.\psi \quad \Leftrightarrow \quad \exists k \in \mathbb{N} : w \not\models_\rho \nu^k X.\psi$

PROOF a) First we will prove the "$\Leftarrow$" direction.

$$\begin{aligned}
&\forall w \in \Sigma^\omega : \text{if } \exists k \in \mathbb{N} : w \models_\rho \mu^k X.\varphi \text{ then } w \models_\rho \mu X.\varphi \\
\Leftrightarrow\ &\forall w \in \Sigma^\omega : \text{if } w \not\models_\rho \mu X.\varphi \text{ then } \forall k \in \mathbb{N} : w \not\models_\rho \mu^k X.\varphi \\
\Leftrightarrow\ &\forall w \in \Sigma^\omega : \forall k \in \mathbb{N} : (\text{if } w \not\models_\rho \mu X.\varphi \text{ then } w \not\models_\rho \mu^k X.\varphi) \\
\Leftrightarrow\ &\forall w \in \Sigma^\omega : \forall k \in \mathbb{N} : (\text{if } w \models_\rho \mu^k X.\varphi \text{ then } w \models_\rho \mu X.\varphi) \\
\Leftrightarrow\ &\forall w \in \Sigma^\omega : \forall k \in \mathbb{N} : [\![\mu^k X.\varphi]\!]^w_\rho \subseteq [\![\mu X.\varphi]\!]^w_\rho
\end{aligned}$$

Let $w \in \Sigma^\omega$ be an arbitrary word. We prove the last line by induction on k:
If $k = 0$ then

$$[\![\mu^k X.\varphi]\!]^w_\rho = [\![\mathtt{ff}]\!]^w_\rho = \emptyset \subseteq [\![\mu X.\varphi]\!]^w_\rho$$

The induction step $k \rightarrow k+1$:

$$\begin{aligned}
[\![\mu^{k+1} X.\varphi]\!]^w_\rho &= [\![\varphi[\mu^k X.\varphi/X]]\!]^w_\rho \\
&= [\![\varphi]\!]^w_{\rho[X \mapsto [\![\mu^k X.\varphi]\!]^w_\rho]} \\
&\subseteq [\![\varphi]\!]^w_{\rho[X \mapsto [\![\mu X.\varphi]\!]^w_\rho]} \\
&=^1 [\![\mu X.\varphi]\!]^w_\rho
\end{aligned}$$

The inclusion holds because of $[\![\mu^k X.\varphi]\!]^w_\rho \subseteq [\![\mu X.\varphi]\!]^w_\rho$ by IH and monotonicity of $[\![\varphi]\!]$ according to Lemma 2.27.

The "$\Rightarrow$" direction can be shown by fixed point induction in the following way.

$$\begin{aligned}
& \forall w \in \Sigma^\omega : \text{if } w \models_\rho \mu X.\varphi \text{ then } \exists k \in \mathbb{N} : w \models_\rho \mu^k X.\varphi \\
\Leftrightarrow\ & \forall w \in \Sigma^\omega : \text{if } w \models_\rho \mu X.\varphi \text{ then } w \models_\rho \bigvee_{k \in \mathbb{N}} \mu^k X.\varphi \\
\Leftarrow\ & \forall w \in \Sigma^\omega : [\![\mu X.\varphi]\!]^w_\rho \subseteq [\![\bigvee_{k \in \mathbb{N}} \mu^k X.\varphi]\!]^w_\rho \\
\Leftrightarrow\ & \forall w \in \Sigma^\omega : [\![\mu X.\varphi]\!]^w_\rho \subseteq \bigcup_{k \in \mathbb{N}} [\![\mu^k X.\varphi]\!]^w_\rho
\end{aligned}$$

To prove the last line we have to show that $\bigcup_{k \in \mathbb{N}} [\![\mu^k X.\varphi]\!]^w_\rho$ is a prefixed point of $\lambda S.[\![\varphi]\!]^w_{\rho[X \mapsto S]}$. In particular

$$[\![\varphi]\!]^w_{\rho[X \mapsto \bigcup_{k \in \mathbb{N}} [\![\mu^k X.\varphi]\!]^w_\rho]} \subseteq \bigcup_{k \in \mathbb{N}} [\![\mu^k X.\varphi]\!]^w_\rho.$$

Since $[\![\mu X.\varphi]\!]^w_\rho$ is the least of all prefixed points of $\lambda S.[\![\varphi]\!]^w_{\rho[X \mapsto S]}$ the inclusion of the last line holds. Let w be an arbitrary ω-word in Σ^ω. Then

$$\begin{aligned}
[\![\varphi]\!]^w_{\rho[X \mapsto \bigcup_{k \in \mathbb{N}} [\![\mu^k X.\varphi]\!]^w_\rho]} &\overset{(*)}{\subseteq} \bigcup_{k \in \mathbb{N}} [\![\varphi]\!]^w_{\rho[X \mapsto [\![\mu^k X.\varphi]\!]^w_\rho]} \\
&= \bigcup_{k \in \mathbb{N}, k > 0} [\![\varphi]\!]^w_{\rho[X \mapsto [\![\mu^{k-1} X.\varphi]\!]^w_\rho]} \quad \cup \quad [\![\mu^0 X.\varphi]\!]^w_\rho \\
&= \bigcup_{k \in \mathbb{N}, k > 0} [\![\mu^k X.\varphi]\!]^w_\rho \quad \cup \quad [\![\mu^0 X.\varphi]\!]^w_\rho \\
&= \bigcup_{k \in \mathbb{N}} [\![\mu^k X.\varphi]\!]^w_\rho
\end{aligned}$$

[1] by Lemma 2.28

The inclusion $(*)$ is proved by induction on the structure of the formula φ. First define a shorthand for the argument: $M_k := [\![\mu^k X.\varphi]\!]^w_\rho$.

If $\varphi = a$ for some letter a then

$$[\![a]\!]^w_{\rho[X\mapsto\bigcup_{k\in\mathbb{N}} M_k]} = [\![a]\!]^w_\rho = \bigcup_{k\in\mathbb{N}} [\![a]\!]^w_\rho = \bigcup_{k\in\mathbb{N}} [\![a]\!]^w_{\rho[X\mapsto M_k]}$$

If $\varphi = Y$ for some variable $Y \neq X$ then

$$[\![Y]\!]^w_{\rho[X\mapsto\bigcup_{k\in\mathbb{N}} M_k]} = [\![Y]\!]^w_\rho = \bigcup_{k\in\mathbb{N}} [\![Y]\!]^w_\rho = \bigcup_{k\in\mathbb{N}} [\![Y]\!]^w_{\rho[X\mapsto M_k]}$$

If $\varphi = \sigma X.\psi$ then

$$[\![\sigma X.\psi]\!]^w_{\rho[X\mapsto\bigcup_{k\in\mathbb{N}} M_k]} = [\![\sigma X.\psi]\!]^w_\rho = \bigcup_{k\in\mathbb{N}} [\![\sigma X.\psi]\!]^w_\rho = \bigcup_{k\in\mathbb{N}} [\![\sigma X.\psi]\!]^w_{\rho[X\mapsto M_k]}$$

If $\varphi = X$ then by definition

$$[\![X]\!]^w_{\rho[X\mapsto\bigcup_{k\in\mathbb{N}} M_k]} \subseteq \bigcup_{k\in\mathbb{N}} M_k$$

If $\varphi = \varphi_1 \vee \varphi_2$ then

$$\begin{aligned}
[\![\varphi_1 \vee \varphi_2]\!]^w_{\rho[X\mapsto\bigcup_{k\in\mathbb{N}} M_k]} &= [\![\varphi_1]\!]^w_{\rho[X\mapsto\bigcup_{k\in\mathbb{N}} M_k]} \cup [\![\varphi_2]\!]^w_{\rho[X\mapsto\bigcup_{k\in\mathbb{N}} M_k]} \\
&\overset{I.H.}{\subseteq} \bigcup_{k\in\mathbb{N}} [\![\varphi_1]\!]^w_{\rho[X\mapsto M_k]} \cup \bigcup_{k\in\mathbb{N}} [\![\varphi_2]\!]^w_{\rho[X\mapsto M_k]} \\
&= \bigcup_{k\in\mathbb{N}} [\![\varphi_1 \vee \varphi_2]\!]^w_{\rho[X\mapsto M_k]}
\end{aligned}$$

If $\varphi = \varphi_1 \wedge \varphi_2$ then

$$\begin{aligned}
[\![\varphi_1 \wedge \varphi_2]\!]^w_{\rho[X\mapsto\bigcup_{k\in\mathbb{N}} M_k]} &= [\![\varphi_1]\!]^w_{\rho[X\mapsto\bigcup_{k\in\mathbb{N}} M_k]} \cap [\![\varphi_2]\!]^w_{\rho[X\mapsto\bigcup_{k\in\mathbb{N}} M_k]} \\
&\overset{IH}{\subseteq} \bigcup_{k\in\mathbb{N}} [\![\varphi_1]\!]^w_{\rho[X\mapsto M_k]} \cap \bigcup_{k\in\mathbb{N}} [\![\varphi_2]\!]^w_{\rho[X\mapsto M_k]} \\
&\subseteq \bigcup_{k\in\mathbb{N}} [\![\varphi_1 \wedge \varphi_2]\!]^w_{\rho[X\mapsto M_k]}
\end{aligned}$$

The last line holds since

$$
\begin{aligned}
& x \in (\bigcup_{k\in\mathbb{N}} [\![\varphi_1]\!]^w_{\rho[X\mapsto M_k]} \cap \bigcup_{k\in\mathbb{N}} [\![\varphi_2]\!]^w_{\rho[X\mapsto M_k]}) \\
\Leftrightarrow\ & x \in \bigcup_{k\in\mathbb{N}} [\![\varphi_1]\!]^w_{\rho[X\mapsto M_k]} \text{ and } x \in \bigcup_{k\in\mathbb{N}} [\![\varphi_2]\!]^w_{\rho[X\mapsto M_k]} \\
\Leftrightarrow\ & \exists i \in \mathbb{N} : x \in [\![\varphi_1]\!]^w_{\rho[X\mapsto M_i]} \text{ and } \exists j \in \mathbb{N} : x \in [\![\varphi_2]\!]^w_{\rho[X\mapsto M_j]} \\
\Rightarrow^2\ & \exists k \in \mathbb{N} : x \in [\![\varphi_1]\!]^w_{\rho[X\mapsto M_k]} \text{ and } x \in [\![\varphi_2]\!]^w_{\rho[X\mapsto M_k]} \\
\Leftrightarrow\ & \exists k \in \mathbb{N} : x \in [\![\varphi_1 \wedge \varphi_2]\!]^w_{\rho[X\mapsto M_k]} \\
\Leftrightarrow\ & x \in \bigcup_{k\in\mathbb{N}} [\![\varphi_1 \wedge \varphi_2]\!]^w_{\rho[X\mapsto M_k]}
\end{aligned}
$$

If $\varphi = \mu Y.\psi$ then it is to show that $[\![\mu Y.\psi]\!]^w_{\rho[X\mapsto\bigcup_{i\in\mathbb{N}} M_i]} \subseteq \bigcup_{i\in\mathbb{N}}[\![\mu Y.\psi]\!]^w_{\rho[X\mapsto M_i]}$. If we check that the right hand side is a prefixed point of $\lambda S.[\![\psi]\!]^w_{\rho[X\mapsto\bigcup_{i\in\mathbb{N}} M_i, Y\mapsto S]} : 2^{\mathbb{N}} \to 2^{\mathbb{N}}$ then this inclusion holds because $[\![\mu Y.\psi]\!]^w_{\rho[X\mapsto\bigcup_{i\in\mathbb{N}} M_i]}$ is the infimum of all prefixed points, i.e. it is contained or equal to any prefixed point.

$$
\begin{aligned}
& [\![\psi]\!]^w_{\rho[X\mapsto\bigcup_{i\in\mathbb{N}} M_i, Y\mapsto\bigcup_{j\in\mathbb{N}}[\![\mu Y.\psi]\!]^w_{\rho[X\mapsto M_j]}]} \\
\overset{IH}{\subseteq}\ & \bigcup_{i\in\mathbb{N}} [\![\psi]\!]^w_{\rho[X\mapsto M_i, Y\mapsto\bigcup_{j\in\mathbb{N}}[\![\mu Y.\psi]\!]^w_{\rho[X\mapsto M_j]}]} \\
\overset{IH}{\subseteq}\ & \bigcup_{i\in\mathbb{N}} \bigcup_{j\in\mathbb{N}} [\![\psi]\!]^w_{\rho[X\mapsto M_i, Y\mapsto[\![\mu Y.\psi]\!]^w_{\rho[X\mapsto M_j]}]} \\
=^3\ & \bigcup_{k\in\mathbb{N}} [\![\psi]\!]^w_{\rho[X\mapsto M_k, Y\mapsto[\![\mu Y.\psi]\!]^w_{\rho[X\mapsto M_k]}]} \\
=^4\ & \bigcup_{k\in\mathbb{N}} [\![\mu Y.\psi]\!]^w_{\rho[X\mapsto M_k]}
\end{aligned}
$$

and the second inclusion holds because $\lambda S.[\![\mu Y.\psi]\!]^w_{\rho[X\mapsto S]}$ is monotone. Therefore the set $\{[\![\mu Y.\psi]\!]^w_{\rho[X\mapsto M_j]} | j \in \mathbb{N}\}$ is directed (Lemma 2.33) and we can apply the induction hypothesis again.

[2]because of Lemma 2.32 and Lemma 2.33

[3]because of directedness and monotonicity

[4]because $[\![\mu Y.\psi]\!]^w_{\rho[X\mapsto M_j]}$ is a fixed point of $\lambda S.[\![\mu Y.\psi]\!]^w_{\rho[X\mapsto M_k, Y\mapsto S]}$

If $\psi = \nu Y.\psi$ then

$$\begin{aligned}
[\![\nu Y.\psi]\!]^w_{\rho[X\mapsto\bigcup_{i\in\mathbb{N}} M_k]} &= \bigcup\{T \mid T \subseteq [\![\psi]\!]_{\rho[X\mapsto\bigcup_{k\in\mathbb{N}} M_k, Y\mapsto T]}\} \\
&\overset{IH}{\subseteq} \bigcup\{T \mid T \subseteq \bigcup_{k\in\mathbb{N}} [\![\psi]\!]_{\rho[X\mapsto M_k, Y\mapsto T]}\} \\
&\overset{(**)}{\subseteq} \bigcup \bigcup_{k\in\mathbb{N}} \{T \mid T \subseteq [\![\psi]\!]_{\rho[X\mapsto M_k, Y\mapsto T]}\} \\
&= \bigcup_{k\in\mathbb{N}} \bigcup \{T \mid T \subseteq [\![\psi]\!]_{\rho[X\mapsto M_k, Y\mapsto T]}\} \\
&= \bigcup_{k\in\mathbb{N}} [\![\nu Y.\psi]\!]_{\rho[X\mapsto M_k]}
\end{aligned}$$

The inclusion (**) holds because

$$\begin{aligned}
&T \in \{T' \mid T' \subseteq \bigcup_{k\in\mathbb{N}} [\![\psi]\!]_{\rho[X\mapsto M_k, Y\mapsto T']}\} \\
\Rightarrow\ & T' \subseteq \bigcup_{k\in\mathbb{N}} [\![\psi]\!]_{\rho[X\mapsto M_k, Y\mapsto T']}
\end{aligned}$$

and since $(M_k)_{k\in\mathbb{N}}$ is a monotonically increasing sequence (Lemma 2.32) the sequence $([\![\psi]\!]_{\rho[X\mapsto M_k, Y\mapsto T]})_{k\in\mathbb{N}}$ is monotonically increasing as well (Lemma 2.33). Therefore there is a least element s.t.

$$\begin{aligned}
\Rightarrow\ & \exists k \in \mathbb{N} : T' \subseteq [\![\psi]\!]_{\rho[X\mapsto M_k, Y\mapsto T']} \\
\Rightarrow\ & T \in \bigcup_{k\in\mathbb{N}} \{T' \mid T' \subseteq [\![\psi]\!]_{\rho[X\mapsto M_k, Y\mapsto T']}\}
\end{aligned}$$

b) directly from *a*). But first, we check by induction that $[\![\neg\mu^k X.\neg\varphi[\neg X/X]]\!]^w_\rho = [\![\nu^k X.\varphi]\!]^w_\rho$ for all $k \in \mathbb{N}$ and arbitrary w, ρ.

If $k = 0$ then

$$[\![\neg\mu^0 X.\neg\varphi[\neg X/X]]\!]^w_\rho = [\![\neg \mathtt{ff}]\!] = [\![\mathtt{tt}]\!] = [\![\nu^0 X.\varphi]\!]^w_\rho$$

For $k \to k+1$:

$$\begin{aligned}
[\![\neg(\mu^{k+1} X.\neg\varphi[\neg X/X])]\!]^w_\rho &= [\![\neg(\neg\varphi[\neg X/X, (\mu^k X.\neg\varphi[\neg X/X])/X])]\!]^w_\rho \\
&= [\![\varphi[\neg(\mu^k X.\neg\varphi[\neg X/X])/X]]\!]^w_\rho \\
&\overset{IH}{=} [\![\varphi[\nu^k X.\varphi/X]]\!]^w_\rho \\
&= [\![\nu^{k+1} X.\varphi]\!]^w_\rho
\end{aligned}$$

By now the proof of $b)$ is straight forward.

$$\begin{aligned} w \not\models_\rho \nu X.\varphi &\Leftrightarrow w \models_\rho \neg\nu X.\varphi \\ &\Leftrightarrow w \models_\rho \mu X.\neg\varphi[\neg X/X] \\ &\Leftrightarrow \exists k \in \mathbb{N} : w \models_\rho \mu^k X.\neg\varphi[\neg X/X] \\ &\Leftrightarrow \exists k \in \mathbb{N} : w \not\models_\rho \neg\mu^k X.\neg\varphi[\neg X/X] \\ &\Leftrightarrow \exists k \in \mathbb{N} : w \not\models_\rho \nu^k X.\varphi \end{aligned}$$

■

Signatures

Usually, a signature (S, f) is a set with a map $f : S \to \mathbb{N}$ which assigns a number to each element of S. In this work we will use signatures to interpret open formulas. Compared with environments ρ, signatures additionally provide a partial order relation. This extra property will be needed later in proofs.

Definition 2.35 (Signature) Let φ be a well-named μTL formula with exactly m μ-variables. A *μ-signature for φ* is a tuple $\kappa = (k_1, k_2, \ldots, k_m) \in \mathbb{N}^m$. We write $\kappa \leq \kappa'$ if κ is less than or equals κ' in lexicographic ordering. Note that this ordering is total and well-founded.

A μ-signature can be seen as a finite word in $\mathbb{N}^m$. So we will use the same abbreviations as in definition 2.1, i.e. we write $\kappa(X_i)$ or $\kappa(i)$ for the i-th projection to k_i. Besides, both $\kappa^{X_i]}$ and $\kappa^{i]}$ denote the truncation of κ to $(k_1, k_2, \ldots, k_i)$.

All these notations are defined for ν-signatures in the same way. □

Definition 2.36 Let φ be a μTL-formula and $\kappa = (k_1, k_2, \ldots, k_m)$ be a μ-signature for φ. Furthermore, let $Z_1, Z_2, \ldots, Z_n$ denote all variables in φ in increasing order, i.e. $\forall i, j \in \{1, 2, \ldots, n\} : Z_i <_\varphi Z_j \Rightarrow i < j$ (less variables get higher indices). Let $\psi \in Sub(\varphi)$ be a sub-formula of φ. Then we define

$$\psi \circ \kappa \quad :\Leftrightarrow \quad ((\psi \circ s_\kappa(Z_1)) \circ \ldots) \circ s_\kappa(Z_n)$$

where $\chi \circ s_\kappa(Z)$ substitutes variable Z in formula χ by its approximant $\sigma^k Z.\mathit{fb}(Z)$ if some $\kappa(Z)$ exists. Otherwise Z is substituted by its fixed point $\sigma Z.\mathit{fb}(Z)$.

