Cost Fixed Point Logic

MSc Thesis *(Afstudeerscriptie)*

written by

Anouk Michelle Oudshoorn
(born November 1st, 1999 in Nijmegen, The Netherlands)

under the supervision of **Dr. Bahareh Afshari**, and submitted to the Examinations Board in partial fulfillment of the requirements for the degree of

MSc in Logic

at the *Universiteit van Amsterdam.*

Date of the public defense:	**Members of the Thesis Committee:**
August 25, 2022	Dr. Maria Aloni (chair)
	Dr. Bahareh Afshari
	Dr. Balder ten Cate
	Dr. Sebastian Enqvist

Institute for Logic, Language and Computation

Abstract

Cost automata, introduced under the name 'distance parity automata' in [10], are in fact automata with something extra: counters. In the most simple case, the automaton has one counter which, during a transition, we can increase, check, reset or perform a combination of these actions. This machinery is very powerful. It can for instance count the number of a's appearing in a word.

Cost automata stand out from other automata because they recognise cost functions instead of languages. The counters are used to evaluate the 'cost' of different runs, which gives rise to a function from words or trees to the natural numbers. Noteworthy is that these counters are initialised for the purpose of bookkeeping certain patterns in the run and as such their actual values are not so important to us.

Next to cost automata, cost games are quite well understood and elaborated on too, but the logical counterpart has remained underdeveloped. In this thesis we propose a notion of cost formulas that connects well to the automata and game counterpart. This is obtained by enriching the modal μ-calculus with an appropriate form of 'counters' that can represent cost functions. We further establish a sequent calculus for a fragment of out logical framework which is sound and complete with respect to our 'cost semantics', though we leave open the question of utilising the system for deciding boundedness.

Acknowledgements

First of all, I would like to thank my supervisor, Bahareh Afshari, a lot, for being the best supervisor and mentor I could wish for. Every time I was stuck at a certain point, you let me explain the problem exactly and if that did not solve my question, you gave me a push in the right direction. Moreover, our meetings were very inspiring; afterwards, I wanted to get back to work as soon as I could to work on all the new ideas.

Another great characteristic is that you push me to look further than what is right in front of me. When I feel I have found something that I particularly like, whether it is a high school, a topic for my master thesis or a PhD position, I have a tendency to just go for it and decide that that is it. As a mentor, you managed this very well. We still need to play Istanbul though. And of course, too much tea does not exist.

I also want to thank my fellow students in the MoL for the nice conversations and games we played: the game nights really helped me through the pandemic and it is a shame that I had almost all my courses online. Especially, I would like to thank Anton: the study sessions with you were fun, and we even managed to get things done.

A special thanks goes to Luc, my family and my friends (I should not have chosen another high school). After each weekend spend with them, I am completely charged for another week of hard work.

Contents

List of Figures

1 Introduction

The origin of cost automata lies in the star height problem: the problem of deciding, given a regular language, how many nestings of Kleene star is needed in a regular expression to represent that language. For quite some time this was an open problem, until Hashigushi gave a first proof for the star height problem over finite words spread over four different papers, in which he introduced distance automata. See for instance [14]. Later, Kirsten, [15], provided an easier alternative proof using a more general class of nested distance desert automata.

The proof by Kirsten inspired Colcombet and Löding to introduce cost automata under the name distance-parity automata in [10]. With these kinds of automata, they were also able to extend the result of Hashigushi to finite trees. Next to this, they work on a proof for another decidability problem in [10]: given a regular language and a degree of the Mostowski hierarchy, decide whether the given language is in that particular degree.

What all of these automata have in common is that they possess some kind of counting features. If we focus on cost automata, an easy example of what we can count is the number of a's appearing in a word. The idea is that a cost automaton assigns a value, a natural number or infinity, to each input structure, whether it is a word or a tree. This way, we can see a cost automaton as a function from structures to numbers. The function that is connected to a cost automaton is the cost function that is recognised by that automaton. We call such a cost function a regular cost function. A cost function differs from the standard idea of functions in the sense that the precise outcome does not really matter; the emphasis lies more on whether the result is bounded or infinite.

The name 'cost automata' might be a bit misleading. The word 'cost' seems to imply a connection with paying and similar concepts. But actually nothing more happens than counting how often several structures within an input structure appear. Moreover, most people do care about the exact value if we talk about costs.

A nice result of dispensing the preciseness of the outcome is that it may become possible to decide whether a cost function is bounded by another. Note that a cost function differs from the standard notion of a function and thus that we cannot use $\leq$. Instead, we look at the relation $\preceq$, which is a weakening of $\leq$. This question of whether a cost function is bounded by another will be the main focus of this thesis.

For some classes of cost automata, namely cost automata over finite words, finite trees and infinite words, it is known that we can decide whether a regular cost function is bounded by another. However, for cost automata over infinite trees, the decidability is only known for a subclass of all regular cost functions, namely the cost functions that are recognised by a nondeterministic cost automaton over infinite trees. It is useful to know whether we can decide this: among others the problem of the Mostowski hierarchy can be reduced to deciding whether a cost function is bounded by another.

In this thesis, we begin with some background, consisting of a short introduction to automata over different structures, some words on the theory of cost

functions and ending with a compact overview of modal μ-calculus. This will be helpful in chapter 3, where cost automata are introduced in detail. In chapter 4, we introduce an extension to modal μ-calculus which we call cost modal μ-calculus and study its properties. Chapter 5 is concerned with proof theory for cost modal μ-calculus, where a sound and complete sequent calculus for a fragment of our cost modal μ-calculus is developed.

1.1 Contributions

The contributions of this thesis are presented in chapters 4 and 5 and include the following.

1. We introduce a notion of cost formulas. Roughly speaking, we add counter actions as 1-ary symbols to the syntax of modal μ-calculus. Similar to cost automata, these counter actions effect a set of counters that are part of determining the value of a cost formula. There exists a B- and an S-version of this logic. We proof that these versions are equivalent.

2. We connect this notion to alternating cost automata over infinite trees. More precisely, we show how to translate a cost formula into a cost automaton, and how to translate a cost automaton into a cost formula. We conclude that the introduced cost formulas are exactly as expressive as alternating cost automata over infinite trees.

3. We prove a Normal Form Theorem for B-cost formulas. That is, we show that it is enough to have the counter action ic and to use it only in combination with a least fixed point.

4. We design a sequent calculus for checking boundedness. Our sequent calculus works for checking whether a plain formula φ is bounded by a cost S-formula ψ, write $\varphi \preceq \psi$. We show the system is sound and complete and conjecture a possible way to extend this result.

2 Preliminaries

In this section we will discuss the background that is needed to understand the rest of this thesis. The topics include automata theory in general, cost functions and the modal μ-calculus. Before we get to this, we fix some notation.

2.1 Notation

By $\mathbb{A}$ we denote a set of letters, named the alphabet, which we can use to form words or label nodes in structures. With $\mathbb{A}^*$ we denote the set of all finite concatenations of letters in $\mathbb{A}$, i.e. a finite word. $\mathbb{A}^\omega$ is used to denote an infinite word. A language L is a (possibly infinite) set of words or trees. The domain $\mathcal{D}_\mathbb{A}$ is the set of all finite and infinite words or trees that can be made of an alphabet $\mathbb{A}$. When it is clear from the context, or it does not really matter which alphabet is used, the subscript $\mathbb{A}$ is left out.

For $u \in \mathbb{A}^* \cup \mathbb{A}^\omega$, a finite or infinite word, define $|u| \in \mathbb{N}_\infty$ as the length of the string u. We might also want to tell how many letters of a certain type we have. We use $|u|_a$ to denote the cardinality of the set of positions where an a occurs in the word u.

A node in the infinite binary tree can be represented as an element of $\{0,1\}^*$. Thus, we can see a labelled binary tree t as a function $t : \{0,1\}^* \to \mathbb{A}$ sending nodes to their label. The set of infinite binary trees labelled with letters from the alphabet $\mathbb{A}$ is denoted by $\mathcal{T}_\mathbb{A}$. $\mathcal{T}$ is the unique infinite binary tree without labels.

With $\mathcal{S} = (S, \to, \rho)$ we denote a transition system, consisting of a set of states S, the accessibility relation $\to \subseteq S \times S$ and a valuation map $\rho : P \to \mathcal{P}(S)$ sending propositions in P to subsets of S. In a pointed transition structure we distinguish one particular state: a beginning state in a sense. This is indicated by $(\mathcal{S}, s_0)$, where s_0 is the distinguished state.

We use the notation $\mathcal{B}^+(A)$ to denote all positive boolean combinations of some set A, written as a disjunction of conjunctions of elements of A.

2.2 Automata theory

What follows is a very basic overview of different kinds of finite state automata over different finite and infinite structures. We start with the most easy form of automata: automata over finite words. After that, we get to automata over infinite words and infinite trees. Before we start, we first define the notion of a regular language.

Definition 2.1. A language L is a *regular language* when there exists an automaton $\mathcal{A}$ with a finite set of states that recognises this language, meaning that $u \in L$ if and only if u is accepted by $\mathcal{A}$.

This definition extends to all the automata described in this chapter: if a language is for instance accepted by an automaton over finite trees, we call a language a regular language over finite trees.

2.2.1 Finite words

An automaton over finite words is some machinery that defines a language. It exists of states and transitions with labels. More precisely, an automaton over finite words is $\mathcal{A} = (Q, \mathbb{A}, q_0, Fin, \Delta)$, where Q is a set of states, $\mathbb{A}$ is the alphabet over which the words are formed, q_0 is the starting state, Fin is the set of accepting or final states and $\Delta \subseteq Q \times \mathbb{A} \times Q$ is the set of transitions.

A word is accepted by the automaton when there exists a path starting in the initial state and ending in one of the final states such that the labels on the transitions form the word. A path in the automaton is called a run for a particular word and the language that is formed like this by the automaton $\mathcal{A}$ is denoted by $L_{\mathcal{A}}$.

Example 2.2. The automaton $\mathcal{A}_0 = (\{q_0, q_1\}, \{a, b\}, q_0, \{q_0\}, \Delta)$, where

$$\Delta = \{(q_0, a, q_1), (q_0, b, q_0), (q_1, a, q_0), (q_1, b, q_1)\},$$

accepts all finite words with an even amount of a's. Thus, the language recognised by $\mathcal{A}_0$ is $L_{\mathcal{A}_0} = \{u \in \{a, b\}^* : \exists n \in \mathbb{N}. |u|_a = 2n\}$.

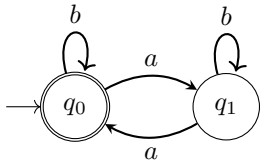

Figure 2.1: The automaton $\mathcal{A}_0$.