Again, all these notations are defined for ν-signatures in the same way. □

Example 2.37

- Let $\varphi := \mu Z_3.\nu Z_2.(Z_3 \vee Z_2 \vee \mu Z_1.(Z_3 \wedge Z_1))$ be a μTL formula with sub-formula

$\psi = Z_3$ and μ-signature $\kappa = (30, 10)$. Then $\psi \circ \kappa$ is the following formula

$$\begin{aligned}
\psi \circ \kappa &= Z_3 \circ s_1(Z_1) \circ s_2(Z_2) \circ s_3(Z_3) \\
&= Z_3[\mu^{30} Z_1.(Z_3 \wedge Z_1)/Z_1] \circ s_2(Z_2) \circ s_3(Z_3) \\
&= \mu^{30} Z_1.(Z_3 \wedge Z_1) \circ s_2(Z_2) \circ s_3(Z_3) \\
&= \mu^{30} Z_1.(Z_3 \wedge Z_1)[\nu Z_2.(Z_3 \vee Z_2 \vee \mu Z_1.(Z_3 \wedge Z_1))/Z_2] \circ s_3(Z_3) \\
&= \mu^{30} Z_1.(Z_3 \wedge Z_1) \circ s_3(Z_3) \\
&= \mu^{30} Z_1.(Z_3 \wedge Z_1)[\mu^{10} Z_3.\nu Z_2.(Z_3 \vee Z_2 \vee \mu Z_1.(Z_3 \wedge Z_1))/Z_3] \\
&= \mu^{30} Z_1.(\mu^{10} Z_3.\nu Z_2.(Z_3 \vee Z_2 \vee \mu Z_1.(Z_3 \wedge Z_1)) \wedge Z_1)
\end{aligned}$$

Lemma 2.38 *Let φ be a closed μTL formula and κ a σ-signature for φ. Then for every $\psi \in Sub(\varphi)$ the formula $\psi \circ \kappa$ is closed.*

Proof Define $\psi_0 := \psi$ and $\psi_{i+1} := \psi_i \circ s(Z_{i+1})$, where $Z_1, Z_2, \ldots, Z_n$ are the variables and s is the substitution given in Definition 2.36.

$$\psi \circ \kappa = \underbrace{\psi \circ s(Z_1) \circ \ldots \circ s(Z_i)}_{=:\, \psi_i} \circ \ldots \circ s(Z_n)$$

We show by induction on i that ψ_i does not contain any free variable $Z_1, Z_2, \ldots, Z_i$ for all $i = 1, 2, \ldots, n$.

For $i = 1$ variable Z_1 is not free in $\psi_1 = \psi \circ s(Z_1)$ since Z_1 gets bound by the substitution $s(Z_1)$.

For $i \to i + 1$ the formula ψ_{i+1} is of the form $\psi_i \circ s(Z_{i+1})$. By IH there is no free variable Z_j in φ_i, where $j < i$. Besides, these variables do not occur free in $fp\,(Z_{i+1})$. Otherwise let Z_j be free in $fp\,(Z_{i+1})$ for some $j < i + 1$, i.e. $Z_{i+1} <_\varphi Z_j$. But then $i + 1 < j$ which contradicts the naming of the variables. Altogether, there is no free variable $Z_1, Z_2, \ldots, Z_i$ in ψ_{i+1}. Since Z_{i+1} gets bound by $s(Z_{i+1})$ there is no free Z_{i+1} in ψ_{i+1}, as well.

The formula $\psi \circ \kappa$ equals ψ_n and hence, $\psi \circ \kappa$ contains no free variable, i.e. it is closed. ∎

2.5 Logical Games

All games in this thesis are played by two players $\exists$ and $\forall$, called *Eliza* and *Albert*. We will define games with respect to a μTL formula φ and show that φ has certain properties depending on the winner of the game.

Every play starts in an initial configuration and proceeds according to the game rules. During that play both players compete each other and try to win by using their strategies. But usually only one player has a winning strategy and therefore the opponent is doomed to lose in advance.

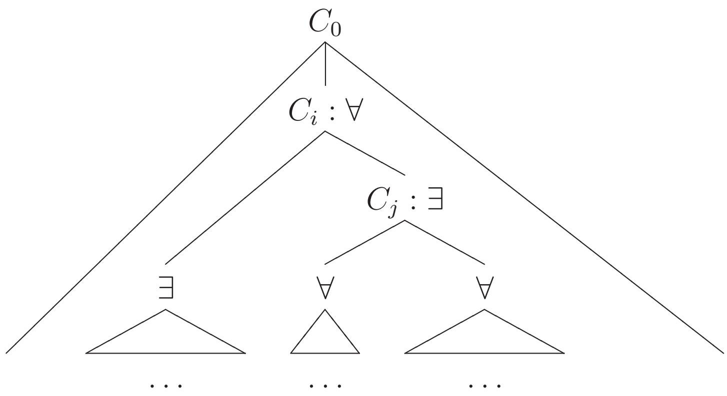


Figure 2.1: A game with all possible plays

Definition 2.39 (Game) A *game* $\mathcal{G}$ is a quadruple $(\mathcal{C}, C_0, \mathcal{R}, \mathcal{W})$ where

- $\mathcal{C}$ is the set of *configurations*,
- $C_0 \in \mathcal{C}$ is the initial configuration,
- $\mathcal{R}$ is a finite set of *rewriting rules* which stipulates the transition between configurations,
- $\mathcal{W}$ is a finite set of *winning conditions*.

A *play* P is a finite or infinite sequence of configurations $P = C_0C_1C_2...$ where for all $i \in \mathbb{N} : (C_i, C_{i+1})$ is an instance of some rule $R \in \mathcal{R}$. Game rules are usually written as

$$r \; \frac{D}{D_1, \ldots, D_n} \quad p \quad i \in \{1, \ldots, n\}$$

meaning: if the actual configuration C_i in a play is of the form D then player p performs a choice $i \in \{1, \ldots, n\}$ and then the next configuration C_{i+1} must be of the form D_i. Define $\mathcal{P}$ as the set of all possible plays in the game.

Winning conditions assign a winner to each play in a game. Notice that a play might be infinite. □

Definition 2.40 (Strategy) A *strategy* for player p in a game $\mathcal{G} = (\mathcal{C}, C_0, \mathcal{R}, \mathcal{W})$ is a partial map $\varsigma : \mathcal{P}_p \to \mathbb{N}$ which determines player p's possible choices, where $\mathcal{P}_p := \{P = C_0C_1 \ldots C_n \in \mathcal{P} \mid p \text{ may perform a choice at } C_n\}$.

A *winning strategy* for player p is a strategy $\varsigma : \mathcal{P}_p \to \mathbb{N}$ s.t. a play is enforced which player p wins, regardless of its opponent's choices. We say, player p *wins* the game iff it has a winning strategy.

A strategy $\varsigma : \mathcal{C} \to \mathbb{N}$ which maps just one configuration $\mathcal{C}$, instead of a whole play, to a configuration is called *positional*. □

3 Model Checking for Words

As described in the introduction the *model-checking* problem for words is to decide whether a certain property, specified as a μTL formula, is preserved in a linear infinite word-model. That is, let $w \in \Sigma^\omega$ be an infinite word and $\varphi \in \mu TL$ be a closed formula. Is w a model of φ, i.e. does $w \models \varphi$ hold?

In this chapter we define a game $MC(w, \varphi)$ which has been introduced by Stirling, see for example [Sti95], and show that player *Eliza* has a winning strategy if and only if w is a model of the formula φ.

3.1 Definition of the Word-Game

Definition 3.1 (*MC* Game) Let $w \in \Sigma^\omega$ and $\varphi \in \mu TL$ be a closed formula. The *model-checking game* $MC(w, \varphi) := (\mathcal{C}, C_0, \mathcal{R}, \mathcal{W})$ is a quadruple where

- configurations $\mathcal{C} = \{w^{[i} \mid i \in \mathbb{N}\} \times Sub(\varphi)$, written as $w^{[i} \vdash \psi$,
- initial configuration $C_0 = w \vdash \varphi$,
- rules $\mathcal{R}$

$$(\vee)\ \frac{w^{[i} \vdash \psi_1 \vee \psi_2}{w^{[i} \vdash \psi_c} \quad \exists c \in \{1,2\} \qquad\qquad (\wedge)\ \frac{w^{[i} \vdash \psi_1 \wedge \psi_2}{w^{[i} \vdash \psi_c} \quad \forall c \in \{1,2\}$$

$$(\mu)\ \frac{w^{[i} \vdash \mu X.\psi}{w^{[i} \vdash X} \qquad\qquad (\nu)\ \frac{w^{[i} \vdash \nu Y.\psi}{w^{[i} \vdash Y}$$

$$(X)\ \frac{w^{[i} \vdash X}{w^{[i} \vdash fb_\varphi(X)} \qquad\qquad (Y)\ \frac{w^{[i} \vdash Y}{w^{[i} \vdash fb_\varphi(Y)}$$

$$(O)\ \frac{w^{[i} \vdash O\psi}{w^{[i+1} \vdash \psi}$$

The rules are read as described in Definition 2.39. For example, if a configuration is of the form $w^{[i} \vdash \psi_1 \vee \psi_2$ then player *Eliza* may choose one disjunct ψ_1 or ψ_2. Then the next configuration is of the form $w^{[i} \vdash \psi_c$.

- and winning conditions $\mathcal{W}$

  An infinite play of *MC* is called *μ-line* if the greatest variable with respect to $<_\varphi$ which occurs infinitely often is of type μ. Otherwise, we call it *ν-line*.

  Player $\exists$ wins a play if

  a) the play ends with $C_n = w^{[i} \vdash a$, where $a \in \Sigma$ and $w(i) = a$,
  b) the play is a ν-line.

  Player $\forall$ wins a play if

  c) the play ends with $C_n = w^{[i} \vdash a$, where $a \in \Sigma$ and $w(i) \neq a$,
  d) the play is a μ-line. □

3.2 Correctness of the Word-Game

Lemma 3.2 *Every play in the game $MC(w, \varphi)$ has a unique winner.*

Proof For every configuration of the form $w \vdash \psi$ where $\psi \notin \Sigma$ there is a rule s.t. the play can continue. Hence, a play ends in a configuration $w \vdash a$ where $a \in \Sigma$ or the play is infinite.

If the play is finite it ends in $w \vdash a$ for some letter a. Then the winning conditions *a*) and *c*) uniquely determine the winner.

As shown in [Sti97] an infinite play is either a μ-line or a ν-line: All rules except for (X) and (Y) reduce the size of the formula of a configuration. Therefore at least one variable Z must occur infinitely often in an infinite play. Let $\{Z_1, Z_2, \ldots, Z_n\}$ be the set of variables which occur infinitely often. For each $i, j \in \{1, \ldots, n\}$ either $Z_i <_\varphi Z_j$ or $Z_j <_\varphi Z_i$ holds (but not both of them). Therefore there is a greatest variable according to transitivity of $<_\varphi$. Then the winning conditions *b*) and *d*) uniquely determine the winner. ■

Lemma 3.3 *Every game $MC(w, \varphi)$ has a unique winner.*

Proof By Martin's theorem [Mar75] every two player game with perfect information and winning conditions which are contained in the Borel hierarchy has a determined winner. Note that the winning conditions are parity conditions and these are in the closure of the second level of the Borel hierachy [Tho03]. ■

Theorem 3.4 (Completeness) *If $w \models \varphi$ then player Eliza wins $MC(w, \varphi)$.*

Proof This theorem can be proved by contradiction. Assume player *Albert* wins the game, i.e. he has a winning strategy for $MC(w, \varphi)$. We will define a winning strategy for player *Eliza* s.t. for the unique resulting play $P = C_0C_1C_2\ldots$, where $C_i =: v_i \vdash \varphi_i$, the following holds

α) for every C_i in P there is a μ-signature κ_i s.t. $v_i \models \varphi_i \circ \kappa_i$

and if P is infinite then

β) there is a C_m in P s.t. for all C_i in P where $m < i$ there is a $i' \in \mathbb{N} : \kappa_{i+i'} < \kappa_i$

Therefore, if player *Albert* wins by winning condition c) then the play ends in $C_n = w^{[i} \vdash a$ where $a \in \Sigma$ and $w(i) \neq a$. But there is no μ-signature κ_n s.t. $w^{[i} \models a \circ \kappa_n$. This is a contradiction to α).

If player *Albert* wins by winning condition d) then the play is an infinite μ-line. Therefore there is a μ variable X which occurs infinitely often. Since β) holds there is $n \in \mathbb{N}$ s.t. $C_n = v_n \vdash X$, $\kappa_n(X) = 0$ and $v_n \models X \circ \kappa_n$ holds. But v_n is never a model of $\mathtt{ff} = X \circ \kappa_n$.

Now we define the strategy for player *Eliza* and prove the properties mentioned above.

Let $C_i = v_i \vdash \psi_1 \vee \psi_2$ be a configuration where player *Eliza* has to select one disjunct. Besides, let κ be the least μ-signature s.t. $v \models (\psi_1 \vee \psi_2) \circ \kappa$. Then we define *Eliza*'s strategy as $\varsigma(C_i) := \psi_c$ s.t. $v \models \psi_c \circ \kappa$ still holds. Notice that this positional strategy only exists if there is such a κ.

We proceed to prove the properties. It is clear that α) holds for C_0 for any κ by precondition since φ_0 is closed. Furthermore, all game rules preserve α): Let $C_i = v_i \vdash \varphi_i$ and $v_i \models \varphi_i \circ \kappa_i$.

If rule $(\wedge)$ is applied on C_i then $\varphi_i =: \psi_1 \wedge \psi_2$.

$$\begin{aligned} v_i \models \varphi_i \circ \kappa_i &\Leftrightarrow v_i \models (\psi_1 \wedge \psi_2) \circ \kappa_i \\ &\Leftrightarrow v_i \models \psi_1 \circ \kappa_i \text{ and } v_i \models \psi_2 \circ \kappa_i \end{aligned}$$

Define $\kappa_{i+1} := \kappa_i$ and then α) holds for C_{i+1} regardless of player *Albert*'s choice.

If rule $(\vee)$ is applied on C_i then $\varphi_i =: \psi_1 \vee \psi_2$.

$$\begin{aligned} v_i \models \varphi_i \circ \kappa_i &\Leftrightarrow v_i \models (\psi_1 \vee \psi_2) \circ \kappa_i \\ &\Rightarrow v_i \models (\psi_1 \vee \psi_2) \circ \kappa \\ &\Leftrightarrow v_i \models \psi_1 \circ \kappa \text{ or } v_i \models \psi_2 \circ \kappa \end{aligned}$$

where κ is the least μ-signature s.t. $v_i \models (\psi_1 \vee \psi_2) \circ \kappa$ holds. Define $\kappa_{i+1} := \kappa$ and apply *Eliza*'s strategy. Then α) holds for C_{i+1}, as well.

If rule (O) is applied on C_i then $\varphi_i =: O\psi$.

$$\begin{aligned} v_i \models \varphi_i \circ \kappa_i &\Leftrightarrow v_i \models O\psi \circ \kappa_i \\ &\Leftrightarrow v_i^{[1} \models \psi \circ \kappa_i \end{aligned}$$

So, the first property holds for C_{i+1}.

If rule (μ) is applied on C_i then $\varphi_i =: \mu X.\psi$.

$$\begin{aligned} v_i \models \varphi_i \circ \kappa_i &\Leftrightarrow v_i \models \mu X.\psi \circ \kappa_i \\ &\Leftrightarrow^1 \exists k \in \mathbb{N} : v_i \models \mu^k X.\psi \circ \kappa_i \\ &\Leftrightarrow \exists k \in \mathbb{N} : v_i \models X \circ \kappa_i[X \mapsto k] \end{aligned}$$

Define $\kappa_{i+1} := \kappa_i[X \mapsto k]$. Then α) holds for C_{i+1}, too.

If rule (ν) is applied on C_i then $\varphi_i =: \nu Y.\psi$.

$$\begin{aligned} v_i \models \varphi_i \circ \kappa_i &\Leftrightarrow v_i \models \nu Y.\psi \circ \kappa_i \\ &\Leftrightarrow v_i \models Y \circ \kappa_i \end{aligned}$$

where $\kappa_{i+1} := \kappa_i$. Note that $\nu Y.\psi \circ \kappa_i = Y \circ \kappa_i$ by definition of the substitution according to κ_i. Again, α) holds for C_{i+1}.

If rule (X) is applied on C_i then $\varphi_i =: X$.

$$\begin{aligned} v_i \models \varphi_i \circ \kappa_i &\Leftrightarrow v_i \models X \circ \kappa_i \\ &\Leftrightarrow^2 v_i \models \mathit{fb}_\varphi(X) \circ \kappa_i[X \mapsto \kappa_i(X) - 1] \end{aligned}$$

Define $\kappa_{i+1} := \kappa_i[X \mapsto \kappa_i(X) - 1]$ and then α) holds for C_{i+1}.

If rule (Y) is applied on C_i then $\varphi_i =: Y$.

$$\begin{aligned} v_i \models \varphi_i \circ \kappa_i &\Leftrightarrow v_i \models Y \circ \kappa_i \\ &\Leftrightarrow v_i \models \mathit{fb}_\varphi(Y)[\nu Y.\mathit{fb}_\varphi(Y)/Y] \circ \kappa_i \\ &\Leftrightarrow v_i \models \mathit{fb}_\varphi(Y) \circ \kappa_i \end{aligned}$$

Define $\kappa_{i+1} := \kappa_i$ and then α) holds for C_{i+1}.

It remains to check property β). Let P be an infinite play. Notice that only after rule (μ) is applied on some configuration C_i the μ-signature κ_{i+1} might be greater than κ_i. All other rules produce a κ_{i+1} which is less or equal to κ_i. We will show that this may only happen finitely many times.

$$\begin{tikzcd} P = & \dots & C_m \arrow[r, bend left, "(\mu)"] & C_{m+1} \arrow[r, bend left, "(X)"] & C_{m+2} & \dots & C_i \arrow[r, bend left, "(X)"] & C_{i+1} & \dots & C_j \arrow[r, bend left, "(X)"] & C_{j+1} & \dots & C_k \arrow[r, bend left, "(X)"] & C_{k+1} & \dots \\ & & & \kappa_{m+1} & < \kappa_{m+2} & & & < \kappa_{i+1} & & & < \kappa_{j+1} & & & < \kappa_{k+1} & \end{tikzcd}$$

Let X be the greatest variable in P which occurs infinitely often. Notice that there must be a configuration C_m where formula $\mu X.\mathit{fb}_\varphi(X)$ occurs the last time

[1]because of Lemma 2.34
[2]by Definition 2.29

since $\mu X.\mathit{fb}_\varphi(X) \notin \mathit{Sub}(\mathit{fb}_\varphi(Z))$ for any variable $Z \leq X$ and X is the greatest of all variables which occur infinitely often. That is, rule (μ) is applied on a configuration with formula X the last time. Hence, β) holds because rule (X) is applied infinitely often and therefore $\kappa_i(X)$ is only counted down. Since X is the greatest of all variables which occur infinitely often even κ_i decreases. ∎

Soundness of the game, i.e. "if $w \not\models \varphi$ then player *Albert* wins $\mathit{MC}(w,\varphi)$", can be proved in a similar way using ν-signatures. Player *Albert*'s strategy is to choose the conjunct s.t. the $\not\models$ relationship of the following configuration holds. For infinite plays, the ν-signature κ can be counted down until a configuration $C_n = v_n \vdash \mathtt{tt}$ is reached.

Another approach is to use negation of a *μTL* formula and show that player *Albert* wins $\mathit{MC}(w,\varphi)$ if player *Eliza* wins $\mathit{MC}(w,\overline{\varphi})$.

Theorem 3.5 (Soundness) *If $w \not\models \varphi$ then player Albert wins $\mathit{MC}(w,\varphi)$.*

Proof This theorem can be transformed to

$$\begin{aligned} w \not\models \varphi \Leftrightarrow w &\models \overline{\varphi} \\ &\Leftrightarrow \textit{Eliza} \text{ wins } \mathit{MC}(w,\overline{\varphi}) \\ &\Rightarrow \textit{Albert} \text{ wins } \mathit{MC}(w,\varphi) \end{aligned}$$

and so the only thing to show is the last implication.

If *Eliza* wins the game $\mathit{MC}(w,\overline{\varphi})$ she has a positional winning strategy ς which determines her moves in the game. We will define a strategy $\overline{\varsigma}$ for player *Albert* and show that he wins every play in $\mathit{MC}(w,\varphi)$ using that strategy.

Define *Albert*'s strategy in $\mathit{MC}(w,\varphi)$ as

$$\overline{\varsigma}(v \vdash \varphi_1 \wedge \varphi_2) := \varsigma(v \vdash \overline{\varphi}_1 \vee \overline{\varphi}_2)$$

Now assume player *Eliza* wins $\mathit{MC}(w,\varphi)$ despite player *Albert*'s strategy $\overline{\varsigma}$ just defined. That is, player *Eliza* wins the resulting play $P = C_0C_1\dots$.

Next, we show that $\overline{P} := \overline{C}_0\overline{C}_1\dots$ is a prefix of a play in $\mathit{MC}(w,\overline{\varphi})$ where player *Eliza* uses her strategy ς.

$$\textit{Eliza} \text{ wins } P = \quad C_0 \searrow r_0 \nearrow C_1 \searrow r_1 \nearrow C_2 \searrow r_2 \nearrow C_3 \searrow r_3 \nearrow C_4 \searrow r_4 \nearrow C_5 \searrow r_5 \nearrow \dots$$

$$\textit{Albert} \text{ wins } \overline{P} = \quad \overline{C}_0 \searrow \overline{r}_0 \nearrow \overline{C}_1 \searrow \overline{r}_1 \nearrow \overline{C}_2 \searrow \overline{r}_2 \nearrow \overline{C}_3 \searrow \overline{r}_3 \nearrow \overline{C}_4 \searrow \overline{r}_4 \nearrow \overline{C}_5 \searrow \overline{r}_5 \nearrow \dots$$

The play starts with $\overline{C}_0 = \overline{\varphi}$ which is a valid initial configuration in $\mathit{MC}(w,\overline{\varphi})$.