This is an example of a deterministic automaton: when we are at a node and read a letter, there is only one arrow that fits with the description. In other words, an automaton is deterministic when Δ is functional in $Q \times \mathbb{A}$. When an automaton is deterministic, it follows that given one input word, there is only one possible path through the automaton.

This clearly does not follow when we are considering a nondeterministic automaton. In that case there can be multiple runs for a word, where a run is a path through the automaton that starts at the starting state and follows arrows corresponding to the letters in the word. We say that a word is accepted by the automaton when there exists a run that ends in a final state.

2.2.2 Infinite words

Before, we defined acceptance by having a run that ends in an accepting state. But, since infinite words do not end, we need to think of something different for infinite words. A possible solution is to assign a priority (natural number) to each state and say that a run is accepting if the highest priority occurring infinitely often is even. We call these kind of automata parity automata.

Such an automaton is thus of the form $\mathcal{A}_1 = (Q, \mathbb{A}, q_0, \Omega, \Delta)$, where, Q is a set of states, $\mathbb{A}$ is the alphabet over which the words are formed, q_0 is the starting state, $\Omega : Q \to \mathbb{N}$ is the priority function and $\Delta \subseteq Q \times \mathbb{A} \times Q$ is the transition condition. An example follows.

Example 2.3. The automaton $\mathcal{A}_1 = (\{q_0, q_1\}, \{a, b\}, q_0, \Omega, \Delta)$, where

$$\Delta = \{(q_0, a, q_0), (q_0, b, q_1), (q_1, q, q_0), (q_1, b, q_1)\},$$

$\Omega(q_0) = 1$ and $\Omega(q_1) = 2$, accepts all infinite words where after each occurrence of an a, there will be at least one occurrence of a b.

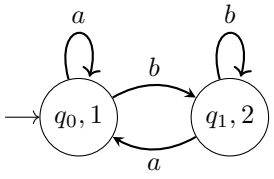

Figure 2.2: The automaton $\mathcal{A}_1$.

Again, this is an deterministic example: the definition of determinacy stays the same when switching to infinite words. If there are multiple possible runs for an infinite word, we say that the word is accepted if in at least one of these runs the highest priority seen infinitely often is even.

There also exist other automata that deal with infinite structures, like Büchi, Muller, Rabin and Streett. If interested in more of this, among others, the book *Automata, Logics and Infinite Games* [13], gives a detailed overview.

2.2.3 Nondeterministic automata over infinite trees

There are multiple types of automata over infinite trees one can consider. The two types that are most relevant for this thesis are called nondeterministic and alternating automata. Alternating automata are of a more general form and can capture nondeterministic automata. Here we define the automata over infinite binary trees, but the definitions can easily be generalised to n-ary trees.

So, let us start with defining a nondeterministic tree automaton $\mathcal{A}$ over a Σ-labelled (full) binary tree as follows.

Definition 2.4. A *nondeterministic automaton $\mathcal{A}$ over infinite trees* is a tuple $(Q, \Sigma, q_0, \Delta, Acc)$, where Q is a finite set of states and q_0 is the initial state, $\Delta \subseteq Q \times \Sigma \times Q \times Q$ is the transition relation, and Acc is an acceptance condition.

This tree automaton walks through the given Σ-tree and descends from a node to both child nodes, which means that our automaton splits itself into two copies. Both copies then proceed with traversing the tree independently.

Deciding whether a tree is accepted by this automaton happens by using a runtree. A runtree of a tree automaton $\mathcal{A}$ is a 2-branching infinite tree labelled

with states of the automaton, such that the root has label q_0 and the labels respect the transition relation Δ.

Definition 2.5. A *runtree* is a function $\rho : \{0,1\}^* \to Q$ such that $\rho(\epsilon) = q_0$ and for all $u \in \{0,1\}^*$, we have $(\rho(u), t(u), \rho(u0), \rho(u1)) \in \Delta$.

A runtree is accepting if every path through this tree satisfies the acceptance condition *Acc*, which can be Büchi, Muller, Rabin, parity or Streett. A tree is accepted by $\mathcal{A}$ if there exists a runtree that fulfils the acceptance condition. Again acceptance means that that particular tree is in the language coded by $\mathcal{A}$.

In the rest of this thesis in general we will use the parity acceptance condition. This entails that every state is assigned a priority, a natural number. A run is accepting when on every branch of the runtree the highest priority occurring infinitely often is even. With this acceptance condition we can recognise exactly the same languages as with the accepting conditions Muller, Rabin and Streett, but it is strictly more expressive than the Büchi acceptance condition for nondeterministic automata.

Example 2.6. The set of $\{a,b\}$-labelled binary trees t such that t has a path with infinitely many a's, is recognised by the nondeterministic parity automaton $\mathcal{A} = (\{q_a, q_b, T\}, \{a,b\}, q_a, \Delta, \Omega)$, with

$$\Delta : \begin{cases} (q_*, a) & \mapsto \{(q_a, T), (T, q_a)\}, \\ (q_*, b) & \mapsto \{(q_b, T), (T, q_b)\}, \\ (T, *) & \mapsto \{(T, T), \} \end{cases}$$

such that $*$ is either a or b and $\Omega(q_a) = \Omega(T) = 2$ and $\Omega(q_b) = 1$.

Now suppose we have a given $\{a,b\}$-labelled tree. Every possible runtree over this tree corresponds to following one path of the tree and checking whether that path has infinitely many a's: the path in the runtree labelled by q_a's and q_b's is that path. The rest of the runtree is labelled by T. For the tree t, the following is a possible start of a runtree.

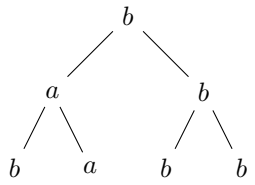

Figure 2.3: The beginning of the tree t.

A runtree is accepting when every path in the runtree contains infinitely many q_a's or T's. All paths that are not the path you chose to check for infinitely many a's are thus accepted: they contain infinitely many T's by construction. The path that you chose must contain infinitely many a's to fulfil the parity

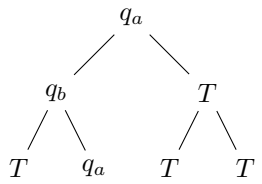

Figure 2.4: The start of one of t's runtrees.

condition (if there was an a in the original tree, q_a will appear as a label in the runtree), since there are no T's on this path.

We can conclude that $\mathcal{A}$ indeed codes the required language: if there is a path with infinitely many a's, there exists a runtree that is accepting. If on the other hand we have a runtree that is accepting, it must mean that the chosen path contains infinitely many a's.

2.2.4 Alternating automata over infinite trees

Recall that in nondeterministic automata over infinite trees, we were sending exactly one copy of the automaton to each child node: $\Delta \subseteq Q \times \Sigma \times Q \times Q$, where the last two Q's represent the states where the left and child node will be in. When making the step to alternating automata, we let go of this restriction of sending exactly one copy to each child node. Instead, we also allow to send zero or multiple copies.

This way, we can for instance state things like, when we are in state q and reading the letter $\sigma \in \Sigma$, we want that there is an accepting run from our left successor in state q' or we want an accepting run from our left successor in state q'' and from our right successor in state q'''.

Definition 2.7. An *alternating automaton $\mathcal{A}$ over infinite trees* is a tuple $(Q, \Sigma, q_0, \delta, Acc)$, where Q is a finite set of states and q_0 is the initial state, δ is the transition function, and Acc is an acceptance condition.

Notice that alternating automata are more general than their nondeterministic versions. If we restrict δ to be of the form that there is an accepting run from our left successor in state q and one from our right successor in state q', the result is actually just nondeterministic automata.

As we will see, alternating automata correspond nicely to the modal μ-calculus. Before we get there, we first discuss cost functions.

2.3 Cost functions

Cost functions were introduced in [8]. This introduction was highly motivated by an extension to the monadic second order logic: $MSO + \mathbb{U}$, or in words, monadic second order logic with the unbounding quantifier. In this logic, the sentence $\mathbb{U}X.\varphi(X)$ expresses "there is no bound on the size of sets X satisfying φ."

The main idea of a cost function is to give some kind of value, the cost, to a structure. Nevertheless, we do not care about the exact value of a structure, but about whether the cost is bounded or infinite. This is achieved by defining a cost function as an equivalence class of functions: we say that two functions are equivalent when they are bounded on the same input.

As noted before, the word 'cost' can be a bit misleading; when speaking about a 'cost', the idea of having to pay something is related, but that part is missing here. Speaking about these functions as 'count functions', where we mean a relaxed, not very precise, form of counting, might be more intuitive.

First, we give a precise definition of what it means for a function to be bounded.

Definition 2.8. We say a function $f : \mathcal{D} \to \mathbb{N} \cup \{\infty\}$ is *bounded on some set* $U \subseteq \mathcal{D}$ if there is some $n \in \mathbb{N}$ such that $f(u) \leq n$ for all $u \in U$.

This gives us the tools to define the equivalence relation $\approx$.

Definition 2.9 (domination preorder)**.** $f \preceq g$ if for all $U \subseteq \mathcal{D}$ we have: if g is bounded on U, then so is f. We write $f \approx g$ when $f \preceq g$ and $g \preceq f$.

It is easy to see that this is indeed an equivalence relation: of course $f \preceq f$, and thus reflexivity follows. If $f \approx g$, then $f \preceq g$ and $g \preceq f$, and thus $g \approx f$. Transitivity follows directly from the fact that $f \preceq g$ is transitive too. Now we can be sure that the following definition is well-defined.

Definition 2.10. A *cost function* f is an equivalence class of $\approx$.

We find that cost functions over some set E ordered by $\preceq$ form a lattice. Take the maximum of two cost functions for the lowest upper bound and the infimum for the highest lower bound.

Another way to compare two functions is by a correction function.

Definition 2.11. A *correction function* $\alpha : \mathbb{N} \to \mathbb{N}$ is a non-decreasing function that satisfies $\alpha(n) \geq n$ for all n. We write $f \preceq_\alpha g$ when $f(u) \leq \alpha(g(u))$ for all $u \in \mathcal{D}$.

This way of comparing is equivalent to the domination preorder.

Lemma 2.12 ([20])**.** *$f \preceq g$ if and only if there is some correction function α such that $f \preceq_\alpha g$.*

Notice that $\preceq$ and $\approx$ are weakened versions of $\leq$ and $=$, as we can see in the following lemma.

Lemma 2.13. *When $\alpha(n) = n$, $\preceq_\alpha$ coincides with the relation $\leq$ and $\approx_\alpha$ with $=$.*

We can express a language with a cost function, but not every cost function corresponds to a language.

This becomes clear when we express a language as a cost function by its characteristic mapping:

$$\chi_L(u) := \begin{cases} 0 & \text{if } u \in L, \\ \infty & \text{otherwise.} \end{cases}$$

Thus, the class of cost functions is a strict extension of the class of languages, when seen as cost functions.

This extension becomes strict as soon as the domain of cost functions becomes infinite: if so, we can find cost functions that do not correspond to any language. Consider for instance the map that sends a word to the length of that word.

We find that we can also express language inclusion.

Lemma 2.14. *$\chi_L \preceq \chi_{L'}$ if and only if $L' \subseteq L$.*

2.4 Modal μ-calculus

After this short introduction into the theory of cost functions, the last part of this chapter consists of a short overview of the modal μ-calculus. First, we will give the syntax and semantics. What follows is an overview of the use of games for the modal μ-calculus, including a game semantics. We end this chapter with the connection between modal μ-calculus and alternating automata on infinite trees.

The content, except for the last part on the equivalence with alternating tree automata, is based on the course *Logic, games and automata*, given by Bahareh Afshari at the University of Amsterdam [1]. The main sources for these slides on modal μ-calculus are [4] and [11]. Originally, the logic in this form was introduced by Kozen in 1983 [16].

2.4.1 Syntax and semantics

Very shortly, modal μ-calculus can be described as modal logic with least, μ, and greatest, ν, fixed point operators. This gives us the possibility to quantify over paths: we can express statements like "on every a-path ..." and "there exists an a-path such that ...". It is an important logic in the sense that it is an expressive logic, way stronger than LTL and S1S, and still robustly decidable.

Define the syntax of the modal μ-calculus as

$$\varphi ::= p \mid \neg p \mid Z \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \Box\varphi \mid \Diamond\varphi \mid \mu Z.\varphi(Z) \mid \nu Z.\varphi(Z),$$

for p a proposition and $Z \in Var$ a variable.

Definition 2.15. We call a modal μ-calculus formula φ *closed* when every variable X in φ occurs within the scope of a fixed point operator σX, for $\sigma \in \{\mu, \nu\}$.

Let us assume that we do not use the same variable for two different fixed point operators.

Remark that we are using $\Box$ and $\Diamond$ here, instead of $[a]$ and $\langle a \rangle$. This choice is made to align with the cost version of this logic introduced later. In this thesis the labels on infinite trees will be on the nodes, and not on the arrows pointing to the next node. This is nothing more than a choice of convention. By using propositions p, we can express exactly the same.

An intuitive way to think about something expressed with a least fixed point operator is as a property that has to happen eventually. As will mostly become clear from the game semantics, this means that we can loop through a least fixed point operator at most finitely often. On the other hand we have the greatest fixed point operator, through which it is good to loop infinitely often. With those operators, we can express 'safety' properties, like "it is always possible to visit an a".

Definition 2.16. The set of *subformulas* of φ, denoted by $Sub(\varphi)$, is defined inductively as

- if $\varphi = p$ for a proposition p, $Sub(\varphi) = \{p\}$,
- if $\varphi = \neg p$ for a proposition p, $Sub(\varphi) = \{\neg p\}$,
- if $\varphi = Z$ for a variable Z, $Sub(\varphi) = \{Z\}$,
- if $\varphi = \psi_1 * \psi_2$ for $* \in \{\wedge, \vee\}$, $Sub(\varphi) = Sub(\psi_1) \cup Sub(\psi_2) \cup \{\psi_1 * \psi_2\}$,
- if $\varphi = @\psi$ for $@ \in \{\Box, \Diamond\}$, $Sub(\varphi) = Sub(\psi) \cup \{@\psi\}$,
- if $\varphi = \sigma Z.\psi(Z)$ for $\sigma \in \{\mu, \nu\}$, $Sub(\varphi) = Sub(\psi(Z)) \cup \{\sigma Z.\psi(Z)\}$.

Before we continue with the semantics, we need the notion of a monotone function.

Definition 2.17. A function f is *monotone* if and only if

$$x \leq y \quad \text{implies} \quad f(x) \leq f(y).$$

If we only loop through a least fixed point finitely often, we can view that as looking for the property within that same amount of steps. Thus, we start with the empty set, and increase that with every loop until we reach some stability in the sense that we do not add any new states with a new loop. The opposite happens for a greatest fixed point: we start with all states, and with every loop get rid of some of them until we find stability.

To remind which states we are exactly considering for a fixed point variable we have the function $V : Var \to \mathcal{P}(S)$, mapping a variable Z to a subset of states $V(Z)$. As just described, we want to be able to change this. The following definition makes this possible,

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

The precise definition of the monotone function connected to a fixed point formula φ is $f_{\varphi,Z,V} : \mathcal{P}(S) \to \mathcal{P}(S)$ such that $U \mapsto ||\varphi||^t_{V[Z\to U]}$. Note that for a least fixed point formula, we have that $x \leq y$ corresponds to $x \subseteq y$, while for greatest fixed point this is turned around to $x \supseteq y$. The theorem of Knaster-Tarski gives us that, because of the monotonicity, we can always find a fixed point: $f(x) = x$.

Given a transition system $\mathcal{S} = (S, \to, \rho)$, the semantics $||\varphi|| \subseteq S$ of a formula φ is defined as

$$
\begin{aligned}
||\bot||^{\mathcal{S}}_V &:= \emptyset,\\
||\top||^{\mathcal{S}}_V &:= S,\\
||p||^{\mathcal{S}}_V &:= \rho(p),\\
||\neg p||^{\mathcal{S}}_V &:= S - \rho(p),\\
||Z||^{\mathcal{S}}_V &:= V(Z),\\
||\varphi \wedge \psi||^{\mathcal{S}}_V &:= ||\varphi||^{\mathcal{S}}_V \cap ||\psi||^{\mathcal{S}}_V,\\
||\varphi \vee \psi||^{\mathcal{S}}_V &:= ||\varphi||^{\mathcal{S}}_V \cup ||\psi||^{\mathcal{S}}_V,\\
||\Box\varphi||^{\mathcal{S}}_V &:= \{x \in S : \text{ for all } y \text{ such that } x \to y, \text{ we have } y \in ||\varphi||^{\mathcal{S}}_V\},\\
||\Diamond\varphi||^{\mathcal{S}}_V &:= \{x \in S : \text{ there exists } y \text{ such that } x \to y \text{ and } y \in ||\varphi||^{\mathcal{S}}_V\},\\
||\mu X.\varphi||^{\mathcal{S}}_V &:= \bigcap\{S' \subseteq S : ||\psi||_{V[X\to S']} \subseteq S'\},\\
||\nu X.\varphi||^{\mathcal{S}}_V &:= \bigcup\{S' \subseteq S : S' \subseteq ||\psi||_{V[X\to S']}\}.
\end{aligned}
$$

We use the notation $\mathcal{S}, s \models \varphi$ when $s \in ||\varphi||^{\mathcal{S}}_{\emptyset}$.

The following approximations can be used to calculate these fixed points. The idea used is that as soon as we do not see any change by applying $f_{\varphi,Z,V}$ one extra time, in formula $f^{\alpha}_{\varphi,Z,V} = f^{\alpha+1}_{\varphi,Z,V}$, we have found our least, or greatest, fixed point, depending on the input. As said, we start with the empty set for least fixed points and the full set S for greatest fixed points:

$$
\begin{aligned}
||\mu^0 Z.\varphi||^{\mathcal{S}}_V &:= \emptyset,\\
||\mu^{\alpha+1} Z.\varphi||^{\mathcal{S}}_V &:= f_{\varphi,Z,V}(||\mu^{\alpha} Z.\varphi||^{\mathcal{S}}_V),\\
&= ||\varphi||^{\mathcal{S}}_{V[Z\mapsto||\mu^{\alpha} Z.\varphi||^{\mathcal{S}}_V]}\\
||\mu^{\lambda} Z.\varphi||^{\mathcal{S}}_V &:= \bigcup_{\alpha<\lambda} ||\mu^{\alpha} Z.\varphi||^{\mathcal{S}}_V,
\end{aligned}
$$

and

$$
\begin{aligned}
||\nu^0 Z.\varphi||^{\mathcal{S}}_V &:= S,\\
||\nu^{\alpha+1} Z.\varphi||^{\mathcal{S}}_V &:= f_{\varphi,Z,V}(||\nu^{\alpha} Z.\varphi||^{\mathcal{S}}_V)\\
&= ||\varphi||^{\mathcal{S}}_{V[Z\mapsto||\nu^{\alpha} Z.\varphi||^{\mathcal{S}}_V]},\\
||\nu^{\lambda} Z.\varphi||^{\mathcal{S}}_V &:= \bigcap_{\alpha<\lambda} ||\nu^{\alpha} Z.\varphi||^{\mathcal{S}}_V.
\end{aligned}
$$

In example 4.21 we use such an approximation.

2.4.2 Games and modal μ-calculus

Instead of using approximations or the semantics $||\varphi||$, we can look at a game semantics, as we will do here. Good sources for more information are [18], [19] and [22].

The idea is that we define a game that, given a transition system $\mathcal{S}$, a state s and a closed modal μ-calculus formula φ, decides whether $s \in ||\varphi||^{\mathcal{S}}_{\emptyset}$. This is the model checking problem. We also have the satisfiability problem; given a closed modal μ-calculus formula, decide whether there exists a structure $\mathcal{S}$ with a state s such that $s \in ||\varphi||^{\mathcal{S}}_{\emptyset}$, but this will be less relevant in this thesis.

In the game there are two players, Eloise, or Verifier, V, who tries to verify the formula, and Abelard, or Refuter, R, who does the opposite. The game that is played over the formula φ and starts in state s in the structure $\mathcal{S}$ is denoted by $\mathcal{G}^{\mathcal{S}}_{V}(s, \varphi)$. If a player has a strategy that lets him or her win, independent of the choices of the other player, we say that he/she has a winning strategy. A strategy is a set of rules telling the player how to play. Knowing who has a winning strategy directly resolves the model checking problem.

Proposition 2.18. *$\mathcal{S}, s \models \varphi$ if and only if there is a winning strategy for V in $\mathcal{G}^{\mathcal{S}}_{V}(s, \varphi)$.*

This is a restatement of the *Fundamental Semantic Theorem* by Streett and Emerson [19].

Moreover, we have a result which means that for each position there either exists a winning strategy for Eloise, or a winning strategy for Abelard, since we can view games for the modal μ-calculus as parity games. We call a class of games determined when there exists a winning strategy for one of the players.

Proposition 2.19 ([12],[17])**.** *Parity games are determined.*

The game is played between Abelard and Eloise. We start in the position (s, φ) and all other positions have a similar form: (x, ψ), for $x \in S$ and $\psi \in Sub(\varphi)$. Moves are the way to move from one position to the next. When depicted as a graph, the positions are the nodes and the moves the edges. We have the following edges in the game graph of φ.