If rule $(\vee)$ is played between C_i and C_{i+1} and player *Eliza* chooses disjunct φ_c then rule $(\wedge)$ can be played between $\overline{C}_i$ and $\overline{C}_{i+1}$ where player *Albert* choses conjunct $\overline{\varphi}_c$. That is, player *Albert* chooses conjuncts according to player *Eliza*'s strategy in $MC(w, \varphi)$.

If rule $(\wedge)$ is played between C_i and C_{i+1} and player *Albert* chooses conjunct φ_c then

$$\overline{\varsigma}(C_i) =: \overline{\varsigma}(v \vdash \varphi_1 \wedge \varphi_2) = \varsigma(v \vdash \overline{\varphi}_1 \vee \overline{\varphi}_2)$$

Therefore rule $(\vee)$ can be played between $\overline{C}_i$ and $\overline{C}_{i+1}$ where player *Eliza* chooses $\overline{\varphi}_c$ according to her strategy!

If rule (μ) is played between C_i and C_{i+1} then rule (ν) can be played between $\overline{C}_i$ and $\overline{C}_{i+1}$. The case for (ν) is dual.

If rule (X), rule (Y) or rule (O) is played between C_i and C_{i+1} then the same rule can be played between $\overline{C}_i$ and $\overline{C}_{i+1}$. Notice that each μ-variable in $MC(w, \varphi)$ becomes a ν-variable in $MC(w, \overline{\varphi})$ and ν-variables become μ-variables.

If the play P is finite it ends with an configuration $C_n = v \vdash a$ where $v(0) = a$. Therefore $\overline{P}$ ends with $\overline{C}_n = v \vdash \overline{a}$ and player *Eliza* is doomed to loose at most $|\Sigma| - 1$ steps later with configuration $\overline{C}_{n+|\Sigma|-1} = v \vdash b$ where $b \in \Sigma$ and $b \neq a = v(0)$, regardless which disjuncts she chooses.

If P is infinite then it is a ν-line. But then $\overline{P}$ must be a μ-line since the types of variables are changed and hence, player *Albert* wins P.

We have seen that player *Albert* wins $\overline{P}$. On the other hand, during the play P player *Eliza* uses her winning strategy which exists by the precondition. So we get a contradiction and therefore player *Eliza* cannot enforce a play in $MC(w, \varphi)$ which she wins. ∎

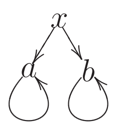
4 Model Checking for Trees

In this chapter we will extend the model checking problem for words using trees as interpretations. The question to be solved is now: let $t \in \mathcal{T}_\Sigma$ be a tree over Σ and φ a μTL-formula, is t a model of φ, i.e. does $t \models \varphi$ hold?

Again, we will define a model checking game $MC(t, \varphi)$, where player *Eliza* wins if and only if $t \models \varphi$. Basically, player *Albert*'s goal is to show that $t \not\models \varphi$, i.e. that for some path $w \in t : w \not\models \varphi$. Therefore, if t is not a model of φ then player *Albert* can choose that path $w \in t$ non-deterministically before the play begins. After that he certainly wins $MC(w, \varphi)$ which provides a counter-example.

In practice, it is impossible to check for all paths $w \in t$ whether player *Albert* wins $MC(w, \varphi)$ or not. Hence, we permit *Albert* to gradually determine its chosen path at every application of rule (O). But then there are some cases where player *Albert* can beat his counterpart *Eliza* effectively even though the trees are models of the formulas:

Let t be a tree with only two paths $w_1 = xa^\omega$ and $w_2 = xb^\omega$ where x is the label of the root. Then t surely is a model of the μTL-formula $\varphi := Oa \vee Ob$ since both $w_1 \models \varphi$ and $w_2 \models \varphi$ hold:



$\models \quad Oa \vee Ob$

But player *Eliza* would be defeated in the following way:

$$\dfrac{x \vdash Oa \vee Ob}{\dfrac{x \vdash Ob}{a \vdash b}\ \forall : a \qquad\qquad \dfrac{b \vdash Oa}{b \vdash a}\ \forall : b}\ \exists$$

As we have seen, player *Albert* becomes too strong because he does not need to reveal his counter-example $w \in t$ before the play begins. Therefore we also strengthen player *Eliza* by allowing her to choose both disjunctions at rule $(\vee)$ simultaneously. Thus, configurations of the resulting game definition become sets of formulas.

4.1 Definition of the Tree-Game

Definition 4.1 (MC game) Let $t \in \mathcal{T}_\Sigma$ be a tree over Σ and let φ be a closed μTL-formula. The model checking game $MC(t, \varphi) := (\mathcal{C}, C_0, \mathcal{R}, \mathcal{W})$ consists of

- configurations $\mathcal{C} = \{w \in \Sigma^* \mid w \in t\} \times 2^{Sub(\varphi)}$, written as $w \vdash \Phi$ where Φ denotes a set of formulas from $Sub(\varphi)$,
- start configuration $C_0 := \varepsilon \vdash \varphi_0$,
- rules $\mathcal{R}$:

$$(\vee)\ \frac{w \vdash \psi_1 \vee \psi_2, \Phi}{w \vdash \psi_1, \psi_2, \Phi} \qquad\qquad (\wedge)\ \frac{w \vdash \psi_1 \wedge \psi_2, \Phi}{w \vdash \psi_c, \Phi}\quad \forall\ c$$

$$(\mu)\ \frac{w \vdash \mu X.\psi, \Phi}{w \vdash X, \Phi} \qquad\qquad (\nu)\ \frac{w \vdash \nu Y.\psi, \Phi}{w \vdash Y, \Phi}$$

$$(X)\ \frac{w \vdash X, \Phi}{w \vdash fb\,(X), \Phi} \qquad\qquad (Y)\ \frac{w \vdash Y, \Phi}{w \vdash fb\,(Y), \Phi}$$

$$(O)\ \frac{w \vdash O\psi_1, \ldots, O\psi_m, a_1, \ldots, a_n}{wa \vdash \psi_1, \ldots, \psi_m}\quad \forall\ a \in \Sigma \text{ s.t. } wa \in t$$

Let $P = C_0C_1\ldots$ be a play. A *principal formula* (PF) of a configuration $C_i \in P$ is a formula which is rewritten according to the rule which is applied on C_i. Note that all rules except for rule (O) rewrites just one formula of a configuration.

In the play P we can connect each formula of a configuration C_{i+1} to the formula of the previous configuration C_i where it comes from. The *connection relation* $\mathcal{C}on_P = \mathbb{N} \times Sub(\varphi_0) \times Sub(\varphi_0)$ is defined as

a) $(i, \psi, \psi) \in \mathcal{C}on_P \quad :\Leftrightarrow \quad \psi \in C_i$ and ψ is no PF and $\psi \in C_{i+1}$.
b) $(i, \psi, \psi') \in \mathcal{C}on_P \quad :\Leftrightarrow \quad \psi \in C_i$ and ψ is a PF and ψ rewrites to ψ'.

A *line* in a play is a finite or infinite sequence of formulas $L = \varphi_0\varphi_1\varphi_2\ldots$ s.t.

$$\varphi_0 \in C_0 \quad \text{and} \quad \text{for all } i = 0, 1, 2, \ldots : (i, \varphi_i, \varphi_{i+1}) \in \mathcal{C}on_P$$

An infinite line is called *μ-line* if the greatest variable which occurs infinitely often is of type μ. Otherwise, we call it *ν-line*.

- winning conditions (WC) $\mathcal{W}$:
Player $\exists$ wins a play, if

a) the play reaches a configuration $C_n = w \vdash a, \Phi$, where $w(|w| - 1) = a \in \Sigma$,
b) the play is infinite and there is a ν-line in the play.

Player $\forall$ wins a play, if

c) the play reaches a configuration $C_n = w \vdash a_1, \dots, a_m$, where $w(|w|-1) \neq a_i \in \Sigma$ for all $i = 1, 2, \dots, m$,

d) the play is infinite and there is no ν-line in the play. □

4.2 Correctness of the Tree-Game

Lemma 4.2 *Every play in the game $MC(t, \varphi)$ has a unique winner.*

PROOF Every play $P = C_0 C_1 \dots$ which does not end in a configuration of the form $C_A := w \vdash a, \Phi$, where $w(|w|-1) = a$, or $C_B := w \vdash a_1, \dots, a_m$, where $w(|w|-1) \neq a_i$ for all a_i, can continue. Assume the P get stuck in configuration $C_n := w_n \vdash \Phi_n$ which is not of the form of C_A and C_B, i.e. Phi_n does not contain the proposition $w_n(|w_n|-1)$ and Φ_n is not a set of propositions. Thus, there must be at least one formula φ in Φ_n which is not a proposition. But then at least one game rule can be applied.

If P is finite then its last configuration $C_n := w_n \vdash \Phi_n$ is of the form C_A or C_B. According to the winning conditions a) and c) player *Eliza* wins if and only if Φ_n contains the proposition $w(|w|-1)$.

If P is infinite then by winning conditions b) and d) player *Eliza* wins if and only if P has a ν-line. ■

Theorem 4.3 (Soundness) *If $t \not\models \varphi_0$ then player Albert wins $MC(t, \varphi_0)$.*

PROOF This theorem is proved in two steps. First we define a strategy $s_\forall$ for player *Albert* which enforces a play $P = C_0 C_1 \dots$ since player *Eliza* never intervenes in these games. Then, due to Lemma 4.2, it remains to show that player *Eliza* does not win the play P. Define $C_i =: v_i \vdash \Phi_i$ as the path and the formula set of configuration C_i.

In this paragraph we define a strategy for player *Albert*. By the precondition $t \not\models \varphi_0$, there is an path $w \in t$ s.t. $w \not\models \varphi_0$. Thus, at every (O) rule in the play player *Albert* can choose a successor $a \in \Sigma$ of the path according to w.

$$s_\forall(v \vdash O\psi_1, \dots, O\psi_m, a_1, \dots, a_n) := va \vdash \psi_1, \dots, \psi_m$$

s.t. va is a prefix of w.

If rule $(\wedge)$ is applied on a configuration $C_i := v_i \vdash \Phi_i$ with principal formula $\psi_1 \wedge \psi_2 \in \Phi_i$ then *Albert* calculates the least ν-signature κ s.t. $v_i \not\models (\psi_1 \wedge \psi_2) \circ \kappa$. Then either $v_i \not\models \psi_1 \circ \kappa$ or $v_i \not\models \psi_2 \circ \kappa$ holds by definition. Player *Albert*'s strategy for this kind of configuration is defined as

$$s_\forall(v \vdash \psi_1 \wedge \psi_2) := v \vdash \psi_c, \quad \text{where } c \in \{1, 2\}$$

$\Phi_0 =$		$\{\varphi_0\}$			
$\Phi_1 = \{$	φ_1	$\cdots$			$\}$
$\Phi_2 = \{$		φ_2	$\cdots$		$\}$
$\Phi_3 = \{$		$\cdots$		φ_3	$\}$
$\Phi_4 = \{$		$\cdots$	φ_4		$\}$
$\cdots$		$\cdots$			

Table 4.1: One possible line in the enforced play P.

s.t. $v \not\models \psi_c \circ \kappa$. Note that we have to show that such a κ always exists! This is done in the next paragraph.

Let $L := \varphi_0\varphi_1\ldots$ be a line in the play P. We will construct a sequence of ν-signatures $K = \kappa_0\kappa_1\ldots$ s.t. for all formulas φ_i on the line $w^{[|v_i|} \not\models \varphi_i \circ \kappa_i$ holds. See Table 4.1.

By the precondition, $w^{[|v_0|} \not\models \varphi_0 \circ \kappa_0$ holds for any κ_0 since $w^{[|v_0|} = w$ and φ_0 is assumed to be closed. Define $\kappa_0 := (0,\ldots,0)$ for example. Next, we show that the rules and player *Albert*'s strategy preserve that property, i.e. for each position $i \in \mathbb{N}$ on the line where $w^{[|v_i|} \not\models \varphi_i \circ \kappa_i$ we will find a κ_{i+1} s.t. $w^{[|v_{i+1}|} \not\models \varphi_{i+1} \circ \kappa_{i+1}$. Let $w^{[|v_i|} \not\models \varphi_i \circ \kappa_i$.

Case 1: If φ_i is not a principal formula of its configuration C_i then $\varphi_{i+1} = \varphi_i$ and $|v_{i+1}| = |v_i|$ since rule (O) is not applied. Define $\kappa_{i+1} := \kappa_i$ and hence $w^{[|v_{i+1}|} \not\models \varphi_{i+1} \circ \kappa_{i+1}$.

Case 2: If φ_i is a principal formula of its configuration C_i then we have to deal with the following sub-cases:

- If $\varphi_i = \psi_1 \vee \psi_2$ then $\varphi_{i+1} = \psi_c$ for some $c = 1,2$ and $|v_{i+1}| = |v_i|$. By assumption $w^{[|v_i|} \not\models (\psi_1 \vee \psi_2) \circ \kappa_i$, i.e. $w^{[|v_i|} \not\models \psi_1 \circ \kappa_i$ and $w^{[|v_i|} \not\models \psi_2 \circ \kappa_i$. But then define $\kappa_{i+1} := \kappa_i$ and so $w^{[|v_{i+1}|} \not\models \psi_c \circ \kappa_{i+1}$ holds for any $c = 1,2$.

- If $\varphi_i = \psi_1 \wedge \psi_2$ then $\psi_{i+1} = \psi_c$ for some $c = 1,2$ and $|v_{i+1}| = |v_i|$. Player *Albert* calculates the least κ s.t. $w^{[|v_i|} \not\models (\psi_1 \wedge \psi_2) \circ \kappa$. The existence of such a ν-signature $\kappa \leq \kappa_i$ follows directly from assumption. Thus, we can define

$\kappa_{i+1} := \kappa$ and player *Albert* can choose that conjunct φ_c s.t. $w^{[|v_{i+1}|} \not\models \psi_c \circ \kappa_{i+1}$ according to its strategy.

- If $\varphi_i = O\psi$ then $\varphi_{i+1} = \psi$ and $|v_{i+1}| = |v_i| + 1$. Define $\kappa_{i+1} := \kappa_i$ and by assumption $w^{[|v_i|} \not\models O\psi \circ \kappa_{i+1}$ holds and therefore $w^{[|v_{i+1}|} \not\models \psi \circ \kappa_{i+1}$ holds, as well.

- If $\varphi_i = \mu X.\psi$ then $\varphi_{i+1} = X$ and $|v_{i+1}| = |v_i|$. Define $\kappa_{i+1} := \kappa_i$ and so $w^{[|v_{i+1}|} \not\models X \circ \kappa_{i+1}$ since X is substituted by $\mu X.\psi \circ \kappa_{i+1}$.

- If $\varphi_i = X$ then $\varphi_{i+1} = \mathit{fb}_{\varphi_0}(X)$ and $|v_{i+1}| = |v_i|$. Define $\kappa_{i+1} := \kappa_i$ and then $w^{[|v_{i+1}|} \not\models \mathit{fb}_{\varphi_0}(X) \circ \kappa_{i+1}$ since all free variables X are substituted by its fixed points $\mu X.\mathit{fb}_{\varphi_0}(X)$ (see Lemma 2.28).

- If $\varphi_i = \nu Y.\psi$ then $\varphi_{i+1} = Y$ and $|v_{i+1}| = |v_i|$. From assumption we can infere that there exists a least $k \in \mathbb{N}$ s.t. $w^{[|v_{i+1}|} \not\models \nu^k Y.\psi \circ \kappa_i$. Notice that φ_i does not contain the free variable Y. Therefore we can define $\kappa_{i+1} := \kappa_i$ and then update the ν-signature to the appropriate index of the approximant $\kappa_{i+1}(Y) := k$. Then $w^{[|v_{i+1}|} \not\models Y \circ \kappa_{i+1}$ holds, as well.

- If $\varphi_i = Y$ then $\varphi_{i+1} = \mathit{fb}_{\varphi_0}(Y)$ and $|v_{i+1}| = |v_i|$. Define $\kappa_{i+1} := \kappa_i$ and update $\kappa_{i+1}(Y)$ to $\kappa_{i+1}(Y) - 1$. Then $w^{[|v_{i+1}|} \not\models \mathit{fb}_{\varphi_0}(Y) \circ \kappa_{i+1}$ by definition of approximants (Definition 2.29). Notice that the ν-signature strictly decreases at rule (Y), i.e. $\kappa_i > \kappa_{i+1}$!

By now we defined a strategy $s_\forall$ and checked that player *Albert* is able to use it in the play. It remains to enshure that player *Albert* wins the enforced play P.

Case 1: Assume player *Eliza* wins $P = C_0C_1 \dots C_n$ by WC a), i.e. $C_n = v_n \vdash a, \Psi'_n$ and $v_n(|v_n| - 1) = a \in \Sigma$. But then there is a line $L = \varphi_0\varphi_1 \dots \varphi_n$ which ends in formula $\varphi_n = a$. Therefore the suffix of the counter-model $w^{[|v_n|} = a \dots$ begins with letter a and hence $w^{[|v_n|} \models a \circ \kappa'$ for any κ'. But this result contradicts the existence of κ_n.

Case 2: Assume player *Eliza* wins $P = C_0C_1 \dots$ by WC b), i.e. there is a ν-line $L = \varphi_0\varphi_1 \dots$ in that play. Let Y be the greatest variable on that line which occurs infinitely often.

First notice that there must be a position m where formula $\varphi_m = \nu Y.\psi$ occurs the last time in the line since $\nu Y.\psi \notin \mathit{Sub}(\mathit{fb}(Y'))$ for any $Y' \leq Y$. Besides, only variables $Y' <_{\varphi_0} Y$ occur after position m.

Furthermore $\kappa_{i+1}^{Y]} \leq \kappa_i^{Y]}$ for all $i \geq m$ since κ_{i+1} is constructed s.t. $\kappa_{i+1} \leq \kappa_i$ in all cases except for case $\varphi_i = \nu Y'.\psi$. But after position m only variables $Y' < Y$ occur and therefore the prefix of κ_{i+1} is not touched.

Since variable Y occurs infinitely often the sequence $(\kappa_i(Y))_{i=m,m+1,\dots}$ strictly decreases. The ordering of κ is well-founded and hence we will reach a position n where

$\varphi_n = Y$ and $\kappa_n(Y) = 0$. In other words, this leads to the contradiction $w^{[|v_n|} \not\models Y \circ \kappa_n$ where $Y \circ \kappa_n \equiv \mathtt{tt}$. ■

Theorem 4.4 (Completeness) *If $t \models \varphi_0$ then player Eliza wins MC(t, φ_0).*

Proof The idea behind this proof is very similar to the proof of Theorem 4.3.

Assume player *Albert* wins this game, i.e. he has a winning strategy $s_\forall$ and can enforce a play $P := C_0C_1\dots$ which he wins. Let w be the path chosen by him during the play and let $C_i =: v_i \vdash \Phi_i$ denote the path and the formula set of configuration C_i. Since this game is only influenced by one player, namely player *Albert*, there is no need to give a strategy for player *Eliza*.

We can show the following property of the play P: There is a line $L = \varphi_0\varphi_1\dots$ in this play P and a sequence of μ-signatures $K = \kappa_0\kappa_1\dots$ s.t. for all formulas φ_i on the line $w^{[|v_i|} \models \varphi_i \circ \kappa_i$ holds.

We will construct the line L and the sequence K step by step. For the initial configuration $w^{[|v_0|} \models \varphi_0 \circ \kappa_0$ for any κ_0 by the precondition since $w^{[|v_0|} \in t$ and φ_0 is assumed to be closed. Define $\kappa_0 := (0, \dots, 0)$ for example. Assume $w^{[|v_i|} \models \varphi_i \circ \kappa_i$.

Case 1: If φ_i is not a principal formula then $\varphi_{i+1} = \varphi_i$. Define $\kappa_{i+1} := \kappa_i$ and hence $w^{[|v_{i+1}|} \models \varphi_{i+1} \circ \kappa_{i+1}$ is trivially true.

Case 2: If φ_i is a principal formula then there are several sub-cases to deal with:

- If $\varphi_i = \psi_1 \vee \psi_2$ then $|v_{i+1}| = |v_i|$. By assumption $w^{[|v_i|} \models (\psi_1 \vee \psi_2) \circ \kappa_i$ and therefore $w^{[|v_i|} \models \psi_c \circ \kappa_i$ holds for at least one of the disjuncts $\varphi_c \in \{\psi_1, \psi_2\}$. Define $\varphi_{i+1} := \varphi_c$ and $\kappa_{i+1} := \kappa_i$ and the property holds for the next configuration C_{i+1}.