- For each $\psi_1 \wedge \psi_2, \psi_1 \vee \psi_2 \in Sub(\varphi)$, $x \in S$, $i \in \{1, 2\}$, we have the edges:

$$(x, \psi_1 \wedge \psi_2) \to (x, \psi_i), \qquad (x, \psi_1 \vee \psi_2) \to (x, \psi_i).$$

- For each $\Box\psi, \Diamond\psi \in Sub(\varphi)$, $x \in S$, and each state $y \in S$ such that $x \to y$, we have the edges:

$$(x, \Box\psi) \to (y, \psi), \qquad (x, \Diamond\psi) \to (y, \psi).$$

- For each $\sigma Z.\psi \in Sub(\varphi)$, where $\sigma \in \{\mu, \nu\}$, $x \in S$, we have the edges:

$$(x, \sigma Z.\psi) \to (x, Z), \qquad (x, Z) \to (x, \psi).$$

There will be no outgoing edges from (t, ψ) when ψ is of the form p or $\neg p$, or a free variable Z, meaning that there is no fixed point variable σZ in whose scope Z occurs. In practice, we are considering closed formulas, in which the last option does not appear.

Following the idea that Eloise tries to verify the formula, and Abelard to refute, Eloise can choose which outgoing edge to take in states of the form $(x, \psi_1 \vee \psi_2)$ and $(x, \Diamond\psi)$. Abelard chooses when we are in a node of the form $(x, \psi_1 \wedge \psi_2)$ or $(x, \Box\psi)$.

Definition 2.20. A *play* of $\mathcal{G}_V^{\mathcal{S}}(s, \varphi)$ is a path in the game graph starting in (s, φ) and following the edges.

We say there is a loop in the play when we encounter a node with the same subformula as we have seen before. A play ends when we get to a node with no outgoing edges.

Before we can define when a player wins a game, we need the notion of fixed points that subsume other fixed points.

Definition 2.21. Given two subformulas $\sigma_1 X_1.\psi_1$ and σ_2, X_2, ψ_2 of φ, we say that X_1 *subsumes* X_2 when $\sigma_2, X_2, \psi_2 \in Sub(\sigma_1 X_1.\psi_1)$.

Eloise wins the game when

- the play is finite and we end in a state of the form
  - (x, p) and $x \in \rho(p)$, or
  - $(x, \neg p)$ and $x \notin \rho(p)$, or
  - (x, Z) and $x \in V(Z)$, or
- the play is infinite and the unique fixed point variable Z, which occurs infinitely often and subsumes all other variables occurring infinitely often, is identified with a least fixed point subformula.

Abelard wins when Eloise does not win.

2.4.3 Equivalence with alternating tree automata

After this short introduction on the content of modal μ-calculus, this part focuses on the translations between this logic and alternating tree automata. First we have a theorem that constructs a modal μ-calculus formula from a given parity automaton over infinite trees.

Theorem 2.22 ([12])**.** *For any alternating automaton $\mathcal{A}$ over infinite trees we can construct a modal μ-calculus formula $\varphi_{\mathcal{A}}$ such that for all trees $\mathcal{S}$ with root s_0, we have*

$$\mathcal{A} \text{ accepts } (\mathcal{S}, s_0) \quad \text{if and only if} \quad s_0 \in ||\varphi_{\mathcal{A}}||_{\mathcal{S}}.$$

In chapter 11 of [13] they give a nice presentation of this construction.

The other direction is way less complicated and follows by a quite straightforward induction on the construction of φ. This way, we realise that the structure of the of the automaton $\mathcal{A}_\varphi$ is very similar to the structure of the game graph of the formula.

Theorem 2.23. *For any modal μ-calculus formula $\varphi_\mathcal{A}$ we can construct an alternating automaton $\mathcal{A}$ over infinite trees such that for all trees $\mathcal{S}$ with root s_0, we have*

$$s_0 \in ||\varphi||_\mathcal{S} \quad \textit{if and only if} \quad \mathcal{A}_\varphi \textit{ accepts } (\mathcal{S}, s_0).$$

We conclude this section with the following corollary.

Corollary 2.24. *Modal μ-calculus is exactly as expressive as alternating automata over infinite trees.*

3 Cost automata

The main difference between automata commonly used and cost automata is in what they recognise. Automata are generally designed to recognise word or tree languages: only if a word/tree is accepted by the automaton, we let it be part of our language. In cost automata the acceptance condition is not the primary focus in the sense that we do not mind at all whether a structure has an accepted run or not. Instead, we assign to every input structure a number: the cost.

This section consists of an introduction to cost automata in general. Here is clarified what exactly is added to an automaton to make it a cost automaton. After that, the precise definitions are given, first for finite words and trees, and then for infinite trees. Also attention is given to some duality and decidability theorems.

3.1 Cost automata on finite words

Recall that we want to assign a cost to every input structure. We can achieve this by adding some counter features to the automata. The counting works as follows: in the definition of our automaton, we add the notion of counters, some pieces of easily adaptable short-term memory. Moreover, we define counter actions. These are actions that can make changes to or read the current value of a counter. More precisely, we have a set of counters $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ for some natural number n, and a set of counter actions: in the case of cost automata, we can either increase, i, the value of a particular counter; we can reset, r, it to 0; we can read or check, c, the value, which means storing it in some long-term memory; or we do nothing, which is denoted by ϵ. Together, these actions form a set of counter actions $\mathbb{C} = \{\epsilon, i, c, r\}$.

These counter actions are added to the arrows in our automaton, which gives us the possibility to influence the value of a particular counter after we reach a particular state in our automaton. In this way, we can very precisely count how often particular patterns are occurring. Notice that we can have multiple counters, and thus for every arrow in our automaton a set of counter actions. The different counters can be used to count different things in the same structure.

The precise definition of a cost automaton on finite words is the following.

Definition 3.1. A *cost automaton* is a tuple $(Q, \mathbb{A}, In, Fin, \Gamma, \Delta)$, such that Q is a finite set of states, $\mathbb{A}$ the alphabet, In are the initial states, Fin the final ones, Γ is a set of counters and $\Delta \subseteq Q \times \mathbb{A} \times (\{\epsilon, i, r, c\}^*)^\Gamma \times Q$ the transition relation.

The definition of a run of such an automaton is as expected.

Definition 3.2. A *run* ρ of a cost automaton on an input word $u = a_1 \cdots a_n \in \mathbb{A}^*$ is a sequence of transitions $\rho = (q_0, a_1, c_1, q_1) \cdots (q_{n-1}, a_n, c_n, q_n) \in \Delta^*$.

For every run we can glue all counter actions together, to create a word $u \in (\mathbb{C}^\Gamma)^*$. This is called the output of a run, which is defined as follows.

Definition 3.3. The *output* $out(\rho)$ of a run ρ is the sequence of counter actions $c_1 \cdots c_n \in (\mathbb{C}^\Gamma)^*$.

Then the long-term memory is in fact a set $C(u)$ which consists of all checked values in that word. So, if $u = iicririiicr$, $C(u) = \{2, 3\}$.

Definition 3.4. Given a word $u \in (\mathbb{C}^\Gamma)^*$, describing the actions for all counters, we define $C(u) \subseteq \mathbb{N}$ as the set that collects all checked values for all counters. More precisely, $C(u) := \cup_{\gamma \in \Gamma} C(pr_\gamma(u))$, where pr_γ is the γ-projection of u.

A cost automaton can either be a B- or S-automaton, depending on which semantics you choose. This decides which cost function is recognised by the automaton $\mathcal{A}$. For all $u \in \mathbb{A}^*$, the B- and S-semantics are defined as

$$[\![\mathcal{A}]\!]_B(u) = \inf\{\sup C(out(\rho)) : \rho \text{ is an accepting run over } u\}$$
$$[\![\mathcal{A}]\!]_S(u) = \sup\{\inf C(out(\rho)) : \rho \text{ is an accepting run over } u\},$$

where $\inf(\emptyset) = \omega$ and $\sup(\emptyset) = 0$.

3.1.1 Examples

The following examples also appear in [6] and [20].

Example 3.5. A B-automaton recognising the function $|\cdot|_a$, the function that counts how often the letter a occurs in a word, is

$$\mathcal{A} = (\{q\}, \{a, b\}, \{q\}, \{q\}, \{\gamma\}, \Delta),$$

where $\Delta = \{(q, a, ic, q), (q, b, \epsilon, q)\}$. Notice that there is only one counter, as we will see in most examples. The only exception is example 3.9.

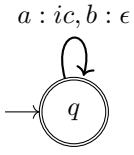

Figure 3.1: B-automaton recognising $|\cdot|_a$.

This automaton works quite straightforward: every time we read an a, we increase the counter and we remember the new value by checking it. There is only one possible run for each word and it is accepting, which means that the value of a word is the supremum of all checked values. Since we exactly increase and check after we read an a, this automaton indeed recognises the function $|\cdot|_a$.

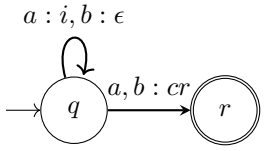

Figure 3.2: S-automaton recognising $|\cdot|_a$.

Example 3.6. An S-automaton recognising the same function $|\cdot|_a$, is

$$\mathcal{A} = (\{q, r\}, \{a, b\}, \{q\}, \{r\}, \{\gamma\}, \Delta),$$

where $\Delta = \{(q, a, i, q), (q, b, \epsilon, q), (q, a, cr, r), (q, b, cr, r)\}$.

This automaton is not deterministic and thus there are multiple possible runs. However, only one of them is accepting: the one that guesses correctly when the end of a word is approaching. In fact, what the automaton is doing on an accepting run, is increasing the counter for every a, and then checking it only at the end of the word. This way, we make sure that the counter is increased for almost all a's appearing in the word. It is of course possible that the last letter of a word is an a, which means that we do not see an 'increase' for the last a. Thus, this automaton indeed recognises the cost function $|\cdot|_a$, but there is a correction function $\alpha(n) = n + 1$ involved.

Example 3.7. A B-automaton recognising the function $minblock_a$, the function that counts the minimum block of a's that appears inbetween two b's, such that when there is at most one b in the word, the cost is infinite, is

$$\mathcal{A} = (\{q, r, s\}, \{a, b\}, \{q\}, \{s\}, \{\gamma\}, \Delta),$$

where

$$\Delta = \{(q, a, \epsilon, q), (q, b, \epsilon, q), (q, b, \epsilon, r), (r, a, ic, r), (r, b, \epsilon, s), (s, a, \epsilon, s), (s, b, \epsilon, s)\}.$$

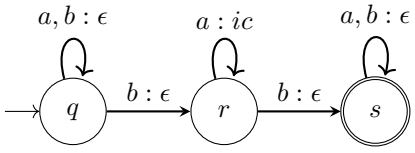

Figure 3.3: B-automaton recognising $minblock_a$.

This automaton gives the following cost on an input word u: $[\![\mathcal{A}]\!]_B(u) = \inf\{\sup C(\sigma) : \sigma \text{ run over } u\}$. Again, since this is a nondeterministic automaton, we find that there are multiple possible runs, and in this case multiple can be accepting. Each single one of these accepting runs is counting the length of a block of a's surrounded by b's. It is a B-automaton, meaning that we only

consider the highest checked value in a run, and then look at the lowest value over all runs. Thus, what happens is that the resulting value is indeed the length of the shortest block of a's surrounded by b's.

When there is at most one b in a word, it is not possible to have an accepting run. The result is that we take the infimum of the empty set. As we saw, the cost is then defined to be infinite, which is exactly what we required. We can conclude that this automaton indeed recognises the function $minblock_a$.

Example 3.8. An S-automaton recognising the same function $minblock_a$ is

$$\mathcal{A} = (\{q, r\}, \{a, b\}, \{q\}, \{q, r\}, \{\gamma\}, \Delta),$$

where $\Delta = \{(q, a, \epsilon, q), (q, b, \epsilon, r), (r, a, i, r), (r, b, cr, r)\}$.

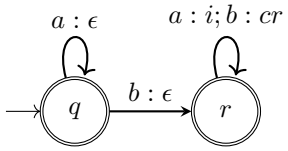

Figure 3.4: S-automaton recognising $minblock_a$.

Because of the S-semantics, the cost on a given input is now $[\![\mathcal{A}]\!]_S(u) = \sup\{\inf C(\sigma) : \sigma \text{ run over } u\}$. It is a deterministic automaton with only one possible run per input word, which means that in practise we just take the infimum of all checked values. Since we exactly check and reset after we have seen a series of a's that all increase the counter, we indeed find the minimum length of a's in between two b's.

When there is at most one b in the word, we never check a counter. This means that we again take the infimum of the empty set, which is infinite, as required.

Example 3.9. In this example we look at a cost automaton with two counters. It is an S-automaton recognising the function

$$f(u) = \min\{minblock_a(u), |u|_b\}.$$

The cost automaton that does the job is

$$\mathcal{A} = (\{q, r, s\}, \{a, b\}, \{q\}, \{s\}, \{\gamma_0, \gamma_1\}, \Delta),$$

where

$$\Delta = \{(q, a, (\epsilon, \epsilon), q), (q, b, (\epsilon, i), r), (r, a, (i, \epsilon), r), \\ (r, b, (cr, i), r), (r, a, (r, cr), s), (r, b, (cr, cr), s)\}.$$

Notice that states q and r with only the first counter are in fact a copy of the automaton in example 3.8. The second counter is used for counting the amount

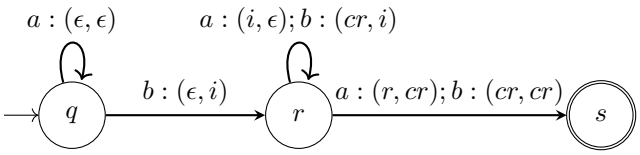

Figure 3.5: S-automaton recognising g.

of b's in the word, which is also just copying what happens in the automaton in example 3.6: increasing for every b and guessing when the word ends.

If there is an accepting run, there is exactly one accepting run. Thus, the cost is the infimum of all checked values of both counters on one pile. So, the result is what we want it to be.

3.1.2 Duality and boundedness

We have seen two different semantics that can be applied to a cost automaton. Although they seem to differ, there is no difference in expressability: if a cost function is recognised by an automaton, it can be recognised by both semantics.

Theorem 3.10 (Duality theorem for finite words, [8])**.** *For every cost function f it is effectively equivalent to be recognisable by a nondeterministic B-automaton over finite words and by a nondeterministic S-automaton over finite words.*

Every cost function that can be recognised by these automata is called a regular cost function over finite words. For these regular cost functions, the boundedness relation can be decided.

Theorem 3.11 (Decidability theorem for finite words, [8], following [3])**.** *Given two regular cost functions f and g on finite words, it is decidable whether or not $f \preceq g$.*

3.1.3 Extension to finite trees

When we want to extend cost automata over finite words to *nondeterministic cost automata over finite trees*, what happens is that we send a copy of the automaton in every direction. The transition function describes in which state both of the copies will be and which counter actions are associated. In the B-version the value of a run is the maximum value across all branches of the run, and the value of the input tree is the minimum value over all accepting runs. For the S-semantics, replace 'minimum' with 'maximum' and the other way around.

We can also extend to *alternating cost automata over finite trees*. This means that the restriction of sending exactly one copy of the automaton in each direction is dropped. However, this does not increase the expressive power. Next to this, we again have the equivalence in expressability of both semantics.

Theorem 3.12 (Duality theorem for finite trees, [9])**.** *For every cost function f it is effectively equivalent to be recognisable by*

- *a nondeterministic B-automaton over finite trees,*
- *a nondeterministic S-automaton over finite trees,*
- *an alternating B-automaton over finite trees, and*
- *an alternating S-automaton over finite trees.*

Again the definition of a regular cost function is justified by such a duality theorem.

Definition 3.13. A cost function over finite trees is a *regular cost function over finite trees*, if there exists a cost automaton over finite trees that recognises that function.

Just like in the case of finite words, we have a decidability theorem.

Theorem 3.14 (Decidability theorem for finite trees, [9])**.** *Given two regular cost functions f and g on finite trees, it is decidable whether or not $f \preceq g$.*

This brings us to the following, since we can check $f \preceq 0$. Clearly we can find similar results for the other decidability results.

Corollary 3.15 (Universality problem for finite trees, [10])**.** *Given a regular cost function, it is decidable whether this function is bounded on the whole domain.*

3.2 Cost automata and games

We have seen nice results for cost automata over finite structures: it does not matter whether you want to express cost functions with a B-, or a S-semantics, because we have the duality theorems. And if two cost functions can be expressed by cost automata, meaning that they are regular cost functions, we know that we can compare them; we can decide the $\preceq$-relation.

In the rest of this section, we continue the story with cost automata over infinite structures and find that some of the nice results stated above still hold, but there remain a number of open problems. To get there, we first give some background information on objectives and games for cost automata, to define their semantics. After that, we continue with the precise definitions of different kinds of cost automata over infinite structures.

3.2.1 Objectives

As we will see shortly, cost games are also different from standard games: players do not win or loose, but aim for the best result possible. You can see an objective as an overview of the scoring rules for the players of a cost game. An objective does not contain information about the exact actions that can be taken by a

player at a certain moment, but of how to interpret the result of actions. More precisely, it contains the set of all actions $\mathbb{C}$ that can happen in a cost game, in the case of cost automata these are the familiar $\{\epsilon, i, c, r\}^*$, a function f that translates a word consisting of actions into a number, and a goal for Eloise. This goal can be minimising or maximising, meaning that she tries to, respectively, minimise or maximise the outcome of f.

These definitions on objectives are originally by Colcombet and Löding, and can be found in [9].

Definition 3.16. An *objective* O is a tuple $(\mathbb{C}, f, goal)$, where $\mathbb{C}$ is a finite alphabet of actions, $f : \mathbb{C}^\omega \to \mathbb{N}_\infty$ a valuation that maps a series of actions to a natural number or infinity, and $goal \in \{min, max\}$, which describes whether we want to minimise or maximise f.

Given an objective O, the dual objective $\overline{O}$ is obtained by switching the goal from min to max or vice versa.

Example 3.17. An example of such an objective is the parity objective:

$$Cost^P_{parity} := (P, cost^P_{parity}, min).$$

Here $P \subseteq \mathbb{N}$ is a finite set of priorities and $cost^P_{parity} : P^\omega \to \{0, \infty\}$ is a map such that

$$cost^P_{parity}(u) := \begin{cases} 0 & \text{if maximum infinitely-occurring priority in } u \text{ is even,} \\ \infty & \text{otherwise.} \end{cases}$$

For instance, we find $cost^P_{parity}((12)^\omega) = 0$ and $cost^P_{parity}((12)^{10}1^\omega) = \infty$.

Notice that it is natural to take min as the goal of this objective; in parity games, a word is accepted when the maximum infinitely-occurring priority is even, which corresponds to minimising the function $cost^P_{parity}$.

In the above example, the objective is rather simple; the only values that can occur are 0 and ∞ and there is no option to do something with the counter actions that were defined before. The following objectives also reserve a place for checked values.

Definition 3.18. The *B-parity objective* is

$$Cost^{\Gamma,P}_B := (\mathbb{B}^\Gamma \times P, cost^{\Gamma,P}_B, min),$$

where

- $\mathbb{B} = \{\epsilon, ic, r\}$,
- $cost^{\Gamma,P}_B(u) := \sup(C(u) \cup \{cost^P_{parity}(u)\})$, and
- $C(u)$ is the set of checked values of $u \in (\mathbb{C}^\omega)^\Gamma$.

When for instance $u = ((\epsilon, 2), (ic, 1), (r, 2), (ic, 1))^\omega$, we find that the function in the objective gives us $cost_B^{\{1\},[1,2]}(u) = \sup(\{1, 2\} \cup \{0\}) = 2$.

For the next definition, in certain sense the dual of the B-parity objective, we need the dual of $cost^P_{parity}$:

$$\overline{cost^P_{parity}}(u) := \begin{cases} \infty & \text{if maximum infinitely-occurring priority in } u \text{ is even,} \\ 0 & \text{otherwise.} \end{cases}$$

Definition 3.19. The *S-parity objective* is

$$Cost_S^{\Gamma,P} := (\mathbb{S}^\Gamma \times P, cost_S^{\Gamma,P}, max),$$

where

- $\mathbb{S} = \{\epsilon, i, r, cr\}$,
- $cost_S^{\Gamma,P}(u) := \inf(C(u) \cup \{\overline{cost^P_{parity}}(u)\})$, and
- $C(u)$ is the set of checked values of $u \in (\mathbb{C}^\omega)^\Gamma$.

As the name of these objectives already suggest, we will use them to define B- and S-parity automata over infinite trees.

3.2.2 Cost games

The players in a game, Eloise and Abelard, do not win or lose. Instead, they try to minimise or maximise the value assigned to the game, based on the objective. Thus, objectives take the place where originally we would find winning conditions. The goal that can be found in O is the aim of the first player, Eloise. Abelard has the opposite goal.

This part is based on [21] and [20] by Vanden Boom.

Definition 3.20. A cost game $\mathcal{G}$ is a tuple (V, v_0, δ, O) consisting of the positions V in the play, the starting position v_0, a control function $\delta : V \to \mathcal{B}^+(\mathbb{C} \times V)$ and an objective $O = (\mathbb{C}, f, goal)$.

The next step in the game graph is determined as follows. Suppose we are in a particular position v. Then $\delta(v)$ gives us a disjunction of conjunctions of possible follow-up states, combined with a counter action. Eloise chooses one of the disjuncts, after which Abelard can pick one of the conjuncts in that disjunct. This gives the next position v', from where the game continues.