- If $\varphi_i = \psi_1 \wedge \psi_2$ then $|v_{i+1}| = |v_i|$. Since $w^{[|v_i|} \models (\psi_1 \wedge \psi_2) \circ \kappa_i$ holds by assumption both $w^{[|v_i|} \models \psi_1 \circ \kappa_i$ and $w^{[|v_i|} \models \psi_2 \circ \kappa_i$ hold. Define the next position of the line $\varphi_{i+1} := s_\forall(C_i)$ and $\kappa_{i+1} := \kappa_i$. Then $w^{[|v_{i+1}|} \models \varphi_{i+1} \circ \kappa_{i+1}$ holds as well, regardless of which conjunct player *Albert* chooses.

- If $\varphi_i = O\psi$ then $\varphi_{i+1} = \psi$ and $|v_{i+1}| = |v_i| + 1$. Define $\kappa_{i+1} := \kappa_i$. Then $w^{[|v_{i+1}|} \models \varphi_{i+1} \circ \kappa_{i+1}$ can easily be inferred from the assumption and the semantics definition of O.

- If $\varphi_i = \mu X.\psi$ then $\varphi_{i+1} = X$ and $|v_{i+1}| = |v_i|$. By assumption we can infer that $w^{[|v_{i+1}|} \models \mu X.\psi \circ \kappa_{i+1}$ holds. Therefore, there is a $k \in \mathbb{N}$ s.t. $w^{[|v_{i+1}|} \models \mu^k X.\psi \circ \kappa_{i+1}$. Define $\kappa_{i+1} := \kappa_i$ and update the index of variable X in this μ-signature to $\kappa_{i+1}(X) := k$. Then $w^{[|v_{i+1}|} \models X \circ \kappa_{i+1}$ holds, as well.

- If $\varphi_i = X$ then $\varphi_{i+1} = \mathit{fb}_{\varphi_0}(X)$ and $|v_{i+1}| = |v_i|$. Define $\kappa_{i+1} := \kappa_i$ and then decrease the index of the approximant $\kappa_{i+1}(X)$ by one, i.e. $\kappa_{i+1}(X) := \kappa_i(X) - 1$. By definition of the approximants $w^{[|v_{i+1}|} \models \mathit{fb}_{\varphi_0}(X) \circ \kappa_{i+1}$ holds (see Definition 2.29).

- If $\varphi_i = \nu Y.\psi$ then $\varphi_{i+1} = X$ and $|v_{i+1}| = |v_i|$. We define $\kappa_{i+1} := \kappa_i$ and hence $w^{[|v_{i+1}|} \models Y \circ \kappa_{i+1}$ holds because Y is substituted by $\nu Y.\psi \circ \kappa_{i+1}$.
- If $\varphi_i = Y$ then $\varphi_{i+1} = \mathit{fb}_{\varphi_0}(Y)$ and $|v_{i+1}| = |v_i|$. Define $\kappa_{i+1} := \kappa_i$ and then $w^{[|v_{i+1}|} \not\models \mathit{fb}_{\varphi_0}(Y) \circ \kappa_{i+1}$ since all free variables Y are substituted by its fixed points $\nu Y.\mathit{fb}_{\varphi_0}(Y)$ (see Lemma 2.28).

In the first part of the proof we demonstrated that a line $L \in P$ and a sequence of μ-signatures K exist s.t. for every φ_i on the line $w^{[|v_i|} \models \varphi_i \circ \kappa_i$. Now we prove that *Albert* cannot win that play P.

Case 1: Assume player *Albert* wins by WC c), i.e. the play P ends with a configuration $C_n := v_n \vdash a_1, a_2, \ldots, a_m$ where none of the propositions $a_j \in \Sigma$ is equal to the last letter of v_n, or mathematically speaking for all $a_j \in \Sigma : v_n(|v_n| - 1) \neq a_j$. But then each line $L' = \varphi_0\varphi_1 \ldots \varphi_n$ of the play ends in some symbol of C_n. Therefore $w^{[|v_n|} \models \varphi_n \circ \kappa_n$ cannot hold since $w^{[|v_n|}$ begins with a letter which is not in Φ_n. This contradicts the existence of the line L.

Case 2: If the play P is infinite then our constructed line must be infinite as well. Suppose L ends in a symbol $a \in \Sigma$ in configuration C_m. Then $w^{[|v_m|} \models a \circ \kappa_m$ holds and hence the word $w^{[|v_m|}$ must begin with letter a and v_m must end with letter a. But then player *Eliza* would win the play by winning condition a). Now assume that player *Albert* wins by WC d), i.e. the infinite play P has no ν-line. Hence, our infinite line L must be a μ-line. Let X be the greatest variable on L which occurs infinitely often. Notice that there is a position m s.t. $\varphi_m = \mu X.\psi$ occurs the last time on line L. Furthermore the prefix of the μ-signatures $\kappa_i^{X]}$ strictly decreases from time to time for $i \geq m$, because only variables $X' <_{\varphi_0} X$ occurs after position m and variable Y occurs infinitely often. So, we will reach a position $\varphi_n = X$ with $\kappa_n(X) = 0$, i.e. $w^{[|v_n|} \models X \circ \kappa_n$ where X is substituted by $\texttt{ff}$. ∎

5 Validity and Satisfiability Games

5.1 Validity Checking by *MC* Games

The tree games of Chapter 4 already provide a method for checking the validity of a μTL formula φ. The idea is to play the game $MC(t, \varphi)$ on a tree t which consists of all possible words $w \in \Sigma$. In our definition the root of that tree can only be annotated by exactly one letter. Therefore we have to adjust this idea a little bit.

Definition 5.1 A *universal tree* $t_\Sigma : \mathbb{D} \to \Sigma$ with $\Sigma = \{a_0, a_1, \ldots, a_n\}$ is defined in the following way. $\mathbb{D} := \{0, 1, ..., n\}^*$ and t is given by

$$\begin{aligned} t(\varepsilon) &:= a_0 \\ t(wi) &:= a_i \end{aligned}$$

where $w \in \mathbb{D}$ and $i = 0, 1, \ldots, n$. $\square$

A universal tree t_Σ for $\Sigma = \{a_0, \ldots, a_n\}$ is depicted in Table 5.1. It is easy to see that the set of all paths in the tree $t_\Sigma^{[0}$ – the tree which begins at the first child with label a_0 – equals to the set of all infinite words in Σ which begins with letter a_0. Hence, the set of all infinite words over Σ equals to $\bigcup_{i=0,\ldots,n}\{w \in \Sigma^\omega \mid w \in t_\Sigma^{[i}\}$.

Lemma 5.2 *Let φ be a closed μTL formula. Then Player Eliza wins $MC(t_\Sigma, O\varphi)$ iff φ is valid.*

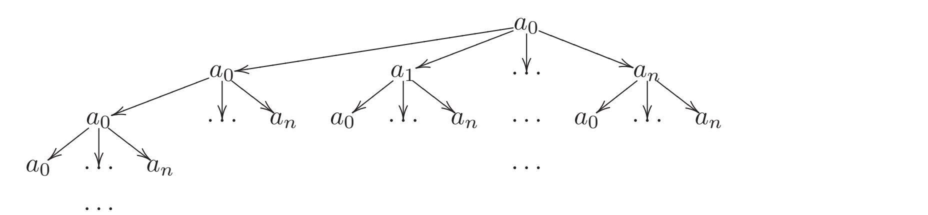


Table 5.1: A universal tree for $\Sigma = \{a_0, \ldots, a_n\}$

Proof

$$\begin{aligned}
\textit{Eliza} \text{ wins } MC(t_\Sigma, O\varphi) &\Leftrightarrow t_\Sigma \models O\varphi \\
&\Leftrightarrow \text{for all } w \in t : w \models O\varphi \\
&\Leftrightarrow \text{for all } w \in t : w^{[1} \models \varphi \\
&\Leftrightarrow \text{for all } i = 0, 1, \ldots, |\Sigma| : \text{for all } w \in t^{[i} : w \models \varphi \\
&\Leftrightarrow \text{for all } w \in \Sigma^\omega : w \models \varphi \\
&\Leftrightarrow \varphi \text{ is valid}
\end{aligned}$$

∎

5.2 Validity Checking Game VAL

Definition 5.3 (VAL Game) Let φ be a closed μTL formula. The *validity checking game* $VAL(\varphi) = (\mathcal{C}, C_0, \mathcal{R}, \mathcal{W})$ consists of

- configurations $\mathcal{C} = 2^{Sub(\varphi)}$
- start configuration $C_0 = \varphi$
- rules $\mathcal{R}$:

$$(\vee)\ \frac{\psi_1 \vee \psi_2, \Phi}{\psi_1, \psi_2, \Phi} \qquad (\wedge)\ \frac{\psi_1 \wedge \psi_2, \Phi}{\psi_j, \Phi} \quad \forall\, j$$

$$(\mu)\ \frac{\mu X.\psi, \Phi}{X, \Phi} \qquad (\nu)\ \frac{\nu Y.\psi, \Phi}{Y, \Phi}$$

$$(X)\ \frac{X, \Phi}{\mathit{fb}_\varphi(X), \Phi} \qquad (Y)\ \frac{Y, \Phi}{\mathit{fb}_\varphi(Y), \Phi}$$

$$(O)\ \frac{O\psi_1, \ldots, O\psi_m, a_1, \ldots, a_n}{\psi_1, \ldots, \psi_m}$$

  For the definition of a principal formula, μ-line and ν-line see Definition 4.1.

- winning conditions (WC) $\mathcal{W}$:
  Player $\exists$ wins a play if

  a) the play reaches a configuration $C_n = (a_1, \ldots, a_m, \Phi)$, where $m = |\Sigma|$,
  b) the play is infinite and there is a ν-line in the play.

  Player $\forall$ wins a play if

c) the play reaches a configuration $C_n = (a_1, \dots, a_m)$, where $m < |\Sigma|$,

d) the play is infinite and there is no ν-line in the play. □

Lemma 5.4 *Let φ be a closed μTL formula and t_Σ a universal tree over Σ. Player Albert wins $MC(t_\Sigma, O\varphi)$ iff he wins $VAL(\varphi)$.*

Proof "⇒" We start with the "only if" direction. Suppose Player *Albert* wins the game $MC(t_\Sigma, O\varphi)$, i.e. he has a winning strategy for this game and can enforce a play

$$P = C_0C_1C_2\dots$$

which he wins. Let $C_i =: v_i \vdash \Phi_i$ for all configurations C_i in the play. Notice that these games are one-player games and therefore only player *Albert* can influence the course of a play. Define

$$P' := \Phi_1\Phi_2\dots$$

as the sequence of formula sets in the play P.

In $MC(t_\Sigma, O\varphi)$ only rule (O) can be applied on the first configuration C_0. So Φ_1 must be formula φ.

Moreover, it is easy to see that for all configurations C_i in the play P the following fact holds: if (C_i, C_{i+1}) is an instance of some rule in $MC(t_\Sigma, O\varphi)$ then there is a rule in $VAL(\varphi)$ where (Φ_i, Φ_{i+1}) is an instance of. This is due to the fact that the rule schemata of the game $VAL(\varphi)$ are almost the same as in the game $MC(t_\Sigma, O\varphi)$. Only the first component of a configuration, namely the position in t_Σ, is disregarded. Hence, P' is a play in $VAL(\varphi)$ which can be enforced by player *Albert*.

It remains to check that player *Albert* wins the play P' in $VAL(\varphi)$.

Case 1: If $P = C_0C_1\dots C_n$ is finite then player *Albert* wins $MC(t_\Sigma, O\varphi)$ by winning condition c), i.e. $C_n = v_n \vdash b_1, b_2, \dots, b_m$ where $m < |\Sigma|$ and the last letter of v_n is not a symbol in C_n. Notice that there cannot be a configuration in P of the form $C_i = v_i \vdash a_1, a_2, \dots, a_m$ where $m = |\Sigma|$ for $i < n$. Otherwise the play would be finished at position i with winner *Eliza* since v_i's last letter must be in Σ. Hence the play P' in $VAL(\varphi)$ does not finish before configuration Φ_m occurs and then player *Albert* wins by winning condition c).

Case 2: If P is infinite then it contains no ν-line. Therefore P' is infinite and contains no ν-line either. Thus, player *Albert* wins P' by winning condition d).

"⇐" The other direction of the proof is similar. If player *Albert* wins the $VAL(\varphi)$ game, he has a winning strategy and can guide the game into a play $P' = \Phi_1\Phi_2\dots$ which he wins.

Since the formula φ is guarded there is only a finite number of configurations between two subsequent configurations Φ_i and Φ_j where rule (O) is applied on. We can define a function $f : \mathbb{N} \to \mathbb{N}$ which maps a configuration in P' – identified by its position – to the position of the next configuration Φ_i where the game has applied rule (O). This is

$$f(i) := min(\{j \in \mathbb{N} \mid j \geq i \text{ and } (O) \text{ is applied on } \Phi_j\} \cup \{n \in \mathbb{N} \mid P' \text{ ends with } \Phi_n\})$$

$$P' = \Phi_1 \ \Phi_2 \ \cdots \ \Phi_i \ \Phi_{i+1} \ \Phi_{i+2} \ \cdots \ \Phi_j \ \Phi_{j+1} \ \cdots$$

Again, let Φ_i and Φ_j be some subsequent configurations in P' where rule (O) is applied on. Let $letters(\Phi)$ denote the set of letters in configuration Φ. According to the rules of $VAL(\varphi)$ no letter in a configuration can be a principal formula, i.e. the number of letters of a configuration increases until rule (O) gets rid of them. Formally, $letters(\Phi_k) \subseteq letters(\Phi_{k+1})$ for all $k = i, i+1, \ldots, j-1$.

Furthermore, the set of letters in any configuration Φ in play P' is not equal to Σ. Otherwise, player *Eliza* would have won that play by winning condition a). In other words, for every configuration C_i there is always a letter $b \in \Sigma$ which is not in the following before-(O)-configuration $f(i)$!

With these preface we are able to define a play $P = C_0C_1\ldots$ of the game $MC(t_\Sigma, O\varphi)$ and show that player *Eliza* will lose this play. Define

$$P := (v_0 \vdash O\varphi)(v_1 \vdash \Phi_1)(v_2 \vdash \Phi_2)\ldots$$

where v_0 is some arbitrary letter in Σ and for all $i = 1, 2, \ldots : v_i$ is a prefix of v_{i+1} for all $i = 1, 2, \ldots$ and every v_i ends with a letter b which is not in $f(i)$.

This sequence is a play in $MC(t_\Sigma, O\varphi)$ since it starts with formula $O\varphi$ and (C_i, C_{i+1}) is an instance of some rule in $MC(t_\Sigma, O\varphi)$ for each $i \in \mathbb{N}$. This holds because the rule schemata which operate on the formula set are identical to the rules in $VAL(\varphi)$ and because the positions, namely the v_is, only change after an application of rule (O).

Now we ensure that player *Albert* wins the play P:

Case 1: Player *Eliza* cannot win a finite play $P = C_0C_1\ldots C_n$ in $MC(t_\Sigma, O\varphi)$ by winning condition a), i.e. $C_n = v_n \vdash a, \Phi'_n$ where the last letter of v_n is a, since for all configurations C_i in P: the last letter of v_i is not in $letters(f(i))$ which contains $letters(C_i)$.

Case 2: She cannot win by winning condition b), i.e. there is a ν-line in P, because then she would win the game $VAL(\varphi)$ by the same winning condition. ∎

Theorem 5.5 *Let φ be a closed μTL formula. φ is valid iff player Eliza wins $VAL(\varphi)$.*

PROOF Directly from Lemma 5.2 and Lemma 5.4:

$$\begin{aligned} \varphi \text{ valid } &\Leftrightarrow \exists \text{ wins } MC(t_\Sigma, O\varphi) \\ &\Leftrightarrow \exists \text{ wins } VAL(\varphi) \end{aligned}$$

■

5.3 Satisfiability Checking *SAT*

Definition 5.6 (SAT game) Let φ be a closed μTL formula. The *satisfiability checking game* $SAT(\varphi) = (\mathcal{C}, C_0, \mathcal{R}, \mathcal{W})$ consists of

- configurations $\mathcal{C} = 2^{Sub(\varphi)}$
- start configuration $C_0 = \varphi$
- rules $\mathcal{R}$:

$$(\vee)\ \frac{\psi_1 \vee \psi_2, \Phi}{\psi_c, \Phi}\ \exists\ c \qquad (\wedge)\ \frac{\psi_1 \wedge \psi_2, \Phi}{\psi_1, \psi_2, \Phi}$$

$$(\mu)\ \frac{\mu X.\psi, \Phi}{X, \Phi} \qquad (\nu)\ \frac{\nu Y.\psi, \Phi}{Y, \Phi}$$

$$(X)\ \frac{X, \Phi}{fb_\varphi(X), \Phi} \qquad (Y)\ \frac{Y, \Phi}{fb_\varphi(Y), \Phi}$$

$$(O)\ \frac{O\psi_1, \ldots, O\psi_m, a_1, \ldots, a_n}{\psi_1, \ldots, \psi_m}$$

  For the definition of a principal formula, μ-line and ν-line see Definition 4.1.

- winning conditions (WC) $\mathcal{W}$:
  Player $\exists$ wins a play if

  a) the play reaches a configuration $C_n = a$, where $a \in \Sigma$,
  b) the play is infinite and there is no μ-line in the play.

  Player $\forall$ wins a play if

  c) the play reaches a configuration $C_n = (a, b, \Phi)$, where $a, b \in \Sigma$ and $a \neq b$,
  d) the play is infinite and there is a μ-line in the play. □

	$SAT(\varphi_0)$	$VAL(\overline{\varphi}_0)$
	$(\wedge)\ \frac{\psi_1 \wedge \psi_2, \Phi}{\psi_1, \psi_2, \Phi}$	$(\vee)\ \frac{\psi_1 \vee \psi_2, \Phi}{\psi_1, \psi_2, \Phi}$
$\exists$ wins:	a) $C_n = a$ b) no μ-line	a) $C_n = a_1, \dots, a_{\lvert\Sigma\rvert}, \Phi$ b) ν-line
$\forall$ wins:	c) $C_n = a, b, \Phi$ d) μ-line	c) $C_n = a_1, \dots, a_m,\ m < \lvert\Sigma\rvert$ d) no ν-line

Table 5.2: Duality of *VAL* and *SAT*

Lemma 5.7 *Let φ be a closed μTL formula. Then φ is satisfiable iff Albert wins $VAL(\overline{\varphi})$.*

Proof

$$\begin{aligned} \varphi \text{ satisfiable} &\Leftrightarrow \text{there is a } w \in \Sigma^\omega : w \models \varphi \\ &\Leftrightarrow \text{there is a } w \in \Sigma^\omega : w \not\models \overline{\varphi} \\ &\Leftrightarrow \overline{\varphi} \text{ is not valid} \\ &\Leftrightarrow \forall \text{ wins } VAL(\overline{\varphi}) \end{aligned}$$ ∎

Lemma 5.8 *Let φ be a closed μTL formula. Player Eliza wins $SAT(\varphi_0)$ iff player Albert wins $VAL(\overline{\varphi}_0)$.*

Proof First notice that both games actually are one-player games, i.e. only one of the players really have the possibility to influence a play. In Table 5.2 the differences of both games are depicted.

"⇒" We start with the "only if" direction. If player *Eliza* wins the game $SAT(\varphi)$ then she has a winning strategy and can enforce a play $P = C_0 C_1 \dots$ which she wins.
Let C be a configuration in P. Define

$$\overline{C} := \{\overline{\varphi[\overline{Z}_1/Z_1, \dots, \overline{Z}_n/Z_n]} \in Sub(\overline{\varphi}_0) \mid \varphi \in C\}$$

containing all formulas of C in negated form, where $Z_1, \dots, Z_n$ are all variables of φ_0. Within two steps we construct a play P' in $VAL(\overline{\varphi}_0)$ which player *Albert* wins. First define $\overline{P} := \overline{C}_0 \overline{C}_1 \dots$.

We show that for any two subsequent configurations C_i, C_{i+1} in the play P the following holds. If (C_i, C_{i+1}) is an instance of some rule $\mathcal{R} \setminus (O)$ in the game $SAT(\varphi_0)$ then there is also a rule r' in the game rules of $VAL(\overline{\varphi}_0)$ s.t. $(\overline{C}_i, \overline{C}_{i+1})$ is an instance of r'.

If (C_i, C_{i+1}) is an instance of rule $(\wedge)$ then $C_i = \varphi_1 \wedge \varphi_2, \Phi$ and $C_{i+1} = \varphi_1, \varphi_2, \Phi$. By definition $\overline{C}_i = \overline{\varphi_1 \wedge \varphi_2}, \overline{\Phi} = \overline{\varphi}_1 \vee \overline{\varphi}_2, \overline{\Phi}$ and so $(\overline{C}_i, \overline{C}_{i+1})$ is an instance of rule $(\vee)$ in $VAL(\overline{\varphi}_0)$.

If (C_i, C_{i+1}) is an instance of rule $(\vee)$ then $C_i = \varphi_1 \vee \varphi_2, \Phi$ and $C_{i+1} = \varphi_c, \Phi$, where player *Eliza* chooses c. By definition $\overline{C}_i = \overline{\varphi_1 \vee \varphi_2}, \overline{\Phi} = \overline{\varphi}_1 \wedge \overline{\varphi}_2, \overline{\Phi}$ and so $(\overline{C}_i, \overline{C}_{i+1})$ is an instance of rule $(\wedge)$ in the $VAL(\overline{\varphi}_0)$ game, where player *Albert* may choose the disjunct $\overline{\varphi}_c$.

If (C_i, C_{i+1}) is an instance of rule (μ) then $C_i = \mu Z.\varphi, \Phi$ and $C_{i+1} = Z, \Phi$. By definition $\overline{C}_i = \overline{\mu X.\varphi}, \overline{\Phi} = \nu Z.\overline{\varphi[\overline{X}/X]}, \overline{\Phi}$ and so $(\overline{C}_i, \overline{C}_{i+1})$ is an instance of rule (ν) in $VAL(\overline{\varphi}_0)$. The case for rule (ν) is similar. Observe that a variable Z in play P is of the other type in sequence $\overline{P}$.

If (C_i, C_{i+1}) is an instance of rule (X) then $C_i = X, \Phi$ and $C_{i+1} = fb_{\varphi_0}(X), \Phi$. By definition $\overline{C}_i = X, \overline{\Phi}$ and $\overline{C}_{i+1} = \overline{fb_{\varphi_0}(\overline{X})}, \overline{\Phi} = fb_{\overline{\varphi}_0}(X), \overline{\Phi}$. So $(\overline{C}_i, \overline{C}_{i+1})$ is an instance of rule (X) in $VAL(\overline{\varphi}_0)$. The case for rule (Y) is similar.

What we have so far is the following sequence

$$
\begin{tikzcd}[column sep=small]
P = C_0 \arrow[r, bend right=60, "r_0"'] & C_1 & \cdots & C_{i-1} \arrow[r, bend right=60, "r_{i-1}"'] & C_i \arrow[r, bend left=60, "(O)"] & C_{i+1} \arrow[r, bend right=60, "r_{i+1}"'] & C_{i+2} & \cdots & C_{j-1} \arrow[r, bend right=60, "r_{j-1}"'] & C_j \arrow[r, bend left=60, "(O)"] & C_{j+1} \arrow[r, bend right=60, "r_{j+1}"'] & C_{j+2} & \cdots
\end{tikzcd}
$$

$$
\begin{tikzcd}[column sep=small]
\overline{P} = \overline{C}_0 \arrow[r, bend right=60, "r'_0"'] & \overline{C}_1 & \cdots & \overline{C}_{i-1} \arrow[r, bend right=60, "r'_{i-1}"'] & \overline{C}_i \arrow[r, bend left=60, "(?)"] & \overline{C}_{i+1} \arrow[r, bend right=60, "r'_{i+1}"'] & \overline{C}_{i+2} & \cdots & \overline{C}_{j-1} \arrow[r, bend right=60, "r'_{j-1}"'] & \overline{C}_j \arrow[r, bend left=60, "(?)"] & \overline{C}_{j+1} \arrow[r, bend right=60, "r'_{j+1}"'] & \overline{C}_{j+2} & \cdots
\end{tikzcd}
$$

Unfortunately $\overline{P}$ is still not a play in $VAL(\varphi_0)$. The transition between $\overline{C}_i$ and $\overline{C}_{i+1}$ might not be defined if (C_i, C_{i+1}) is an instance of rule (O). We need to insert some extra configurations between $\overline{C}_i$ and $\overline{C}_{i+1}$ in the sequence $\overline{P}$.

Let C_i be a configuration in P where rule (O) is applied on.

Case 1: If C_i is of the form $O\varphi_1, \ldots, O\varphi_m$ then $\overline{C}_i$ is of the form $\overline{O\varphi_1}, \ldots, \overline{O\varphi_m}$. That is equal to $O\overline{\varphi}_1, \ldots, O\overline{\varphi}_m$ by definition and hence rule (O) can transform $\overline{C}_i$ to $\overline{C}_{i+1}$, at once.

Case 2: If C_i is of the form $O\varphi_1, \ldots, O\varphi_m, a$, where a is a letter in Σ, then $\overline{O\varphi_1}, \ldots, \overline{O\varphi_m}, \overline{a}$. That is equal to $O\overline{\varphi}_1, \ldots, O\overline{\varphi}_m, \overline{a}$ by definition. Now rule $(\vee)$ is deterministically applied $|\Sigma| - 2$ times on $\overline{a} = \bigvee_{b \in \Sigma, b \neq a} b$ with resulting configuration $C'_{i+|\Sigma|-2} := O\overline{\varphi}_1, \ldots, O\overline{\varphi}_m, b_1, \ldots, b_{|\Sigma|-1}$. Afterwards rule (O) can rewrite this configuration to C_{i+1}.

Case 3: If C_i is of the form $O\varphi_1, \ldots, O\varphi_m, a_1, \ldots, a_m$ for more than one letter $a_1, \ldots, a_m$ then player *Albert* would win by winning condition c). Therefore this case does not occur.

Notice that in all configurations between C_i and C_{i+1} the number of letters are less than $|S|$.

Define P' as the sequence $\overline{P}$ where additional configurations $C'_{i+1}, \ldots, C'_{i+|\Sigma|-2}$ are inserted to flatten the disjunction a s.t. rule (O) can be applied. Then P' is a play in $VAL(\varphi_0)$:

$$\begin{tikzcd}[column sep=small]
P' = & \overline{C}_0 \arrow[r, bend right, "r_0"'] & \overline{C}_1 & \cdots & \overline{C}_{i-1} \arrow[r, bend right, "r_{i-1}"'] & \overline{C}_i \arrow[r, bend left, "(\vee)"] & C'_{i+1} \arrow[r, bend left, "(\vee)"] & C'_{i+2} & \cdots & C'_{i+|\Sigma|-2} \arrow[r, bend left, "(O)"] & \overline{C}_{i+1} \arrow[r, bend right, "r_{i+1}"'] & \overline{C}_{i+2} & \cdots
\end{tikzcd}$$

Finally, it remains to show that player *Albert* wins that play.

Case 1: If $P = C_0 \ldots C_n$ in $SAT(\varphi_0)$ is finite then P' is finite as well. By definition of P' there is a $\overline{C}_n = \overline{a} = \bigvee_{b \in \Sigma, b \neq a} b$ for some $a \in \Sigma$ and therefore this play will end in configuration $C'_{n+|\Sigma|-2} = b_1, \ldots, b_m$ where $m = |\Sigma| - 1$. Hence, player *Albert* wins by WC c).

Case 2: If $P = C_0 C_1 \ldots$ is infinite then it does not contain a μ-line. Otherwise player *Albert* would win $SAT(\varphi_0)$ by WC d). Notice that *Eliza* cannot win the play P' by winning condition a) since each negated configuration $\overline{C}$ does not contain a single letter $a \in \Sigma$. This is because there is no formula φ s.t. $\overline{\varphi} = a$. Secondly, in the additional inserted configurations $C'_{i+1} \ldots C'_{i+|\Sigma|-2}$ the maximal number of letters is restricted by $|\Sigma| - 1$ as mentioned above. Therefore the play P' is infinite as well and contains no ν-line, by definition. Remember that a variable in P is of contrary type in P'. But then player *Albert* wins that play by WC d).

"$\Leftarrow$" If player *Albert* wins $VAL(\overline{\varphi}_0) =: (\mathcal{C}, \overline{\varphi}, \mathcal{R}, \mathcal{W})$, he has a winning strategy and can enforce a play $P := C_0 C_1 \ldots$ which he wins. Notice that the order of the application of rules $\mathcal{R} \setminus (O)$ does not matter. So, we may assume that formulas of the form $\overline{a}$, where $a \in \Sigma$, become principal formulas last. That is, no rule is applied on such a formula until all remaining formulas in the configuration are of the form $O\varphi$, $\overline{a}$ or a.

Let $C_{j_1}, C_{j_2}, \ldots$ denote the configurations in P where rule (O) is applied on. Additionally, let C_{i_k} be the configuration with the following properties

- C_{i_k} precedes C_{j_k}, and
- C_{i_k} is the first configuration after the last application of rule (O) which contains a negated letter $\overline{a}$ as principal formula.

This designation is depicted in the following figure:

$$P = \overbrace{C_0\ C_1 \cdots \underbrace{C_{i_1}\ C_{i_1+1}}_{(\vee)\text{ on some }\overline{a}} \cdots}^{\text{no }(O)} \overbrace{C_{j_1}\ C_{j_1+1}}^{(O)} \overbrace{\cdots \underbrace{C_{i_2}\ C_{i_2+1}}_{(\vee)\text{ on some }\overline{a}} \cdots}^{\text{no }(O)} \overbrace{C_{j_2}\ C_{j_2+1}}^{(O)} \cdots$$

Notice the following facts: (i) the configurations $C_0, \ldots, C_{i_1}$ and all configurations between $C_{j_{k-1}}$ and C_{i_k+1} for $k = 2, 3, \ldots$ do not contain any letter $a \in \Sigma$. A letter a in $\overline{\varphi}$ only occurs as an argument of a disjunction $\overline{b}$ for some negated letter $b \in \Sigma$, where $a \neq b$. Since we assumed that formulas of the form $\overline{b}$ become principal formulas last there cannot be a letter in these configuration.

Secondly, (ii) there is at most one negated letter $\overline{a}$ in any configuration of P. Assume there is a configuration $C = \overline{a}, \overline{b}, \Phi$ in P where $a \neq b$. Then this configuration will eventually be rewritten to $C' = a_1, \ldots, a_{|\Sigma|}, \Phi'$ and there is nothing player *Albert* can do about it. But that means player *Eliza* wins the play by WC a).

$$P = \underbrace{\overbrace{C_0\ C_1 \cdots C_{i_1}}^{(i)\text{ no }a}\ C_{i_1+1} \cdots \overbrace{C_{j_1}\ C_{j_1+1}}^{(O)} \overbrace{\cdots C_{i_2}}^{(i)\text{ no }a}\ C_{i_2+1} \cdots \overbrace{C_{j_2}\ C_{j_2+1}}^{(O)} \cdots}_{(ii)\text{ at most one }\overline{a}}$$

Combining (i) and (ii), for any k the configuration C_{i_k} must look like

$$\begin{aligned} & C_{i_k} = O\varphi_1, \ldots, O\varphi_m, \\ \text{or} \quad & C_{i_k} = O\varphi_1, \ldots, O\varphi_m, \overline{a}, \end{aligned}$$

i.e. the configuration C_{i_k} contains at most one negated letter. Note that on these configurations C_{i_k} for $k = 1, 2, \ldots r_i$ either rule $(\vee)$ or rule (O) can be applied.

Next we delete all configurations in P which are between C_{i_k} and C_{j_k} and show that the resulting sequence Q is a "negation" of a possible play in $SAT(\varphi) =: (\mathcal{C}', \varphi, \mathcal{R}', \mathcal{W}')$:

$$Q = D_0 \xrightarrow{r_0} D_1 \xrightarrow{r_1} D_2 \ldots := C_0 \xrightarrow{r_0} C_1 \xrightarrow{r_1} C_2 \cdots C_{i_1} \xrightarrow{?} C_{j_1} \cdots C_{i_2} \xrightarrow{?} C_{j_2} \cdots$$

where $r_i \in \mathcal{R}$ denotes the rule which is applied on configuration D_i according to the play P.

Define the negation of a configuration $\overline{C}$ like in the first part of the proof, namely

$$\overline{C} := \{\overline{\varphi[\overline{Z}_1/Z_1, \ldots, \overline{Z}_n/Z_n]} \in Sub(\overline{\varphi}_0) \mid \varphi \in C\}.$$

Then the following holds: "Let D_i and D_{i+1} be two subsequent configurations in Q and let C be a configuration s.t. $\overline{C} = D_i$. Then there is a rule $r' \in \mathcal{R}'$ s.t. $\overline{r'(C)} = D_{i+1}$." We write $r(C)$ for the configuration which results from application of rule r on configuration C. Let us prove this fact:

Let ψ be the principal formula in D_i. Since $\overline{C} = D_i$ there must be a formula φ in C s.t. $\overline{\varphi} = \psi$. There are several chases:

If $\varphi = a$ for some letter in Σ then r_i operates on $\psi = \overline{a}$, i.e. by definition of Q $D_i = O\psi_1, \dots, O\psi_m, \overline{a}$ and $D_{i+1} = \psi_1, \dots, \psi_m$. Furthermore configuration C must be of the form $O\varphi_1, \dots, O\varphi_m, a$ where $\overline{\varphi}_j = \psi_j$ for all j. This fact holds due to $\overline{O\varphi_j} = O\overline{\varphi}_j = O\psi_j$ for all j and because there is no other formula χ_j s.t. $\overline{\chi}_j = O\psi_j$ for any j.

If $\varphi = \varphi_1 \wedge \varphi_2$ then r_i operates on $\psi = \overline{\varphi}_1 \vee \overline{\varphi}_2$. Therefore $r' := (\wedge)$ applied on C with principal formula φ yields a set $r'(C)$ s.t. $\overline{r'(C)} = \overline{(C \setminus \varphi) \cup \{\varphi_1, \varphi_2\}} = (D_i \setminus \psi) \cup \{\overline{\varphi}_1, \overline{\varphi}_2\} = D_{i+1}$.

If $\varphi = \varphi_1 \vee \varphi_2$ then player *Albert* chooses a conjunct in $\psi = \overline{\varphi}_1 \wedge \overline{\varphi}_2$ using rule $r = (\wedge)$. Hence rule $r' := (\vee)$ can rewrite φ where player *Eliza* chooses the corresponding disjunct. Again $\overline{r'(C)} = \overline{(C \setminus \varphi) \cup \{\varphi_j\}} = (D_i \setminus \psi) \cup \{\overline{\varphi}_j\} = D_{i+1}$.

If $\varphi = O\varphi'$ then $\psi = O\overline{\varphi}'$ and rule $r_i = (O)$. Moreover, D_i must be of the form $(O\psi_1, \dots, O\psi_m)$ without any letter a because there is no formula χ s.t. $\overline{\chi} = a$ and therefore there would not be a configuration C s.t. $\overline{C} = D_i$ otherwise. As in the first case, $C = (O\varphi_1, \dots, O\varphi_m)$ where $\overline{\varphi}_j = \psi_j$ for all j. If we apply rule $r' = (O)$ on C we get $\overline{r'(C)} = \overline{\{\varphi_1, \dots, \varphi_m\}} = \{\psi_1, \dots, O\psi_m\} = D_{i+1}$.

If $\varphi = \mu Z.\varphi'$ then $\psi = \nu Z.\overline{\psi'[\overline{Z}/Z]}$ and rule $r = (\nu)$. Define $r' := (\mu)$ and apply it on C to obtain $\overline{r'(C)} = \overline{(C \setminus \mu Z.\varphi') \cup \{Z\}} = (D_i \setminus \nu Z.\varphi) \cup \{Z\} = D_{i+1}$. The case $\varphi = \nu Z.\varphi'$ is dual. Notice that a variable Z in the sequence Q is of the other type in the play P'.

If $\varphi = X$ then $\psi = X$, as well (see the definition of $\overline{C}$). We can apply the same rule $r' := r$ and get $r'(C) = D_{i+1}$ since $fb(X)$ according to $Sub(\overline{\varphi}_0)$ is the negation of $fb(X)$ with respect to $Sub(\varphi_0)$.

This leads to a definition of the possible play $P' := C'_0 C'_1 \dots$ in $SAT(\varphi_0)$:

$$
\begin{aligned}
C_0 &:= \overline{D}_0 \\
C_i &:= r'(C_i), \quad \text{s.t. } \overline{r'(C_i)} = D_i
\end{aligned}
$$

Next, we show that player *Eliza* wins the play $P' = C'_0 C'_1 \dots$ in $SAT(\varphi_0)$. Assume her opponent *Albert* wins by winning condition c), i.e. the play ends in a configuration of the form $C_n = a, b, \Phi$ for some distinct letters a and b. But then D_n must contain two negated letters which is impossible by fact (ii) and definition of Q.

If player *Albert* wins by winning condition d), i.e. there is a μ-line in P, then there must be a ν-line in Q because the type of the variables changes. Then P also contains a ν-line since in the parts of the play which are dropped to define Q the rules only operate on negated letters. So, player *Eliza* would win P by winning condition b) which contradicts the precondition. ∎

Theorem 5.9 *Let φ be a closed μTL formula. Then φ is satisfiable iff player Eliza wins $SAT(\varphi)$.*

Proof Directly from Lemma 5.7 and Lemma 5.8:

$$\begin{aligned} \varphi \text{ satisfiable } &\Leftrightarrow \textit{Albert} \text{ wins } VAL(\overline{\varphi}) \\ &\Leftrightarrow \textit{Eliza} \text{ wins } SAT(\varphi) \end{aligned}$$

∎

6 ν-Line Automata

In the preceding chapters we presented tree-games to solve the model checking problem. Furthermore, we showed how to adapt these games for deciding validity and satisfiability of closed μTL formulas. The winning conditions for infinite plays in these games depend on the existence of a μ- or ν-line but we have not given an effective algorithm for detecting such lines, so far.

In this chapter we introduce an automaton based approach to find ν-lines in a play. Automata seem to be appropriate to solve this task because they consist of elementary mathematical structures like sets and relations on sets which on the one hand can easily be examined in terms of mathematical theorems and proofs and which on the other hand are simple enough to be implemented in a common programming language. For deeper insight into automata theory see for example [Tho97, GTW02, HMU02].

Our aim in this chapter is to construct an automaton which reads a play $P = C_0C_1\ldots$ of a game and outputs whether there is a ν-line in that play. If this automaton is deterministic we can annotate each configuration of the play by a state of the automaton. Thus, it will be possible to create a decision procedure for the validity game which is in PSPACE because no extra branching due to the non-determinism of the automaton is needed.

We will define two kinds of automata. Parity automata have a complex acceptance condition and hence, we can directly define these kinds of automata for detecting a ν-line. Next we will transform these automata into Büchi automata where we follow a standard construction from Parity to Büchi. The advantage of Büchi automata are that there exist standard procedures for converting them into deterministic Muller automata. We will use an optimal algorithm which has been introduced by Safra [Saf89].

6.1 Preliminaries

Definition 6.1 (Büchi Automaton) A non-deterministic *Büchi automaton* (NBA) is a quintuple $\mathcal{A}_B = (S, A, \delta, s_0, F)$ where

- S is a finite set of *states*,
- A is a finite *alphabet*,
- $\delta \subseteq S \times A \times S$ is called *transition relation*,

- $s_0 \in S$ is the *initial state*,
- $F \subseteq S$ is the set of *finite states*.

A *run* of $\mathcal{A}_B$ on an infinite word $w \in A^\omega$ is a sequence of states $R = s_0 s_1 \ldots$ s.t. $(s_i, w(i), s_{i+1}) \in \delta$ for all $i \in \mathbb{N}$. A run is *accepting* if it contains a state $s \in F$ which occurs infinitely often. We say that $\mathcal{A}_B$ accepts a word w if there is an accepting run on w. □

Definition 6.2 (Parity Automaton) A nondeterministic *parity automaton* (NPA) is a quintuple $\mathcal{A}_P = (S, A, \delta, s_0, F)$ where

- S, A, δ, s_0 are defined as in Büchi automata,
- $F : S \to \mathbb{N}$ is a *priority function* which assigns a priority to each state.

A *run* of $\mathcal{A}_P$ on an infinite word $w \in A^\omega$ is a sequence of states $R = s_0 s_1 \ldots$ s.t. $(s_i, w(i), s_{i+1}) \in \delta$ for all $i \in \mathbb{N}$. A run is *accepting* if the least priority of all states, which occur infinitely often, is even. We say that $\mathcal{A}_P$ accepts a word w if there is an accepting run on w. □

Concise Representation of a Play

A play in a game $MC(t, \varphi_0)$ can be concisely represented by just storing the rules between its configurations. Since the principal formula of a configuration uniquely determines which rule is applied we only have to remember which conjunct player *Albert* chooses at rule $(\wedge)$.

Definition 6.3 Given a play $P = C_0 C_1 \ldots$ of a game $MC(t, \varphi_0)$ then the concise representation for P is defined as $\tilde{P} = r_0 r_1 \ldots$ where

$$r_i := \begin{cases} \psi & \text{, if } \psi \text{ is the PF in } C_i \text{ and } \psi \neq \cdot \wedge \cdot \text{ and } \psi \neq O\cdot \\ \psi_1 \wedge_1 \psi_2 & \text{, if } \psi_1 \wedge \psi_2 \text{ is the PF in } C_i \text{ and player } \textit{Albert} \text{ chooses } \psi_1 \\ \psi_1 \wedge_2 \psi_2 & \text{, if } \psi_1 \wedge \psi_2 \text{ is the PF in } C_i \text{ and player } \textit{Albert} \text{ chooses } \psi_2 \\ O & \text{, if the PF in } C_i \text{ is of the form } O\psi \end{cases}$$

for all $i = 0, 1, \ldots$.