The *dual* of a cost game $\mathcal{G}$ is the game where the role and goal of the players have reversed: it is the tuple $\overline{\mathcal{G}} = (V, v_0, \overline{O}, \overline{\delta})$, where $\overline{O}$ switches the goal of the players, and $\overline{\delta}$ is obtained from δ by switching conjunctions and disjunctions, and then rewriting in disjunctive normal form. This indeed changes the role of the players, but not the resulting value, as we will see shortly.

The set of *moves* in the game $\mathcal{G}$ is defined as

$$E_\mathcal{G} := \{(v, c, v') \in V \times \mathbb{C} \times V : (c, v') \text{ appears in } \delta(v)\}.$$

A *play* π consists of an infinite series of moves, such that the first move starts in the starting node v_0. The output $out(\pi)$ of a play π is the concatenation of all counter actions that appear in the moves in π.

If σ is a set of plays, define $pref(\sigma)$ as the set of *prefixes* of plays in σ. A prefix can be extended by adding one move, such that the last state of the last move of the prefix corresponds to the first state of the move you add.

The idea of a *strategy* σ is that, given the history of the play, it is determined what the player will do. This means that a *strategy* σ for Eloise in $\mathcal{G}$ consists of a set of plays such that for every partial play $\pi \in pref(\sigma)$ ending in state v, there exists a unique disjunct in $\delta(v)$ such that π extended with any move that can follow from this disjunct is contained in $pref(\sigma)$. A strategy σ for Abelard is a strategy for Eloise in the dual game $\overline{\mathcal{G}}$.

Definition 3.21 (Value of a play/strategy/game)**.** The value of a play, strategy and game is defined as follows.

- Given an objective $O = (\mathbb{C}, f, goal)$, the value of a play π is
$$value(\pi) := f(out(\pi)).$$
- If $goal = min$, we define the value of a strategy σ for Eloise as
$$value(\sigma) := \sup\{value(\pi) : \pi \in \sigma\},$$
and the value of a game $\mathcal{G}$ as
$$value(\mathcal{G}) = \inf\{value(\sigma) : \sigma \text{ strategy for Eve in } \mathcal{G}\}.$$
- If $goal = max$, we define the value of a strategy σ for Eloise as
$$value(\sigma) := \inf\{value(\pi) : \pi \in \sigma\},$$
and the value of a game $\mathcal{G}$ as
$$value(\mathcal{G}) = \sup\{value(\sigma) : \sigma \text{ strategy for Eloise in } \mathcal{G}\}.$$

As announced, the value of a cost game is equal to the value of its dual.

Proposition 3.22 ([20])**.** *For all cost games $\mathcal{G}$ we have $value(\mathcal{G}) = value(\overline{\mathcal{G}})$.*

3.3 Extension to infinite trees

With this background, we can start defining cost automata over infinite trees. The definitions are given for infinite binary trees, for the notational simplicity, but can be extended to labelled trees over some ranked alphabet $\mathbb{A}$ where every symbol has an arity determining the finite branching at that position. In infinite binary trees, every letter in the alphabet has rank 2.

Cost automata over infinite words are easy to understand once you understand cost automata over infinite trees: the only difference is that the choice of which child of a node to continue with does not have to be made. Therefore, those definitions are left out, to have more focus on the trees.

Definition 3.23. An *(alternating) cost automaton over infinite trees* is a tuple $\mathcal{A} = (Q, \mathbb{A}, q_0, O, \delta)$, where Q is the finite set of states, $\mathbb{A}$ is the alphabet used for labelling the infinite tree, q_0 is the initial state, O is an objective and $\delta : Q \times \mathbb{A} \to \mathcal{B}^+(\{0,1\} \times \mathbb{C} \times Q)$ the transition/control function.

This means that δ is a function that is dependent on the state the automaton is in and which label we read: for instance $\delta(q, a)$. The result must be a positive Boolean combination of elements like (i, c, q), for $i \in \{0, 1\}$, $c \in C$ and $q \in Q$.

Given a tree $t \in \mathcal{T}_{\mathbb{A}}$, we can express how our automaton works on this tree t in terms of the cost game

$$\mathcal{A} \times t := (Q \times \mathcal{T}, (q_0, \epsilon), O, \delta').$$

Here $\mathcal{T} = \{0, 1\}^*$ is the unlabelled binary tree and

$$\delta'((p, x)) = \delta(p, t(x))[(c, (q, xk))/(k, c, q)].$$

Here, δ' is a natural translation that ensures that the game is well-defined. We can conclude that every point of the game corresponds to a state of the automaton in combination with a position of the input tree.

Finally, we set

$$[\![\mathcal{A}]\!](t) := value(\mathcal{A} \times t),$$

which means that when the goal for Eloise is min,

$$[\![\mathcal{A}]\!](t) = \inf\{\sup\{value(\pi) : \pi \in \sigma\} : \sigma \text{ strategy for Eloise in } \mathcal{A} \times t\}.$$

In the case her goal is max, we find

$$[\![\mathcal{A}]\!](t) = \sup\{\inf\{value(\pi) : \pi \in \sigma\} : \sigma \text{ strategy for Eloise in } \mathcal{A} \times t\}.$$

A B-parity automaton is an alternating cost automaton with the B-parity objective, and similarly for S-parity automata. The main focus in this thesis will be on alternating B- and S-parity automata. We will also refer to them as B- and S-automata.

In fact, there is not even a difference in expressibility between B-parity and S-parity automata, as the next theorem states. The other possible objectives we do not take into account here.

Theorem 3.24 (Duality theorem for infinite trees [20])**.** *For any cost function f over infinite trees, it is effectively equivalent to be recognisable by an alternating B-parity automaton and by an alternating S-parity automaton.*

This again means that the following definition is well-defined.

Definition 3.25. A *regular cost function over infinite trees* is a cost function that is recognised by an alternating cost automaton over infinite trees.

Decidability of the boundedness relation $\preceq$ is not known for all regular cost functions over infinite trees, but only for a subset of cost functions that are recognisable by a nondeterministic cost automaton.

Alternating cost automata over infinite trees are a generalisation of the nondeterministic version in the sense that nondeterministic cost automata send exactly one copy of the automaton in every direction, while the alternating version can send more than one copy, or none at all. Concretely, this means that nondeterministic cost automata have the same definition as alternating cost automata with the restriction that each $\delta(q, a)$ is a disjunction of clauses of the form $(0, c, q) \wedge (1, c', q')$, for $c, c' \in \mathbb{C}$ and $q, q' \in Q$.

Theorem 3.26 ([20])**.** *The relation $f \preceq g$ is decidable for cost functions over infinite trees if f is given by a nondeterministic S-parity automaton and g is given by a nondeterministic B-parity automaton.*

To make the overview complete, we have stated the results for infinite words separately.

3.3.1 Results for infinite words

For cost functions over infinite words, we have the following theorem and definition. The Büchi objective is a parity objective with the set $[0, 1]$ as possible priorities.

Theorem 3.27 (Duality theorem for infinite words, can be found in [20], based on personal communication between Vanden Boom and Colcombet in 2012)**.** *For every cost function f over infinite words it is effectively equivalent to be recognisable by*

- *a nondeterministic B-parity automaton,*
- *a nondeterministic S-parity automaton,*
- *a nondeterministic B-Büchi automaton, and*
- *a nondeterministic S-Büchi automaton.*

Definition 3.28. A *regular cost function over infinite words* is a cost function that is recognised by a nondeterministic cost automaton over infinite words.

Since the set of regular cost functions over infinite words only consists of cost functions recognised by nondeterministic automata, we do find that $\preceq$ is decidable for the whole class of regular cost functions over infinite words, as a direct result of theorem 3.26.

Corollary 3.29 (Decidability theorem for infinite words)**.** *Given two regular cost functions f and g on infinite words, it is decidable whether $f \preceq g$.*

3.3.2 Examples

To make things more concrete, the rest of this section is devoted to some examples of automata over infinite trees.

In all these examples, we are considering infinite binary trees. The last two examples are partly taken from [20].

Example 3.30. We can recognise the function $f(t) = \inf\{|\pi|_a : \pi \text{ branch in } t\}$, which counts the minimum number of a's appearing on a branch in t, with the following B-automaton:

$$\mathcal{A} = (\{q_0\}, \{a, b\}, q_0, Cost_B^{\{1\},[1,2]}, \delta).$$

Here δ is defined as follows,

$$\begin{aligned}\delta(q_0, a) &:= (0, (ic, 2), q_0) \vee (1, (ic, 2), q_0)\\ \delta(q_0, b) &:= (0, (\epsilon, 2), q_0) \vee (1, (\epsilon, 2), q_0).\end{aligned}$$

So, when we read an a on our path through the tree, we increase and check the counter. If no 'c' is encountered, we do nothing.

Recall that

$$Cost_B^{\{1\},[1,2]} = (\mathbb{B}^1 \times \{1, 2\}, cost_B^{\{1\},[1,2]}, min)$$

and

$$cost_B^{\{1\},[1,2]}(u) = \sup(C(u) \cup \{cost_{parity}^P(u)\}).$$

Since the only priority that is assigned is 2, we know the maximum infinitely-occurring priority is even, which means that $cost_{parity}^P(u) = 0$ for every u. Thus, Eloise tries to minimise the function $\sup(C(u) \cup \{0\})$. Since she is in full control of which path to take through the tree, Eloise can pick the branch with the least number of a's. We can conclude that this automaton indeed gives us a function that maps a tree to the minimum of a-labelled positions over all branches.

Example 3.31. The function $g(t) = \sup\{|\pi|_a : \pi \text{ branch in } t\}$ is also recognisable. We can, for instance, adapt the previous example and let Abelard choose the path instead of Eloise. That is, change the $\vee$'s to $\wedge$'s in δ. This gives a B-automaton recognising $g(t)$.

The following S-automaton also recognises $g(t)$:

$$\mathcal{A} = (\{q_0, q_T\}, \{a, b\}, q_0, Cost_S^{\{1\},[1,2]}, \delta),$$

where

$$\begin{aligned}\delta(q_0, a) &:= (0, (i, 1), q_0) \vee (1, (i, 1), q_0) \vee (1, (cr, 2), q_T)\\ \delta(q_0, b) &:= (0, (\epsilon, 1), q_0) \vee (1, (\epsilon, 1), q_0) \vee (1, (cr, 2), q_T)\\ \delta(q_T, *) &:= (1, (\epsilon, 2), q_T).\end{aligned}$$

In this example Eloise has full control of where to go. The idea is that she keeps looking for a's, until she has found enough. At that point she goes to state q_T, which means the counter is checked. When in state q_T, the players cannot leave the state and nothing happens anymore.