The set of *principal rules* is defined as

$$\begin{aligned} \tilde{R} := & \{\psi \mid \psi \in \mathit{Sub}(\varphi_0),\ \psi \neq \cdot \wedge \cdot \text{ and } \psi \neq O\cdot\} \\ & \cup \{\psi_1 \wedge_1 \psi_2 \mid \psi_1 \wedge \psi_2 \in \mathit{Sub}(\varphi_0)\} \\ & \cup \{\psi_1 \wedge_2 \psi_2 \mid \psi_1 \wedge \psi_2 \in \mathit{Sub}(\varphi_0)\} \\ & \cup \{O \mid O\psi \in \mathit{Sub}(\varphi_0) \text{ for some } \psi\}. \end{aligned}$$ □

6.2 Parity Automaton

We will construct an automaton which accepts a play of the form $\tilde{P}$ if and only if it has a ν-line. All lines in a play begin at φ_0, the formula which the game starts with. Only at rule $(\vee)$ a line can continue in two different directions. The idea behind the automaton is to follow a single line in the play non-deterministically and detect whether it is a ν-line or not.

Definition 6.4 Let φ_0 be a closed μTL formula. The NPA $\mathcal{A}_P(\varphi_0) := (S, A, \delta, \varphi_0, F)$ is defined by

- $S := Sub(\varphi_0)$
- $A := \tilde{\mathcal{R}}$
- $\delta :=$

$$
\begin{array}{ll}
(\psi_1 \wedge \psi_2, \quad \psi_1 \wedge_1 \psi_2, \quad \psi_1) & \\
(\psi_1 \wedge \psi_2, \quad \psi_1 \wedge_2 \psi_2, \quad \psi_2) & \\
(\psi_1 \wedge \psi_2, \quad r, \quad \psi_1 \wedge \psi_2) & \text{, where } r \in \tilde{R} \setminus \{\psi_1 \wedge_1 \psi_2, \psi_1 \wedge_2 \psi_2\} \\
(\psi_1 \vee \psi_2, \quad \psi_1 \vee \psi_2, \quad \psi_1) & \\
(\psi_1 \vee \psi_2, \quad \psi_1 \vee \psi_2, \quad \psi_2) & \\
(\psi_1 \vee \psi_2, \quad r, \quad \psi_1 \vee \psi_2) & \text{, where } r \in \tilde{R} \setminus \{\psi_1 \vee \psi_2\} \\
(O\psi, \quad O, \quad \psi) & \\
(O\psi, \quad r, \quad O\psi) & \text{, where } r \in \tilde{R} \setminus \{O\psi\} \\
(\nu Y.\psi, \quad \nu Y.\psi, \quad Y) & \\
(\nu Y.\psi, \quad r, \quad \nu Y.\psi) & \text{, where } r \in \tilde{R} \setminus \{\nu Y.\psi\} \\
(Y, \quad Y, \quad fb_{\varphi_0}(Y)) & \\
(Y, \quad r, \quad Y) & \text{, where } r \in \tilde{R} \setminus \{Y\} \\
(\mu X.\psi, \quad \mu X.\psi, \quad X) & \\
(\mu X.\psi, \quad r, \quad \mu X.\psi) & \text{, where } r \in \tilde{R} \setminus \{\mu X.\psi\} \\
(X, \quad X, \quad fb_{\varphi_0}(X)) & \\
(X, \quad r, \quad X) & \text{, where } r \in \tilde{R} \setminus \{X\}
\end{array}
$$

- Let $Z_1, Z_2, \ldots, Z_n$ denote all variables in S s.t. for all $i, j \in \mathbb{N} : Z_i <_{\varphi_0} Z_j \Rightarrow i > j$. (greater variables first)

  $$F(Z_i) := \begin{cases} 2i & \text{, if } Z_i \text{ is of type } \nu \\ 2i+1 & \text{, if } Z_i \text{ is of type } \mu \end{cases}$$

  $F(\psi) := 2n + 1$, for all remaining states $\psi \in S \setminus \mathcal{V}$. □

Theorem 6.5 *Let P be a play in $MC(t, \varphi_0)$ where t is a tree and φ_0 is a closed μTL formula.*

$$\mathcal{A}_P(\varphi_0) \text{ accepts } \tilde{P} \quad \text{iff} \quad \text{there is a } \nu\text{-line in } P.$$

Proof "$\Rightarrow$" First we show the "only if" direction. Suppose the NPA $\mathcal{A}_P(\varphi_0)$ accepts $\tilde{P}$. Then there is an infinite run $R = s_0 s_1 \ldots$ s.t. the least priority of all states which occur infinitely often is even. Intuitively, the automaton $\mathcal{A}_P(\varphi_0)$ draws the line R on the play by marking a formula $s_i \in C_i$ for each configuration C_i in the play.

Since $s_0 = \varphi_0$ it remains to show that every possible connection $(i, s_i, s_{i+1}) \in \mathcal{C}on_{\varphi_0}$ for all $i \in \mathbb{N}$. If s_i is not a principal formula in C_i then $\mathcal{A}_P(\varphi_0)$ remains in the same state, i.e. only transitions of the form (s_i, r, s_i) can be applied where $r \in \tilde{R}$ is a rule which does not change s_i. Otherwise, if s_i is not a principal formula in C_i then $\mathcal{A}_P(\varphi_0)$ changes its state to the formula $s_{i+1} \in C_{i+1}$. In other words, all transitions (s_i, r, s_{i+1}) which are applicable result in a state $s_{i+1} \in C_{i+1}$ s.t. $(i, s_i, s_{i+1}) \in \mathcal{C}on$.

In addition, the acceptance condition of $\mathcal{A}_P(\varphi_0)$ assures that there must be a ν-variable which occurs infinitely often on R. Let Y be the variable which occurs infinitely often with the least priority. There cannot be a greater μ-variable X, which occurs infinitely often, because otherwise $F(X) < F(Y)$ would hold. But that contradicts the acceptance of run R. Therefore there is a ν-line R in P.

$\Leftarrow$ Now we will show the "if" direction. Let $L = \varphi_0 \varphi_1 \ldots$ be the ν-line in P and let Y be the greatest ν-variable which occurs infinitely often on P. We will prove that L is an accepting run for $\mathcal{A}_P(\varphi_0)$ on $\tilde{P} = \tilde{\varphi}_0 \tilde{\varphi}_1 \ldots$.

The run starts with φ_0 as expected. So, we only have to ensure that the transition $(\varphi_i, \tilde{\varphi}_i, \varphi_{i+1})$ exists in δ. If φ_i is not a principal formula in C_i then $\varphi_{i+1} = \varphi_i$ and $\tilde{\varphi}_i$ does not rewrite formula φ_i. But this transition $(\varphi_i, r, \varphi_i)$ where r does not operate on φ_i is in δ. If φ_i is a principal formula in C_i then we have to deal with the following cases:

If $\varphi_i = \psi_1 \wedge \psi_2$ is a conjunction then $\varphi_{i+1} = \psi_j, j \in \{1, 2\}$ is the conjunct chosen by player *Albert*. Hence, $\tilde{\varphi}_i = \psi_1 \wedge_j \psi_2$ and transition $(\psi_1 \wedge \psi_2, \psi_1 \wedge_j \psi_2, \psi_j)$ exists in δ.

If $\varphi_i = \psi_1 \vee \psi_2$ is a disjunction then $\varphi_{i+1} = \psi_j, j \in \{1, 2\}$. But there are two transitions $(\psi_1 \vee \psi_2, \psi_1 \vee \psi_2, \psi_j) \in \delta$.

If φ_i is of the form $O\psi$, $\mu X.\psi$, $\nu Y.\psi$, X or Y then φ_{i+1} and $\tilde{\varphi}_i$ are deterministically defined. And for each form there is a transition $(\varphi_i, \tilde{\varphi}_i, \varphi_{i+1}) \in \delta$ in $\mathcal{A}_P(\varphi_0)$, as well.

At last, we check that $\mathcal{A}_P(\varphi_0)$ accepts this run L, i.e. the least priority of all states which occurs infinitely often is even. Since L is a ν-line there is no state $s \in S$ which occurs infinitely often in L s.t. $F(s)$ is odd and $F(s) < F(Y)$. We can disregard all such states s' which are not variables because by definition $F(s') > F(Y)$. Furthermore, only μ-variables greater than Y are mapped to an odd priority less than $F(Y)$. But these variables do not occur infinitely often in L. Therefore the parity automaton $\mathcal{A}_P(\varphi_0)$ accepts the run L. ∎

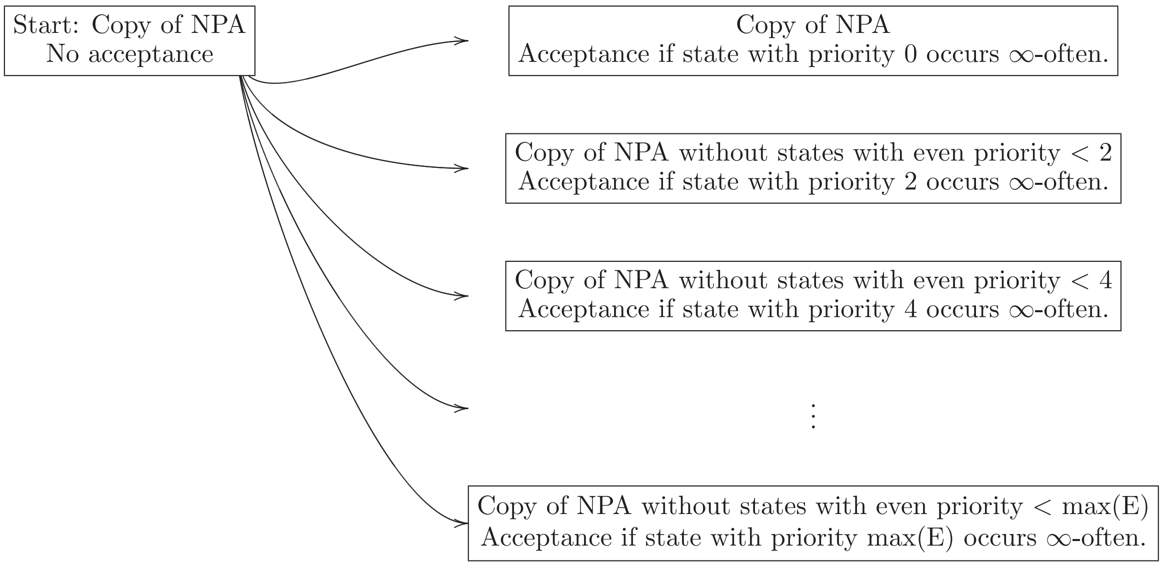


Table 6.1: Standard Transformation: NPA $\to$ NBA

6.3 Transformation to Büchi Automaton

There is a standard procedure for transforming a parity automaton into a Büchi automaton. Let $\mathcal{A}_P = (S, A, \delta, s_0, F)$ be an NPA. Then an NBA $\mathcal{A}_B := (S', A, \delta', s_0', F')$ can be constructed in the following way:

- $S' := (S \times E) \cup S$, where $E := \{s \in S \mid F(s) \text{ is even}\}$ is the set of all states with even priorities,
- $s_0' := s_0$,
- the NBA $\mathcal{A}_B$ can simulate the NPA $\mathcal{A}_P$ using the same transitions or it can switch to (s, e) by choosing a finite state e which must occur infinitely often. After that it can simulate the run of $\mathcal{A}_P$ in the first component of its state.

$$
\begin{aligned}
\delta' := \quad & \delta \quad \cup \\
& \{(s, a, (\hat{s}, e)) \mid (s, a, \hat{s}) \in \delta, e \in E\} \quad \cup \\
& \{((s, e), a, (\hat{s}, e)) \mid (s, a, \hat{s}) \in \delta, e \in E, F(\hat{s}) \geq F(e)\}
\end{aligned}
$$

- $F' := \{(e, e) \mid e \in E\}$.

This new NBA consists of $|E| + 1$ copies of the NPA, see Table 6.1. The transformation is not optimal and can be optimized by uniting states of E which have the same priority (i.e. $E := \{F(s) \mid F(s) \text{ even}\}$), for example. But the NBA outlined here is more similar to the specific ν-line NPA to NBA construction.

If the NPA accepts a word w then there is an accepting run where the least priority p of the states which occur infinitely often is even. That is, there must be a position m in the run after which no state s with $F(s) < p$ occurs. This can be simulated by the NBA by using the transitions of the NPA during the first m steps and then switching to (s, p) and imitating the run of the NPA on w on the first component.

On the other hand, if the NBA accepts a word w then there is a run where a state of the form (e, e), where $e \in E$, occurs infinitely often. By definition the run does not contain any state (s, e), where $F(s)$ is odd and $F(s) < F(e)$, which occurs infinitely often. Therefore the NPA accepts w by imitating the run of the NBA, as well.

Definition 6.6 Let φ be a closed μTL formula. A NBA $\mathcal{A}_B(\varphi_0) := (S, A, \delta, \varepsilon, F)$ is defined by

- $S := (Sub(\varphi_0) \times \mathcal{V}_\nu) \cup \{\varepsilon\}$
- $A := \tilde{\mathcal{R}}$
- $\delta :=$

$$
\begin{array}{llll}
(\varepsilon, & r, & \varepsilon) & \text{, where } r \in \tilde{R} \\
(\varepsilon, & Y, & (\mathit{fb}_{\varphi_0}(Y), Y)) & \\
((\psi_1 \wedge \psi_2, Y), & \psi_1 \wedge_1 \psi_2, & (\psi_1, Y)) & \\
((\psi_1 \wedge \psi_2, Y), & \psi_1 \wedge_2 \psi_2, & (\psi_2, Y)) & \\
((\psi_1 \wedge \psi_2, Y), & r, & (\psi_1 \wedge \psi_2, Y)) & \text{, where } r \in \tilde{R} \setminus \{\psi_1 \wedge_1 \psi_2, \psi_1 \wedge_2 \psi_2\} \\
((\psi_1 \vee \psi_2, Y), & \psi_1 \vee \psi_2, & (\psi_1, Y)) & \\
((\psi_1 \vee \psi_2, Y), & \psi_1 \vee \psi_2, & (\psi_2, Y)) & \\
((\psi_1 \vee \psi_2, Y), & r, & (\psi_1 \vee \psi_2, Y)) & \text{, where } r \in \tilde{R} \setminus \{\psi_1 \vee \psi_2\} \\
((O\psi, Y), & O, & (\psi, Y)) & \\
((O\psi, Y), & r, & (O\psi, Y)) & \text{, where } r \in \tilde{R} \\
((\nu Y'.\psi, Y), & \nu Y'.\psi, & (Y', Y)) & \\
((\nu Y.\psi, Y), & r, & (\nu Y.\psi, Y)) & \text{, where } r \in \tilde{R} \setminus \{\nu Y.\psi\} \\
((Y, Y), & Y, & (\mathit{fb}_{\varphi_0}(Y), Y)) & \\
((Y, Y), & r, & (Y, Y)) & \text{, where } r \in \tilde{R} \setminus \{Y\} \\
((\mu X.\psi, Y), & \mu X.\psi, & (X, Y)) & \\
((\mu X.\psi, Y), & r, & (\mu X.\psi, Y)) & \text{, where } r \in \tilde{R} \setminus \{\mu X.\psi\} \\
((X, Y), & X, & (\mathit{fb}_{\varphi_0}(X), Y)) & \text{, if } X <_{\varphi_0} Y \\
((X, Y) & r, & (X, Y)) & \text{, where } r \in \tilde{R} \setminus \{X\}
\end{array}
$$

- $F := \{(Y, Y) \mid Y \in Sub(\varphi_0)\}$ □

Theorem 6.7 *Let P be a play in $MC(t, \varphi_0)$ where t is a tree and φ_0 is a closed μTL formula.*

$$\mathcal{A}_B(\varphi_0) \text{ accepts } \tilde{P} \quad \text{iff} \quad \mathcal{A}_P(\varphi_0) \text{ accepts } \tilde{P}.$$

Proof "$\Rightarrow$" The "only if" direction is shown in the following way. If the NBA $\mathcal{A}_B(\varphi_0) = (S, A, \delta, \varepsilon, F)$ accepts $\tilde{P}$ then there is an infinite run $R = s_0 s_1 \ldots$ s.t. (Y, Y) occurs infinitely often on R for some ν-variable Y. Notice that once the second component of its state is set to variable Y, it is never changed along the run. Furthermore this Y can only be introduced by transition $(\varepsilon, Y, (\mathit{fb}_{\varphi_0}(Y), Y))$ where $\nu Y.\psi \in \mathit{Sub}(\varphi_0)$. In short, the automaton starts in state $s_0 := \varepsilon$ and stays there until it non-deterministically changes into state $s_{m+1} := (\mathit{fb}_{\varphi_0}(Y), Y)$ when it reads $\tilde{\varphi_m} = Y$:

$$R = \varepsilon \ldots \varepsilon \underbrace{(\mathit{fb}_{\varphi_0}(Y), Y)}_{s_{m+1}} \ldots (Y, Y) \ldots (Y, Y) \ldots$$

That is, Y must be the principal formula in configuration C_m and therefore a line $L = \varphi_0 \varphi_1 \ldots \varphi_m$ in the play P can be drawn which ends with $\varphi_m = Y$. Besides, all states after s_m are tuples with exactly two components. Define $s(1)$ as the first component for such a state s.

Now we can construct an accepting run R' for an NPA $\mathcal{A}_P(\varphi_0) = (S', A, \delta', \varphi_0, F')$ which is defined in definition 6.4. Intuitively, the NPA $\mathcal{A}_P(\varphi_0)$ begins with $s'_0 := \varphi_0$ and follows the line L until $s'_m := \varphi_m$. Then it imitates the run of the NBA $\mathcal{A}_B(\varphi_0)$ on $\tilde{P}^{(m+1)} = \tilde{\varphi}_{m+1} \tilde{\varphi}_{m+2} \ldots$. If the NBA $\mathcal{A}_B(\varphi_0)$ uses transition $((\varphi, Y), \tilde{\varphi}, (\psi, Y))$ the parity automaton chooses transition $(\varphi, \tilde{\varphi}, \psi)$ by disregarding the second component Y.

$$R' := \varphi_0 \ldots \varphi_m \underbrace{\mathit{fb}_{\varphi_0}(Y)}_{s'_{m+1}} \ldots Y \ldots Y \ldots$$

One can check that $R' = s'_0 s'_1 \ldots$ is indeed a run for the NPA $\mathcal{A}_P(\varphi_0)$, i.e. there exists a transition $(s'_{i-1}, \tilde{\varphi}_{i-1}, s'_i)$ for each $i \in \mathbb{N}$:

The prefix of R' is a line in the play P, for all $i \le m : (\varphi_{i-1}, \varphi_i) \in \mathcal{C}on$. That is, φ_{i-1} is a PF and φ_i is its rewriting, or $\varphi_{i-1} = \varphi_i$ is no PF. This is preserved by the transitions of the NPA $\mathcal{A}_P(\varphi_0)$. For example, $\varphi_0 \in C_0$ and if φ_{i-1} is a conjunct and the PF in C_{i-1} then $\tilde{\varphi}_{i-1}$ is the formula φ_{i-1} with the annotation $j \in \{1, 2\}$ which specifies the choice of player *Albert*. But the transition $(\varphi_{i-1}, \tilde{\varphi}_{i-1}, \varphi_i)$ where φ_i is the rewriting of φ_{i-1} is in δ'. If the conjunct φ_{i-1} is not a PF then the automaton reaches φ_i by transition $(\varphi_{i-1}, r, \varphi_i)$ where r is not the annotated φ_{i-1}. In both cases φ_i is in C_i again. All other cases are similar.

Since $s'_m = Y$ is the PF in C_m, i.e. $\tilde{\varphi}_m = Y$, the NPA $\mathcal{A}_P(\varphi_0)$ can go to state $s'_{m+1} = \mathit{fb}_{\varphi_0}(Y)$ by transition $(Y, Y, \mathit{fb}_{\varphi_0}(Y)) \in \delta'$.

Comparing the set of transitions δ and δ' of both automatons we see that

$$\{(\varphi, \quad \tilde{\varphi}, \quad \psi) \quad | \quad ((\varphi, Y), \quad \tilde{\varphi}, \quad (\psi, Y)) \quad \in \quad \delta\} \quad \subseteq \quad \delta'.$$

As mentioned above, the run of the NBA $\mathcal{A}_B(\varphi_0)$ after position m consists of tuples and so it only uses transitions of the form $((\varphi, Y), \tilde{\varphi}, (\psi, Y))$. Hence, the NPA $\mathcal{A}_P(\varphi_0)$ can reach every state s'_i, $i > m$, as well.

It remains to show that R' is accepting. In the run R the state (Y, Y) occurs infinitely often, i.e. in $\mathcal{R}'$ there are infinitely many states Y. Furthermore, no state which is a μ-variable X greater than Y occurs after s'_m. Otherwise the NBA $\mathcal{A}_B(\varphi_0)$ must have used a transition of the form $((X, Y), \tilde{\varphi}, (\mathit{fp}(X), Y))$ where $X > Y$. But this transition does not exist in δ. Therefore the least priority of all states which occur infinitely often is $F'(Y)$ which is even.

"$\Leftarrow$" The "if" direction is a little bit more complicated. If the NPA $\mathcal{A}_P(\varphi_0) = (S', A, \delta', \varphi_0, F')$ accepts the play $\tilde{P}$ then there must be an infinite run $R' = s'_0 s'_1 \ldots$ s.t. the least priority of all variables which occur infinitely often on R' is even. Let Y be the variable having the least priority of all variables which occurs infinitely often.

There must be a state Y in R' s.t. no greater variable occurs afterwards. Assume the opposite, i.e. after each state Y some greater variable occurs. Since the set of variables is finite there must be at least one variable greater than Y which occurs infinitely often on the run. But then Y would not be the variable with the least priority. Let $s'_m = Y$ be the first state with that property s.t. $s'_{m+1} = \mathit{fb}_{\varphi_0}(Y)$.

$$R' = s'_0 s'_1 \ldots \overbrace{\underbrace{Y}_{s'_m} \underbrace{\mathit{fb}_{\varphi_0}(Y)}_{s'_{m+1}} \ldots Y \ldots Y \ldots}^{\text{no variable } Z > Y}$$

With these information we can find an accepting run $R = s_0 s_1 \ldots$ for the Büchi automaton $\mathcal{A}_B(\varphi_0) = (S, A, \delta, \varepsilon, F)$ defined as in Definition 6.6. Define $s_i := \varepsilon$ for all $i = 0, 1, \ldots, m$ and for every state s_i after s_m define $s_i := (s'_i, Y)$ by adding the second component Y.

$$R := \varepsilon \ldots \overbrace{\underbrace{\varepsilon}_{s_m} \underbrace{(\mathit{fb}_{\varphi_0}(Y), Y)}_{s_n} \ldots (Y, Y) \ldots (Y, Y) \ldots}^{\text{no state } (Z, Y) \text{ where } Z > Y}$$

This is a valid run for the NBA $\mathcal{A}_B$ because

- with transition $(\varepsilon, r, \varepsilon)$ where r is some arbitrary rule, the NBA $\mathcal{A}_B(\varphi_0)$ can reach state s_m.
- In state s'_m the parity automaton changed into state $\mathit{fb}_{\varphi_0}(Y)$ by reading the formula $\tilde{\varphi}_m = Y$. In the same manner the Büchi automaton can proceed from state $s_m = \varepsilon$ to state $s_{m+1} = (\mathit{fb}_{\varphi_0}(Y), Y)$ by transition $(\varepsilon, Y, (\mathit{fb}_{\varphi_0}(Y), Y)) \in \delta$.
- Since after state s'_m no variable greater than Y occurs in run R' there cannot be any state (Z, Y) where $Z > Y$ after s_m in run R. Therefore the set of transitions needed to simulate the run of the NPA $\mathcal{A}_P$ on $\tilde{P}$ is contained in δ:

$$\{((\varphi, Y), \quad \tilde{\varphi}, \quad (\psi, Y)) \mid Y \in \mathcal{V}, \quad (\varphi, \quad \tilde{\varphi}, \quad \psi) \in \delta', \quad \varphi \neq Z > Y\} \quad \subseteq \quad \delta$$

Thus, the play $\tilde{P}$ is accepted by the NBA $\mathcal{A}_B$ because (Y, Y) occurs infinitely often in its run R. ∎

6.4 Deterministic Automata

Safra introduced a way to transform an NBA into a deterministic automaton which accepts by a Muller condition. The idea is a refinement of the classical subset construction using Safra trees as states. Theses trees help to recognize whether a run of the automaton passes a set of final states of the NBA infinitely often or not.

Let $\mathcal{A}_B = (S, A, \delta, s_0, F)$ be an NBA. The states in the deterministic Muller automaton (DMA) are Safra trees. Such a tree consists of nodes $n \in \mathbb{N} \times 2^S \times \{., !\}$ with a node name $\in \mathbb{N}$, a set of NBA states, and a light which can flash. We write $n!$ if the node n is highlighted. The initial state of the DMA is $(0, s_0, .)$ – a Safra tree with only one node. The successor state of this DMA is constructed in several steps:

a) For every node in the tree which contains final states of the NBA create a child with a new name, the set of all these final states and no flash.

b) Apply the usual subset construction for each node of the new tree, i.e. replace any node s' by $\{r \in S \mid \exists s \in s' : (s, a, r) \in \delta\}$ and remove a possible flash !.

c) For each node in the tree remove a final state s from the node if there is an older sibling which contains s. Then remove empty nodes.

d) If the union of sibling nodes equals the parent node then delete all siblings and their descendants and let the parent node flash.

Because of the last two steps the union of sibling nodes in a Safra tree is a proper subset of the parent node and therefore the maximal number of nodes is bounded by the number of Büchi states $|S|$.

The Muller automaton accepts a word if its run satisfies the following Muller condition: at least one node is missing only finitely often but is highlighted (!) infinitely often.

Lemma 6.8 *Safra's construction converts an NBA with $|S|$ states into a deterministic Muller automaton with $2^{O(|S| \cdot log(|S|))}$ states.*

Proof See [Tho97] for example. ∎

In the next theorem we develop an algorithm based on the DMA which solves the validity problem of a μTL formula in PSPACE.

Theorem 6.9 (Complexity) *Let $t \in \mathcal{T}_\Sigma$ be a tree over Σ and let φ be a closed μTL-formula.*

a) *The validity game $VAL(\varphi)$ is in PSPACE.*

b) *The satisfiability game $SAT(\varphi)$ is in PSPACE.*

c) *The model-checking game $MC(t, \varphi)$ is in PSPACE.*

PROOF a) Let $\mathcal{C}$ be the set of configurations of the validity game $VAL(\varphi)$ and let T be the set of states in the DMA constructed by Safra's method. First we show that $VAL(\varphi)$ is in NPSPACE.

A Deterministic Muller automaton allows us to decide whether or not an infinite play contains a ν-line. Due to its determinism each configuration C in a play can be annotated by a unique state t of the automaton. We will write (C, t) for such an annotated configuration. Since $|\mathcal{C}|$ and $|T|$ are bounded, there exist at most $P_{max} := |\mathcal{C}| \times |T| = 2^{|\varphi|} \times 2^{O(|\varphi|^2 \cdot log(|\varphi|^2))}$ different annotated configurations. Therefore, a loop of a play can be detected by a counter using space $log(P_{max})$ which is polynomial in the input $|\varphi|$.

If φ is not valid than player *Albert* can enforce a play which he wins. In case this play is infinite, it contains no ν-line and so, player *Albert* can find a loop which has no ν-line, non-deterministically.

Formula φ is valid if and only if player *Albert* is unable to win. Thus, this decision procedure rejects if player *Albert* would accept. A sketch of this algorithm is depicted in Table 6.2.

First all variables are initialised. Notice that the space needed for a configuration and a Safra tree is polynomial in the input because there are at most $|Sub(\varphi)|$ different formulas in a configuration and the number of nodes of a Safra tree is bounded by the number of NBA states which is in $O(|\varphi|^2)$. Moreover, the number of node names stored in the set "Nodes" is bounded by $2 * |\text{NBA states}|$ due to Safra's construction [Saf89].

Then a play which is directed by player *Albert* is played. If he is not able to win within P_{max} steps then φ is valid and the algorithm accepts. Otherwise player *Albert* can find a configuration (C', t') which will be repeated. To check that there is no ν-line in this loop, the Muller condition must hold, i.e. there is not a single node which permanently exists and which is highlighted at least once.

By Savitch's result [Sav69] – NPSPACE $\subseteq$ PSPACE – our algorithm can be simulated in PSPACE.

Algorithms for tree-games and satisfiability games of this work are designed in a similar fashion and it can be shown that all of them are in PSPACE, as well. The following sketches will show a way to proof these results.

b) The length of an annotated play is polynomial in the input $|\varphi|$ and hence the a play will be decided after at most P_{max} steps.

```
Configuration C', C := initial configuration;
Safratree t', t := initial Safra tree;
Rule r;
Counter c := 0;
Boolean b := false;
Nodes N := ∅;


While c <= P_max Do
  If WC a) applies on C Then accept;
  If WC c) applies on C Then reject;

  Albert chooses whether the lines shall be executed:
    b := true;
    (C', t') := (C, t);
    N := { n | n is a node in t};
  End;

  C' := Albert determines the next successor of C;
  r := rule which has rewritten C to C';
  t' := δ_DMA(t, r);
  (C, t) := (C', t');

  If b Then
    N := N ∩ { n | n is a node in t};
    If (C, t) := (C', t') Then
      If {n | n! ∈ T} = ∅ Then reject;
    End;
  End;

  c := c + 1;
End;

accept;
```

Table 6.2: Algorithm for Deciding Validity

If φ is satisfiable, player *Eliza* can enforce a play where she wins. In case the play is infinite, she is able to find a loop which does not contain a μ-line. In these cases the algorithm accepts φ, otherwise it rejects.

c) We assume that the tree t is represented by a finite graph with N distinct nodes. Thus, an annotated play has at most $N \times P_{max}$ different configurations.

If $t \models \varphi$ then player *Albert* can enforce a play where he wins. If the resulting play is infinite then he is able to find a loop which does not contain a ν-line within $N \times P_{max}$ steps. In these cases the algorithm accepts (t, φ), otherwise it rejects. ∎

A Implementation

The decision procedure for validity checking of a μTL formula is implemented in OCaml (Objective Categorically Abstract Machine Language), a functional programming language which belongs to the ML language family. This implementation will outline a translation of validity games and its ν-line automata into concrete Ocaml code.

A μTL formula φ_0 is parsed from the command line. Then, as described in Table 6.2, the algorithm tries to find a play in $VAL(\varphi_0)$ where player *Albert* wins. If such a play exists, the input formula φ_0 is not valid and the algorithm prints out the annotated play. Otherwise, φ_0 must be valid.

Example A.1 Let w be an infinite word of the form $(a^+b)^\omega$ with $\Sigma = \{a, b\}$, i.e. property b occurs infinitely often but never successively. This run can be described by the following μTL formula

$$\varphi_0 := G(F\,b) \wedge G(\neg(b \wedge Ob))$$

where $G(\varphi) := \nu Z.\varphi \wedge OZ$ (generally) and $F(\varphi) := \mu Z.\varphi \vee OX$ (finally). Since $w \models \varphi_0$ the negated formula $\neg\varphi_0$ cannot be valid.

First we have to transform $\neg\varphi_0$ into a formula which can be parsed by the Ocaml program.

$$\begin{aligned}
\neg\varphi_0 &= \neg G(F\,b) \vee \neg G(\neg(b \wedge Ob)) \\
&= F(G\,\neg b) \vee F(\neg\neg(b \wedge Ob)) \\
&= F(G\,a) \vee F(b \wedge Ob) \\
&= (\mu X.G\,a \vee OX) \vee \mu Y.(b \wedge Ob) \vee OY \\
&= (\mu X.(\nu Z.a \wedge OZ) \vee OX) \vee \mu Y.(b \wedge Ob) \vee OY
\end{aligned}$$

The formula is not valid and therefore our algorithm returns a play where opponent *Albert* wins. The play is annotated by Safra trees, the states of the DMA:

```
~/tex/ocaml> ocaml main.ml ”(mu␣X.(nu␣Z.a␣&␣OZ)␣|␣OX)␣|␣mu␣Y.(b␣&␣Ob)␣|␣OY”
VAL Game

Sigma := ab
phi0 := ((mu X.((nu Z.(a /\ O Z)) \/ O X)) \/ (mu Y.((b /\ O b) \/ O Y)))

-----
1:  ((mu X.((nu Z.(a /\ O Z)) \/ O X)) \/ (mu Y.((b /\ O b) \/ O Y))),  |      (00:  e,  { })
```

```
2: (mu X.((nu Z.(a /\ O Z)) \/ O X)), (mu Y.((b /\ O b) \/ O Y)),  |    (00: e, { })
3: X, (mu Y.((b /\ O b) \/ O Y)),  |    (00: e, { })
4: ((nu Z.(a /\ O Z)) \/ O X), (mu Y.((b /\ O b) \/ O Y)),  |    (00: e, { })
5: O X, (mu Y.((b /\ O b) \/ O Y)), (nu Z.(a /\ O Z)),  |    (00: e, { })
6: Z, O X, (mu Y.((b /\ O b) \/ O Y)),  |    (00: e, { })
7: (a /\ O Z), O X, (mu Y.((b /\ O b) \/ O Y)),  |    (00: e, ((a /\ O Z), Z), { })
8: a, O X, (mu Y.((b /\ O b) \/ O Y)),  |    (00: e, (a, Z), { })
9: a, Y, O X,  |    (00: e, { })
10: a, ((b /\ O b) \/ O Y), O X,  |    (00: e, { })
11: a, (b /\ O b), O X, O Y,  |    (00: e, { })
12: a, O b, O X, O Y,  |    (00: e, { })
13: b, X, Y,  |    (00: e, { })
14: b, Y, ((nu Z.(a /\ O Z)) \/ O X),  |    (00: e, { })
15: b, Y, O X, (nu Z.(a /\ O Z)),  |    (00: e, { })
16: b, Y, Z, O X,  |    (00: e, { })
17: b, Y, (a /\ O Z), O X,  |    (00: e, ((a /\ O Z), Z), { })
18: b, Y, O X, O Z,  |    (00: e, (O Z, Z), { })
19: b, ((b /\ O b) \/ O Y), O X, O Z,  |    (00: e, (O Z, Z), { })
20: b, (b /\ O b), O X, O Y, O Z,  |    (00: e, (O Z, Z), { })
21: b, O X, O Y, O Z,  |    (00: e, (O Z, Z), { })
22: X, Y, Z,  |    (00: e, (Z, Z), { })
23: X, Y, (a /\ O Z),  |    (00: e, ((a /\ O Z), Z), { (01: ((a /\ O Z), Z), { }) })
24: a, X, Y,  |    (00: e, (a, Z), { (01: (a, Z), { }) })
25: a, Y, ((nu Z.(a /\ O Z)) \/ O X),  |    (00: e, { })
26: a, Y, O X, (nu Z.(a /\ O Z)),  |    (00: e, { })
27: a, Y, Z, O X,  |    (00: e, { })
28: a, Y, (a /\ O Z), O X,  |    (00: e, ((a /\ O Z), Z), { })
29: a, Y, O X,  |    (00: e, (a, Z), { })
30: a, ((b /\ O b) \/ O Y), O X,  |    (00: e, { })
-----


Result: phi0 is not valid

~/tex/ocaml>
```

The last configuration of the play is a repeat of configuration number 10 and between theses configurations no node of the Safra tree has been highlighted (there is no exclamation mark next to a node name). Therefore player *Albert* wins that play.

If we gather all letters of the configurations on which the next rule has been applied to, namely configurations 12 and 21, we obtain the following sequence:

$$v := \begin{tikzcd} {} \arrow[r] & 10 \arrow[r] & 12 \arrow[r] & 21 \arrow[r] & 30 \arrow[lll, bend right] \\ & & a & b & & \cdots \end{tikzcd}$$

Notice that the "negated" word v, i.e. we replace a by $\neg a$ and b by $\neg b$ in v, is a model of φ_0.

On the following pages we present the source code of the implementation. The syntax of a *μTL* formula is defined separately in *Formula.ml*:

```
(* data type of a muTL formula *)
type muTL =
   Prop of string
   | Var of string
   | Or of muTL * muTL
   | And of muTL * muTL
   | Next of muTL
   | Mu of string * muTL
   | Nu of string * muTL;;
```