On the tree t with only a's as label, this works as follows. Since $g(t) = \infty$, we also expect that $[\![\mathcal{A}]\!](t) = \infty$. If Eloise keeps looking for more a's, and thus never enters q_T, the value of that play will be 0, since the highest priority seen infinitely often is 1. So, if Eloise wants to maximise, at a certain point she has to switch to state q_T. This means that for every strategy she has, the value will be finite. Nevertheless, the value of the play will be infinite because it is defined as

$$[\![\mathcal{A}]\!](t) = \sup\{\inf\{value(\pi) : \pi \in \sigma\} : \sigma \text{ strategy for Eloise in } \mathcal{A} \times t\}.$$

Since Eloise can for every natural number n find a strategy π such that

$$\inf\{value(\pi) : \pi \in \sigma\} \geq n,$$

we find that indeed $[\![\mathcal{A}]\!](t) = \infty$.

The previous examples view the trees as the sum of all branches, but we can do more. For instance, we can count the total amount of a's appearing in the whole tree. This is worked out in the next example.

Example 3.32. The function $h(t) = |t|_a$, counting the number of a's in a labelled tree t, is recognisable by a cost automaton. There is no automaton that exactly computes this function, but there exists an $\mathcal{A}$ such that for $\alpha(n) = 2^n$, we find $[\![\mathcal{A}]\!] \approx_\alpha h$.

The idea is that Eloise guesses which subtrees have an a. In the beginning, every transition has priority 2, and it will stay that way, until Abelard proofs that one of Eloise's guesses was wrong. If the current position has an a, or Eloise guesses that both subtrees of the current position have an a, the counter is increased and checked. If she guesses that only one or none of the subtrees has an a, nothing happens to the counter. Abelard can always choose which way to go and can try to find as many a's as possible. If Abelard finds an a in a subtree Eloise labelled as without a's, we end up in a state where nothing happens anymore with priority 1, which results in an infinite cost.

This function can be recognised by the following B-automaton $\mathcal{A}$:

$$\mathcal{A} = (\{q, r, s\}, \{a, b\}, q, Cost_B^{\{1\},[1,2]}, \delta),$$

where

$$\begin{aligned}
\delta(q,a) =& (0,(ic,2),q) \wedge (1,(ic,2),q) \\
\delta(q,b) =& (0,(ic,2),q) \wedge (1,(ic,2),q) \vee \\
& (0,(\epsilon,2),r) \wedge (1,(\epsilon,2),r) \vee \\
& (0,(\epsilon,2),q) \wedge (1,(\epsilon,2),r) \vee \\
& (0,(\epsilon,2),r) \wedge (1,(\epsilon,2),q) \\
\delta(r,a) =& (0,(\epsilon,1),s) \wedge (1,(\epsilon,1),s) \\
\delta(r,b) =& (0,(\epsilon,2),r) \wedge (1,(\epsilon,2),r) \\
\delta(s,*) =& (0,(\epsilon,1),s) \wedge (1,(\epsilon,1),s).
\end{aligned}$$

4 Cost modal μ-calculus

In this chapter, we will first define the syntax and semantics for a somewhat quantitative extension of modal μ-calculus which we call cost modal μ-calculus. Afterwards, we will give translations of cost automata to this logic and vice versa and proof the correctness of these translations. With this, we can conclude that cost modal μ-calculus is exactly as expressive as cost automata over infinite trees, which is the main result of this chapter. We continue with proving directly that the B- and S-semantics yield same expressability. Then we talk about a normal form for this logic. We conclude with describing a fragment of cost modal μ-calculus that does not appeal games for its semantics, hence providing a different perspective.

4.1 Syntax and semantics

We begin with the syntax of cost formulas, which is:

$$\varphi ::= p \mid \neg p \mid Z \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \Box\varphi \mid \Diamond\varphi \mid \mu Z.\varphi(Z) \mid \nu Z.\varphi(Z) \mid c\varphi,$$

where p is a proposition, Z a variable and $c \in \mathbb{C}$, which is a set of counter actions. Note that the syntax is that of standard modal μ-calculus extended with counter actions that can be added to the formulas.

Remark 4.1. In this logic we assume that Γ consists of exactly one counter. However, all definitions and theorems can be extended to the case with multiple counters, but this would make the notation unnecessarily complicated.

The semantics of this logic will be defined via game semantics. Therefore, we need to first redefine the notion of a subformula, and then define the game graph.

Definition 4.2. The set of *subformulas* of a cost formula φ, denoted by $Sub(\varphi)$, is defined inductively as

- if $\varphi = p$ for a proposition p, $Sub(\varphi) = \{p\}$,
- if $\varphi = \neg p$ for a proposition p, $Sub(\varphi) = \{\neg p\}$,
- if $\varphi = Z$ for a variable Z, $Sub(\varphi) = \{Z\}$,
- if $\varphi = \psi_1 * \psi_2$ for $* \in \{\wedge, \vee\}$, $Sub(\varphi) = Sub(\psi_1) \cup Sub(\psi_2) \cup \{\psi_1 * \psi_2\}$,
- if $\varphi = @\psi$ for $@ \in \{\Box, \Diamond\}$, $Sub(\varphi) = Sub(\psi) \cup \{@\psi\}$,
- if $\varphi = \sigma Z.\psi(Z)$ for $\sigma \in \{\mu, \nu\}$, $Sub(\varphi) = Sub(\psi(Z)) \cup \{\sigma Z.\psi(Z)\}$,
- if $\varphi = c\psi$ for $c \in \mathbb{C}$, $Sub(\varphi) = Sub(\psi) \cup \{c\psi\}$.

Fix a structure $\mathcal{S} = (S, \to, \rho)$, a distinguished state $s \in S$ in that structure and a cost fixed point formula φ. The nodes of the game graph $\mathcal{G}^t(\varphi)$ are given by $S \times Sub(\varphi)$. The edges of the game graph are as follows.

- For each $\psi_1 \wedge \psi_2, \psi_1 \vee \psi_2 \in Sub(\varphi)$, $x \in S$, $i \in \{1,2\}$, we have the edges:

$$(x, \psi_1 \wedge \psi_2) \to (x, \psi_i), \qquad (x, \psi_1 \vee \psi_2) \to (x, \psi_i).$$

- For each $\Box\psi, \Diamond\psi \in Sub(\varphi)$, $x \in S$, and each state $y \in S$ such that $x \to y$, we have the edges:

$$(x, \Box\psi) \to (y, \psi), \qquad (x, \Diamond\psi) \to (y, \psi).$$

- For each $\sigma Z.\psi \in Sub(\varphi)$, where $\sigma \in \{\mu, \nu\}$, $x \in S$, we have the edges:

$$(x, \sigma Z.\psi) \to (x, Z), \qquad (x, Z) \to (x, \psi).$$

- For each $c\psi \in Sub(\varphi)$, we have the edge:

$$(x, c\psi) \to (x, \psi).$$

The starting point of the game is the pair (s, φ). Eloise owns all states where the formula is of the form $\psi_1 \vee \psi_2$ or $\Diamond\psi$. Abelard owns the $\psi_1 \wedge \psi_2$ and $\Box\psi$ states. Owning a state means that if we are in that state in the game, the owner can choose where we go next. Clearly then, for all states with just one outgoing arrow, it does not matter who owns them.

Just like in the automata counterpart, players cannot win or lose the game: they can only try to minimise or maximise the outcome. We will define two types of semantics for this logic to mimic the two kinds of semantics for cost automata. In the B-game semantics, Eloise is the minimising player and Abelard the maximising. As can be expected, these roles turn around in the S-game semantics.

Definition 4.3. A *play* π is a sequence of the form

$$(x_0, \varphi_0), (x_1, \varphi_1), \ldots, (x_n, \varphi_n), \ldots,$$

where $(x_0, \varphi_0) = (s, \varphi)$ and for all $i \geq 0$ the edge $(x_i, \varphi_i) \to (x_{i+1}, \varphi_{i+1})$ is in the game graph. The *length* of a finite play, denoted $l(\pi)$, is n if and only if $\pi = (x_0, \varphi_0), \ldots, (x_{n-1}, \varphi_{n-1})$.

Before we can define the value of a game, we first need a function that keeps track of the fixed point alternations of μ and ν operators along a play π: if a μ-subformula is seen infinitely often, we require that there is a ν-subformula, of which the μ-formula is again a subformula, also seen infinitely often. We will also refer to this property as the parity condition, because of its tight connection to that. More precisely,

$$check(\pi) := \begin{cases} 1 & \text{if } \mu X.\psi \in i(\pi) \to (\exists X'.\nu X'.\psi' \in i(\pi) \wedge \mu X.\psi \in Sub(\psi')), \\ 0 & \text{otherwise,} \end{cases}$$

where $i(\pi)$ denotes the set of formulas that appear infinitely often in π.

Definition 4.4. The value of a play in the B-semantics is defined as

$$value_B(\pi) := \sup(C(u) \cup \{f(\pi)\}),$$

where, for all $n \in \mathbb{N}$,

$$f(\pi) := \begin{cases} 0 & \text{if } l(\pi) = \infty \wedge check(\pi) = 1, \\ 0 & \text{if } l(\pi) = n+1 \wedge \exists p.\varphi_n = p \wedge x_n \in \rho(p), \\ \infty & \text{otherwise,} \end{cases}$$

is a correction function, and $C(u)$ is the set of checked values of $u \in (\mathbb{C}^\omega)^\Gamma$.

Definition 4.5. The outcome of a play in the S-semantics is defined as

$$value_S(\pi) := \inf(C(u) \cup \{\overline{f}(\pi)\}),$$

where

$$\overline{f}(\pi) := \begin{cases} 0 & \text{if } f(\pi) = \infty, \\ \infty & \text{if } f(\pi) = 0, \end{cases}$$

is the opposite of the correction function $f(\pi)$ and $C(u)$ is the set of checked values of $u \in (\mathbb{C}^\omega)^\Gamma$.

Analogous to how it was defined in cost games, let the value of a strategy for Eloise in the B-semantics be $value_B(\sigma) := \sup\{value_B(\pi) : \pi \in \sigma\}$. The value of a game $\mathcal{G}^t(\varphi)$ is the value of the best strategy for Eloise:

$$value_B(\mathcal{G}^t(\varphi)) := \inf\{value_B(\sigma) : \sigma \text{ strategy for Eloise}\}.$$

For the S-semantics, let $value_S(\sigma) := \inf\{value_S(\pi) : \pi \in \sigma\}$. Again, the value of a game $\mathcal{G}^t(\varphi)$ is the value of the best strategy for Eloise:

$$value_S(\mathcal{G}^t(\varphi)) := \sup\{value_S(\sigma) : \sigma \text{ strategy for Eloise}\}.$$

A strategy here is a history-deterministic strategy; players know the whole history of moves and thus the value of the counter at any point in the game.

Remark 4.6. When we want to use this logic to say something about how often certain patterns appear in *labelled* infinite binary trees, we can do so by utilising propositions. Recall that we use t to denote a labelling of an infinite binary tree.