The Ocaml code which implements the NBA and DMA automaton and the $VAL(\varphi_0)$ game is given in the following listing:

```
#load "lexer.cmo";;
#load "parser.cmo";;
#use "Formula.ml";;

(* contains type muTL for mu-calculus formulas *)
open Formula;;

(****************************************************************)
(* A set is represented by a list with distinct elements *)
(****************************************************************)
module Set = struct
   (* inserts an element into a set *)
   let insert a l = if List.mem a l then l else a :: l;;

   (* true if set1 is a subset of set2 *)
   let rec isSubset = function
      [] -> (fun _ -> true)
      | h :: t -> (fun l -> List.mem h l && isSubset t l);;

   (* true if set1 = set2 *)
   let isEqual s1 s2 = (isSubset s1 s2) && (isSubset s2 s1);;

   (* results the union of two lists *)
   let rec union set1 set2 =
      let set1' = List.rev set1 in
      let rec union = function
         [] -> set2
         | h :: t -> insert h (union t)
      in
      union set1';;

   (* one-level flattening of sets (with duplicate check) *)
```

```
  let rec flatten = function
      [] -> []
    | h :: t -> union h (flatten t) ;;

  (* returns all elements which are in s1 and not in s2, i.e. set := s1 \ s2 *)
  let rec minus = function
      [] -> fun _ -> []
    | h :: t -> fun s2 -> if List.mem h s2 then minus t s2 else h :: minus t s2;;

end;;


(*****************************************************************)
(* Definition and several operations on muTL formulas *)
(*****************************************************************)
module MuTL = struct
  (* returns the fixed point of z w.r.t. phi0 *)
  let fp phi0 z =
    let rec fp' = function
        (Or (phi1, phi2), z) -> if fp' (phi1, z) != None then fp' (phi1, z) else fp' (phi2, z)
      | (And (phi1, phi2), z) -> if fp' (phi1, z) != None then fp' (phi1, z) else fp' (phi2,
          z)
      | (Next phi, z) -> fp' (phi, z)
      | (Mu (x, phi), z) -> if x = z then Some (Mu (x, phi)) else fp' (phi, z)
      | (Nu (y, phi), z) -> if y = z then Some (Nu (y, phi)) else fp' (phi, z)
      | _ -> None in
    let remove_option = function
        None -> failwith "fp: Fixed point not found."
      | Some a -> a in
    remove_option (fp' (phi0,z)) ;;

  (* returns the fixed point body of z w.r.t. phi0 *)
  let fb phi0 z =
    let removeSigma = function
        Mu (_, phi) -> phi
      | Nu (_, phi) -> phi
      | _ -> failwith "fb: Function fp did not return a fixed point." in
    removeSigma (fp phi0 z);;

  (* returns the list Sub(phi) *)
  let rec sub = function
      Prop a -> [Prop a]
    | Var z -> [Var z]
    | Or (phi1, phi2) -> Or (phi1, phi2) :: sub phi1 @ sub phi2
    | And (phi1, phi2) -> And (phi1, phi2) :: sub phi1 @ sub phi2
    | Next phi -> Next phi :: sub phi
    | Mu (x, phi) -> Mu (x, phi) :: sub phi
    | Nu (y, phi) -> Nu (y, phi) :: sub phi;;

  (* <_phi0 relation *)
```

```
let less phi0 x y =
  List.mem (fp phi0 x) (sub (fb phi0 y));;

(* returns true if the variable is of type nu *)
let vartype phi z =
  let get_type = function
      Mu _ -> false
    | Nu _ -> true
    | _ -> failwith "MuTL.vartype: Type cannot be infered." in
  get_type (fp phi z);;

(* returns the negated formula *)
let rec negate sigma =
  let rec bigor = function
      [] -> failwith "MuTL.negate: Sigma is empty"
    | a :: [] -> Prop a
    | a :: t -> Or(Prop a, bigor t) in function
  Prop a -> bigor (Set.minus sigma [a])
  | Var x -> Var x
  | Or (phi1, phi2) -> And (negate sigma phi1, negate sigma phi2)
  | And (phi1, phi2) -> Or (negate sigma phi1, negate sigma phi2)
  | Next phi -> Next (negate sigma phi)
  | Mu (z, phi) -> Nu (z, negate sigma phi)
  | Nu (z, phi) -> Mu (z, negate sigma phi);;

(* returns all letters of the formula *)
let rec getLetters = function
  Prop a -> [a]
  | Var x -> []
  | Or (phi1, phi2) -> Set.union (getLetters phi1) (getLetters phi2)
  | And (phi1, phi2) -> Set.union (getLetters phi1) (getLetters phi2)
  | Next phi -> (getLetters phi)
  | Mu (z, phi) -> (getLetters phi)
  | Nu (z, phi) -> (getLetters phi);;

(* ---------- string - convertions for printing ----------*)
let fromStr str =
  let lexbuf = Lexing.from_string str in
  let result = Parser.main Lexer.token lexbuf in
  result ;;

let rec phiToStr = function
  Prop a -> a ^ ""
  | Var v -> v ^ ""
  | Or (phi1, phi2) -> "(" ^ (phiToStr phi1) ^ " \\/ " ^ (phiToStr phi2) ^ ")"
  | And (phi1, phi2) -> "(" ^ (phiToStr phi1) ^ " /\\ " ^ (phiToStr phi2) ^ ")"
  | Next phi -> "O " ^ (phiToStr phi)
  | Mu (x, phi) -> "(mu " ^ x ^ "." ^ phiToStr phi ^ ")"
  | Nu (x, phi) -> "(nu " ^ x ^ "." ^ phiToStr phi ^ ")";;
```

```
  let rec phiListToStr = function
      [] -> ""
      | h :: t -> (phiToStr h) ^ ", " ^ phiListToStr t;;

end;;


(****************************************************************)
(* NBA automaton with states, alphabet (game rules) and transition *)
(****************************************************************)
module Nba = struct
  (* data type of an NBA state *)
  type state =
      Epsilon
      | Tuple of muTL * string;;

  (* data type for the short representation of game rules *)
  type rules =
      NextRule
      | Formula of muTL
      | LAnd of muTL
      | RAnd of muTL;;

  (* transition function of the nu-line NBA *)
  let delta phi0 = function
      Epsilon, Formula (Var y) -> if MuTL.vartype phi0 y then [Epsilon; Tuple (MuTL.fb phi0 y, y)] else [Epsilon]
      | Epsilon, _ -> [Epsilon]
      | Tuple (And (phi1, phi2), y), LAnd psi -> if And (phi1, phi2) = psi then [Tuple (phi1, y)] else [Tuple (And (phi1, phi2), y)]
      | Tuple (And (phi1, phi2), y), RAnd psi -> if And (phi1, phi2) = psi then [Tuple (phi2, y)] else [Tuple (And (phi1, phi2), y)]
      | Tuple (And (phi1, phi2), y), _ -> [Tuple (And (phi1, phi2), y)]
      | Tuple (Or (phi1, phi2), y), Formula psi -> if Or (phi1, phi2) = psi then [Tuple (phi1, y); Tuple (phi2, y)] else [Tuple (Or (phi1, phi2), y)]
      | Tuple (Or (phi1, phi2), y), _ -> [Tuple (Or (phi1, phi2), y)]
      | Tuple (Next phi, y), NextRule -> [Tuple (phi, y)]
      | Tuple (Next phi, y), _ -> [Tuple (Next phi, y)]
      | Tuple (Nu (y', phi), y), Formula psi -> if Nu (y', phi) = psi then [Tuple (Var y', y)] else [Tuple (Nu (y', phi), y)]
      | Tuple (Nu (y', phi), y), _ -> [Tuple (Nu (y', phi), y)]
      | Tuple (Var z, y), Formula psi ->
         if MuTL.vartype phi0 z then
            (if Var z = psi then [Tuple (MuTL.fb phi0 z, y)] else [Tuple (Var z, y)])
         else
            (if Var z = psi then
               (if MuTL.less phi0 z y then [Tuple (MuTL.fb phi0 z, y)] else [])
            else [Tuple (Var z, y)])
      | Tuple (Var z, y), _ -> [Tuple (Var z, y)]
      | Tuple (Mu (x, phi), y), Formula psi -> if Mu (x, phi) = psi then [Tuple (Var x, y)] else
```

```
        [Tuple (Mu (x, phi), y)]
    | Tuple (Mu (x, phi), y), _ -> [Tuple (Mu (x, phi), y)]
    | _ -> [];;  (* no transition for the rest *)

  (* true if the argumet is a final NBA state *)
  let isFinal phi = function
    Epsilon -> false
    | Tuple (Var z, y) -> if (z = y) && (MuTL.vartype phi y) then true else false
    | _ -> false;;

  (* ---------- string - convertions for printing ----------*)
  let stateToStr = function
    Epsilon -> "e"
    | Tuple (phi, y) -> "(" ^ MuTL.phiToStr phi ^ ",␣" ^ y ^ ")";;

  let rec stateListToStr = function
    [] -> ""
    | h :: t -> stateToStr h ^ ",␣" ^ stateListToStr t ;;

  let ruleToStr = function
    NextRule -> "NextRule"
    | Formula phi -> MuTL.phiToStr phi
    | LAnd phi -> MuTL.phiToStr phi
    | RAnd phi -> MuTL.phiToStr phi;;

end;;


(*****************************************************************)
(* DMA automaton with states, alphabet (game rules) and transition *)
(*****************************************************************)
module Dma = struct
  (* data type of an DMA state *)
  type safratree =
    (* age, name, flash, NBA states, children *)
    Node of int * int * bool * Nba.state list * safratree list ;;

  (* only node names of this list may be used *)
  let age = ref 10;;
  let nodeIDs = ref [];;

  (* ---------- string - convertions for printing ----------*)
  let rec nodeToStr = function
    Node (a, i, b, states, _) ->
      if b then (string_of_int a)^( string_of_int i) ^ "!:␣" ^ Nba.stateListToStr states
      else ( string_of_int a)^( string_of_int i) ^ ":␣" ^ Nba.stateListToStr states ;;

  let rec treeToStr = function
    Node (a, i, b, states, children) ->
      let nStr = nodeToStr (Node (a, i, b, states, children)) in
```

```
        let c = List.map treeToStr children in
        let cStr = List.fold_left (^) " " c in
        "(" ^ nStr ^ "{" ^ cStr ^ "}) ";;

let rec intListToStr = function
    [] -> ""
  | h::t -> string_of_int h ^ ", " ^ intListToStr t;;

let rec nodeListToStr = function
    [] -> ""
  | (i, false) :: t -> string_of_int i ^ ", " ^ nodeListToStr t
  | (i, true) :: t -> string_of_int i ^ "!, " ^ nodeListToStr t;;
(* ---------- *)

(* returns a "normalized" tree, i.e. age=0, sorted states & children *)
let rec treeNormalize = function
    Node (_, i, b, ss, cs) ->
      let ss' = List.sort compare ss in
      let cs' = List.sort compare (List.map treeNormalize cs) in
      Node (0, i, b, ss', cs');;

(* returns list of nodenames and their flash of a list of trees *)
let rec nodesOf = function
    [] -> []
  | Node (_, i, b, _, children) :: tail -> (i, b) :: nodesOf children @ nodesOf tail;;

(* returns a finite list of possible node names: 0, 1, .., 2*|phi0|^2 *)
let createNames phi0 =
    let rec nList = function
        0 -> [] (* name "0" is used by the tree of the test formula *)
      | n -> n :: nList (n-1) in
    let n = List.length (MuTL.sub phi0) in
    List.rev (nList (2*n*n));;

(* transition of the deterministic Muller automaton *)
let delta phi0 t isFinal delta rule =
    (* -- I. -- returns a tree whith new children containing the final states *)
    let rec splitFinalStates =
        (* returns set of final states from a list *)
        let rec getFStates = function
            [] -> []
          | h :: t ->
              if isFinal h then Set.insert h (getFStates t) else
              getFStates t in function

        Node (a, i, b, states, children) ->
            let childrenDone = List.map splitFinalStates children in
            let fStates = getFStates states in
            if fStates = [] then
                Node (a, i, b, states, childrenDone)
```

```
        else
          let newChild = Node (!age+1, List.hd !nodeIDs, false, fStates, []) in
          if !age < 1000000000 then age := !age + 1 else failwith "Dma.delta: age overflow";
          nodeIDs := List.tl !nodeIDs; (* pop node name *)
          Node (a, i, b, states, newChild :: childrenDone) in

  (* -- II. -- classical subset construction applied on a Dma state (=Safra tree) *)
  let rec subsetDelta =
    let compose = function q -> Nba.delta phi0 (q, rule) in function
    Node (a, i, _, states, children) ->
      Node (a, i, false, Set.flatten (List.map compose states), List.map subsetDelta children) in

  (* -- III. --- horizontal merge *)
  let rec hMerge =
    (* union of the states of all older nodes *)
    let rec mergeOlderSiblings a' = function
        [] -> []
      | Node (a, j, b, states, _) :: t ->
          if a<a' then states @ mergeOlderSiblings a' t else
          mergeOlderSiblings a' t in

    (* merge siblings *)
    let rec mergeSiblings siblings = function
        [] -> []
      | Node (a, i, b, states, children) :: t ->
          let afterDelete = Set.minus states (mergeOlderSiblings a siblings) in
          if afterDelete = [] then
            (let l = List.map fst (nodesOf children) @ !nodeIDs in (* push children names *)
            nodeIDs := i :: l; (* push node name *)
            mergeSiblings siblings t)
          else
            Node (a, i, b, afterDelete, children) :: mergeSiblings siblings t in function

    Node (a, i, b, states, []) -> Node (a, i, b, states, [])
    | Node (a, i, b, states, children) ->
      Node (a, i, b, states, List.map hMerge (mergeSiblings children children)) in

  (* -- IV. --- vertical merge *)
  let rec vMerge =
    (* union of states of all nodes *)
    let rec unionOfStates = function
        [] -> []
      | Node (a, j, b, states, _) :: t -> states @ unionOfStates t in function

    Node (a, i, b, states, []) -> Node (a, i, b, states, [])
    | Node (a, i, b, states, children) ->
      if Set.isEqual states (unionOfStates children) then
```

```
                (let l = List.map fst (nodesOf children) @ !nodeIDs in
                nodeIDs := l;   (* push node names *)
                Node (a, i, true, states, []) )
            else
                Node (a, i, b, states, List.map vMerge children) in

    vMerge (hMerge (subsetDelta (splitFinalStates t))) ;;

end;;


(*****************************************************************)
(* Implementation of the VAL game *)
(*****************************************************************)
module Game = struct
  (* this variable stores a whole play *)
  let play = ref [];;

  (* winning conditions a) and c) for letters *)
  let isWCa sigma conf =
    Set.isSubset (List.map (function a -> Prop a) sigma) conf;;
  let isWCc sigma conf =
    (Set.isSubset conf (List.map (function a -> Prop a) sigma)) &&
    not (isWCa sigma conf);;

  (* detects a nu-Line: union of all nodes from the last conf until the begin of the loop *)
  let nuLineExists lastStep pl =
    (* intersection of node names; notice: {5} /\ {5!} becomes {5!} *)
    let rec intsec l = function
        [] -> []
      | (n, false) :: t ->
          if List.mem (n, true) l then (n, true) :: intsec l t else
          if List.mem (n, false) l then (n, false) :: intsec l t else
          intsec l t
      | (n, true) :: t ->
          if List.mem (n, true) l || List.mem (n, false) l then (n, true) :: intsec l t else
          intsec l t in

    let rec intsecNodes = function
        [] -> failwith "intsecNodes: Loop not found."
      | (c,t) :: tl ->
          if (c, t) = lastStep then Dma.nodesOf [t] else
          intsec (Dma.nodesOf [t]) (intsecNodes tl) in

    let rec hasFlashNode = function
        [] -> false
      | (i,true) :: tl -> true
      | (i,false) :: tl -> hasFlashNode tl in

    let nodes = intsecNodes pl in
```

```
    (* print_string (Dma.nodeListToStr nodes ^ "\n");*)
    hasFlashNode nodes;;

(* returns a list of all possible (two at most) configurations which can follow *)
let nextConfs phi0 conf =
    let changeFirstComponent op = function (a, b) -> (op a, b) in
    let rec nextCs = function
        [] -> [([], Nba.NextRule)] (* only dummy rule *)
        | Prop p :: t -> List.map (changeFirstComponent (Set.insert (Prop p))) (nextCs t)
        | Var z :: t -> [(Set.insert (MuTL.fb phi0 z) t, Nba.Formula (Var z))]
        | Or (phi1, phi2) :: t -> [(Set.insert phi1 (Set. insert phi2 t), Nba.Formula (Or
            (phi1, phi2)))]
        | And (phi1, phi2) :: t -> [(Set.insert phi1 t, Nba.LAnd (And (phi1, phi2)));
            (Set.insert phi2 t, Nba.RAnd (And (phi1, phi2)))]
        | Next phi :: t -> List.map (changeFirstComponent (Set.insert (Next phi))) (nextCs t)
        | Mu (x, phi) :: t -> [(Set.insert (Var x) t, Nba.Formula (Mu (x, phi)))]
        | Nu (y, phi) :: t -> [(Set.insert (Var y) t, Nba.Formula (Nu (y, phi)))] in

    let rec applyNext = function
        [] -> []
        | Prop p :: t -> applyNext t
        | Next phi :: t -> Set.insert phi (applyNext t)
        | _ -> failwith "applyNext: Next rule can only apply on Prop or Next operators."
            in

    let applyNextIfNecessary = function
        (c, Nba.NextRule) :: t -> [(applyNext c, Nba.NextRule)]
        | x -> x in

    applyNextIfNecessary (nextCs conf);;

(* returns a list which contains the next possible confs. and DMA states *)
let nextStep phi0 c t =
    List.map (fun (a, rule) -> (a, Dma.delta phi0 t (Nba.isFinal phi0) (Nba.delta phi0) rule))
            (nextConfs phi0 c);;

(* ---------- string - convertions for printing ----------*)
let rec playToStr pl =
    let playLen = List.length pl in
    let revPlay = List.rev pl in
    let rec convert = function
        [] -> ""
        | (c,t) :: tl ->
            ( string_of_int (playLen - List.length tl)) ^
            ": " ^ (MuTL.phiListToStr c) ^ "   |    " ^
            Dma.treeToStr t ^ "\n" ^ (convert tl)
    in
    ("\n-----\n" ^ convert revPlay ^ "-----\n\n");;
(* ---------- *)
```

```
    (* returns true if Ex wins *)
    let rec valGame sigma phi0 (c,t) pl =
        let t' = Dma.treeNormalize t in
        let c' = List.sort compare c in

        if isWCa sigma c then true else
        if isWCc sigma c then (print_string (playToStr ((c', t') :: pl)); false) else

        if List.mem (c', t') pl then
            if nuLineExists (c', t') pl then true else
            (print_string (playToStr ((c', t') :: pl)); false)
        else
            let pl' = (c', t') :: pl in
            let nSteps = nextStep phi0 c t in
            if not (valGame sigma phi0 (List.hd nSteps) pl') then false else
            if List.tl nSteps != [] then valGame sigma phi0 (List.nth nSteps 1) pl' else
            true;;

    (* initializes the first (C, t) and calls valGame *)
    let startValGame sigma phi0 =
        Dma.nodeIDs := Dma.createNames phi0;
        valGame sigma phi0 ([phi0], Dma.Node(0, 0, false, [Nba.Epsilon], [])) [];;

end;;


(******************************************************************)
(* main function *)
(******************************************************************)
let main =
    let phi0 = MuTL.fromStr Sys.argv.(1) in
    let sigma = MuTL.getLetters phi0 in

    (* call of the game *)
    print_string "VAL Game\n\n";
    print_string ("Sigma := " ^ (List.fold_left (^) "" sigma ^ "\n"));
    print_string ("phi0 := " ^ (MuTL.phiToStr phi0) ^ "\n");
    if Game.startValGame sigma phi0 then
        print_string "\nResult: phi0 is valid\n\n" else
        print_string "\nResult: phi0 is not valid\n\n";;
```

References

[BEM96] J. Bradfield, J. Esparza, and A. Mader. An effective tableau system for the linear time mu-calculus. In F. Meyer auf der Heide and B. Monien, editors, *Proc. of ICALP'96*, number 1099 in Lecture Notes in Computer Science, pages 98–109. Springer-Verlag, 1996.

[BKP86] H. Barringer, R. Kuiper, and A. Pnueli. A Really Abstract Concurrent Model and Its Temporal Logic. In *Conf. Record of the 13th Annual ACM Symp. on Principles of Programming Languages, POPL'86*, pages 173–183. ACM, ACM, 1986.

[EH81] E. A. Emerson and J. Y. Halpern. Synthesis of Synchronization Skeletons for Branching Time Temporal Logic. In *Logics of Programs: Workshop*, volume 131 of *LNCS*, pages 52–71, Yorktown Heights, New York, 1981. Springer.

[EH85] E. A. Emerson and J. Y. Halpern. Decision Procedures and Expressiveness in the Temporal Logic of Branching Time. *Journal of Computer and System Sciences*, 30:1–24, 1985.

[Gar05] Simson Garfinkel. History's Worst Software Bugs. *Wired News*, 11, 2005.

[GPSS53] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. Infinite games of perfect information. *Ann. Math. Studies*, 28:245–266, 1953.

[GPSS80] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. The Temporal Analysis of Fairness. In *Proc. 7th Symp. on Principles of Programming Languages, POPL'80*, pages 163–173. ACM, 1980.

[GTW02] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

[HMU02] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie*. Pearson Studium, München, 2002. 2., überarbeitete Auflage.

[Hoa83] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 26(1):53–56, 1983.

[JW96] D. Janin and I. Walukiewicz. On the expressive completeness of the propositional μ-calculus with respect to monadic second order logic. In U. Montanari and V. Sassone, editors, *Proc. 7th Conf. on Concurrency Theory, CONCUR'96*, volume 1119 of *LNCS*, pages 263–277, Pisa, Italy, 1996. Springer.

[Kai95] R. Kaivola. A Simple Decision Method for the Linear Time μ-calculus. In J. Desel, editor, *Proc. Int. Workshop on Structures in Concurrency Theory, STRICT'95*, pages 190–204, Berlin, 1995.

[Kai97] R. Kaivola. *Using Automata to Characterise Fixed Point Temporal Logics*. PhD thesis, LFCS, Division of Informatics, The University ofEdinburgh, 1997.

[Kam68] H. W. Kamp. *On tense logic and the theory of order*. PhD thesis, Univ. of California, 1968.

[Klo05] Karlhorst Klotz. Software ohne Fehl und Tadel. *Technology Review*, 7:10–23, 2005.

[Koz83] Dexter Kozen. Results on the Propositional μ-Calculus. *Theoretical Computer Science*, 27(3):333–354, 1983.

[Lan05] M. Lange. Weak automata for the linear time μ-calculus. In R. Cousot, editor, *Proc. 6th Int. Conf. on Verification, Model Checking and Abstract Interpretation, VMCAI'05*, volume 3385 of *LNCS*, pages 267–281, 2005.

[Mar75] D. A. Martin. Borel determinacy. *Ann. Math.*, 102:363–371, 1975.

[Mat02] R. Mateescu. Local Model-Checking of Modal Mu-Calculus on Acyclic Labeled Transition Systems. In *Proc. 8th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'02*, volume 2280 of *LNCS*, pages 281–295. Springer, 2002.

[McF93] Michael C. McFarland. Formal Verification of Sequential Hardware: A Tutorial. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(5):633–654, 1993.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. 18th Symp. on Foundations of Computer Science, FOCS'77*, pages 46–57, Providence, RI, USA, October 1977. IEEE.

[Saf89] Shmuel Safra. *Complexity of Automata on Infinite Objects*. PhD thesis, The Weizman Institute of Science, Rehovot, Israel, March 1989.

[Sav69] W. J. Savitch. Deterministic simulation of non-deterministic Turing Machines. In *Proc. 1st ACM Symposium on Theory of Computing (STOC)*, pages 247–248, 1969.

[Sti95] Colin Stirling. Local Model Checking Games. In *Proceedings of the 6th International Conference on Concurrency Theory, CONCUR '95*, volume 962 of *Lecture Notes in Computer Science*, pages 1–11. Springer-Verlag, 1995.

[Sti97] C. Stirling. Bisimulation, Model Checking and Other Games, 1997. Notes for Mathfit instructional meeting on games and computation,Edinburgh, June 1997.

[SW91] Colin Stirling and David Walker. Local model checking in the modal mu-calculus. In *TAPSOFT '89: 2nd international joint conference on Theory and practice of software development*, pages 161–177, Amsterdam, The Netherlands, The Netherlands, 1991. Elsevier Science Publishers B. V.

[Tar55] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[Tho79] W. Thomas. Star-Free Regular Sets of ω-Sequences. *Information and Control*, 42(2):148–156, 1979.

[Tho97] W. Thomas. Languages, Automata and Logic. In A. Salomaa and G. Rozenberg, editors, *Handbook of Formal Languages*, volume 3, Beyond Words. Springer-Verlag, Berlin, 1997.

[Tho03] W. Thomas. Infinite Games and Verification (Extended Abstract of a Tutorial). In *Proc. 15th Int. Conference on Computer Aided Verification, CAV'03*, volume 2725 of *LNCS*, pages 58–64. Springer, 2003.

[Var88] M. Y. Vardi. A temporal fixpoint calculus. In ACM, editor, *Proc. Conf. on Principles of Programming Languages, POPL'88*, pages 250–259, NY, USA, 1988. ACM Press.

[Wal00] Igor Walukiewicz. Completeness of Kozen's axiomatisation of the propositional μ-calculus. *Inf. Comput.*, 157(1-2):142–182, 2000.

[Win93] Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction.* MIT Press, 1993.