First of all, for each $e \in \mathbb{A}$ we say p_e holds when a node is given the label e:

$$\rho(p_e) := \{x : t(x) = e\}.$$

Next to this, we also want to be able to express that the pointed structure $(\mathcal{S}, s)$ is an infinite binary tree. Which means that for every node, except s, which is the root, we want to express whether it is a left or right child. For this, we use the propositions p_0 and p_1, respectively.

Before we will get to the translations between this logic and alternating cost automata, we provide some examples of how formulas in this logic look like.

Example 4.7. Take a look at the B-automaton of example 3.30. A cost formula, based on δ, with a B-semantics that corresponds to this automaton is:

$$\nu Z.((p_a \wedge \Diamond icZ) \vee (p_b \wedge \Diamond Z)).$$

What happens in the formula is that we loop infinitely often through this greatest fixed point. Every time we read an a, we end up in the left disjunct (if the play is played according to optimal strategies) from where we move on to the next point in the tree and adapt our counter to the fact that we have seen an a. When we read a b, we are directed to the right side of the disjunct, which lets Eloise choose whether we go left or right.

Example 4.8. The B-automaton in example 3.31 is quite similar to the B-automaton of example 3.30. This is also mirrored in the corresponding formula:

$$\nu Z.((p_a \wedge \Box icZ) \vee (p_b \wedge \Box Z)).$$

These examples already give a glimpse of what we now prove: alternating cost automata and cost modal μ-calculus coincide in the sense of capturing the same cost functions.

4.2 Cost automata to cost logic

Recall that in cost automata, the priorities appear on the arrows. In the following proof, we want that priorities are connected to states instead. This helps us to determine whether a state should be identified with a least or a greatest fixed point. We achieve this by requiring that $\Omega : Q \to \omega$ is a function sending states to a natural number and δ is of the form $(*, (c, \Omega(q)), q)$, where $* \in \{0, 1\}$ and $c \in \mathbb{C}^*$. For the expressability of automata it does not matter whether priorities appear on arrows or on states.

Next to this, we need the concept of a strongly connected component in this proof. Some set of vertices is a *strongly connected component* if for all vertices in the component it holds that there is a path to any vertex in that component. See the following figure for a small example.

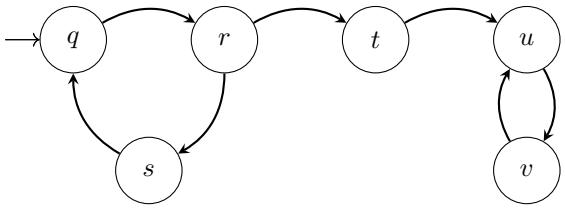

Figure 4.1: Strongly connected components: $\{q, r, s\}$ and $\{u, v\}$.

Recall that the index of a parity automaton is the cardinality of the set of priorities used in that automaton.

The next proof is inspired by the proof of a somewhat similar theorem that constructs a modal μ-calculus formula for normal parity automata: theorem 2.22.

Theorem 4.9. *For every cost automaton $\mathcal{A} = (Q, \mathbb{A}, q_0, Cost_B^{\{1\},P}, \delta)$ over infinite trees, we can construct a cost modal μ-calculus formula $\varphi_\mathcal{A}$ such that*

$$[\![\mathcal{A}]\!](t) \approx value_B(G^t(\varphi_\mathcal{A})).$$

In example 4.10 we apply some of the steps of this proof in an actual case, which can be helpful for understanding this proof.

Proof. In this proof, we only consider cost automata over infinite binary trees for notational simplicity. Thus, we also require t to be an infinite binary tree. Without loss of generality, this proof can be extended to hold for unranked infinite trees.

Assume that the priority function is the lowest priority in the set P when a node is not in a strongly connected component. Next to this, assume that the highest priority, $\max(P)$, does appear as the priority of a node in a strongly connected component. These are no restrictions since every automaton is equivalent to such an automaton. The proof will be on induction on the cardinality of the range of priorities used in strongly connected components.

- For the induction base, meaning that there is only one priority that can be seen infinitely often, we do the following.

  For the translation, let the set of variables be $\{X_q : q \in Q\}$. The idea is that for every state q we create a formula $\varphi_q = \sigma X_q.\psi_q$, by translating the information in $\delta(q, e)$ for all $e \in \mathbb{A}$.

  Since knowing which variables we saw before is useful for knowing when we can refer back to earlier defined fixed points, instead of defining them again, we let all these formulas carry a set of states. The idea is that the current fixed point variable is subsumed by the fixed point variables that correspond to the states in this set. So, every state is assigned a set of formulas: one for each subset of all states. More precisely,

  $$\varphi_{q,Q'} := \begin{cases} V_q & \text{if } q \text{ is moved into a variable,} \\ X_q & \text{if } q \in Q', \\ \sigma X_q.\psi_{q,Q' \cup \{q\}} & \text{otherwise,} \end{cases}$$

  where if $\Omega(q)$ is even, $\sigma = \mu$, and if $\Omega(q)$ is odd, $\sigma = \nu$, and where

  $$\psi_{q,Q'} := (\bigvee_{e \in \mathbb{A}} (p_e \wedge \delta'(q, e))),$$

  where $\delta'(q, e)$ is constructed out of $\delta(q, e)$ by replacing

– $(0, (c, \Omega(q'), q')$ with the formula $\Diamond(p_0 \wedge c\varphi_{q',Q'})$, and
– $(1, (c, \Omega(q'), q')$ with the formula $\Diamond(p_1 \wedge c\varphi_{q',Q'})$.

Note that when we only consider the base case and thus never encounter an induction step there will be no states moved into variables. How the translating works and why we need it, is explained in the induction step.

For the full formula $\varphi_{\mathcal{A}}$, state

$$\varphi_{\mathcal{A}} := \sigma X_{q_0}.\psi_{q_0,\{q_0\}},$$

meaning that we start with the formula for the starting state, while only have seen the fixed point for the starting state.

The idea behind the whole construction is to make sure that the games that are played to determine the value of both sides of this equality, are approximately the same. This is explained in more detail in the remainder of this induction base.

Because the outermost fixed point in $\varphi_{\mathcal{A}}$ is the one corresponding to the starting state, both games start in a corresponding state. Then, on the formula side, Eloise can pick one of the conjuncts of the form $p_e \wedge \delta'(q, e)$. She does not really have a choice here: it only makes sense for her to pick the one that corresponds with the label $e \in \mathbb{A}$ of the node of the tree the game is at the moment. If she picks a different disjunct, Abelard can make the cost infinite. Since Eloise is the minimising player, any strategy that picks the correct label is at least as good as a strategy that picks a wrong label at a certain point.

So, in an optimal strategy for Eloise, we will end up in $p_e \wedge \delta'(q, e)$ in such a way that the label holds. Abelard will in general let the game continue with the right side of this conjunct, because that is the only way to let the cost increase. On the automata side, we will also look at the label and get to $\delta(q, e)$, for the same label e. Thus, the automata game will lead to $\delta(q, e)$ and the formula one to $\delta'(q, e)$.

Note that by construction there is not much difference between the structures of these subgames. So, the only thing we need to look at is whether $(0, (c, \Omega(q'), q')$ corresponds to the result of the choices Eloise and Abelard would make in the subgame generated by $\Diamond(p_0 \wedge c\varphi_{q'})$ (and similarly for the 1 case). First, Eloise can choose whether she wants to go to the left or right child of the current node. Again, there is not really a choice here for Eloise, since it is in fact determined by p_0: if she does not go to the left child, Abelard can make the cost infinite. By the same reason as before, Abelard will in general choose the right conjunct, which means that what happened to the counter in the automata play when following optimal strategies, is exactly the same as in the formula play, also following optimal strategies, of the game. Both plays will end up in a subgame that corresponds to being in state q' in the same child of the node in the tree where we started.

Clearly, we can now follow the same argumentation to conclude that, in optimal strategies, similar choices are made in the games. Thus, what happens to the counter is similar. The correction function $f(\pi)$ guarantees that the cost will be infinite when the highest priority seen infinitely often in the automata part is odd.

Note that we are using induction because we also want to be able to translate automata like the one depicted in figure 4.2. The number after the state denotes the priority of that state.

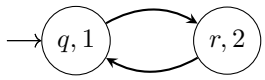

Figure 4.2: Automaton that needs unwinding.

An infinite play in this automaton would satisfy the parity condition, but if we would follow the translation as described in the induction basis, the order of the μ- and ν-fixed point is such that we do not satisfy the parity condition, although both fixed points are seen infinitely often. The automaton needs to get unwinded to find a fixed point formula that does represent this automaton. The induction step takes care of the unwinding.

- Let us focus on the induction step: $|P| = n + 1$. Define $R \subseteq Q$ as the set of states in a strongly connected component that indeed have priority $\max(P)$. This is a nonempty set by assumption, so state $R = \{q_1, \ldots, q_k\}$.

  The idea is that we move these states with the highest priorities into variables, which gives us automata that do not use $\max(P)$. Thus, we can apply the induction hypothesis here. Then, everything is combined to form the formula that is equivalent to $\mathcal{A}$. But first we have to adjust the definition of alternating cost automata over infinite trees slightly, to allow variables. Notice that this is just a technical detail, only here to make the proof technically possible.

  Now let us redefine our automata: an alternating cost automaton over infinite trees is a tuple

  $$\mathcal{A} = (Q, X, \mathbb{A}, q_0, O, \delta),$$

  where X is a set of variables and $\delta : Q \times \mathbb{A} \to \mathcal{B}^+([0,1]^* \times \mathbb{C} \times (Q \cup X))$. Before, plays could not finish, but now they can if we end up in a variable. Note that we can remain to use the same objectives: $cost^P_{parity}(u) = 0$ if $|u|$ is finite, since there is no priority occurring infinitely often, so definitely all highest priorities with this property are even.

  Before we can continue with the rest of the plan, we need to define two kinds of transformations of automata. The first one is to make precise what 'moving states into variables' means. Take an automaton

$\mathcal{A} = (Q, X, \mathbb{A}, q_0, Cost_B^{\{1\},P}, \delta)$ and a set $R \subseteq Q$ such that $q_0 \notin R$. This transformation defines a new automaton

$$\mathcal{A}_{free(R)} = (Q - R, X \cup R, \mathbb{A}, q_0, Cost_B^{\{1\},P'}, \delta'),$$

where δ' is the restriction of δ to $Q - R$ and $P' = P - \{\max(P)\}$. The runs in this new automaton are in a sense the beginning of a run in $\mathcal{A}$; if a run in $\mathcal{A}_{free(X)}$ stops in one of the new variables, then this same run in $\mathcal{A}$ would continue.

The other transformation takes an automaton

$$\mathcal{A} = (Q, X, \mathbb{A}, q_0, Cost_B^{\{1\},P}, \delta),$$

copies an existing node and lets the automaton start in this new node. This helps us to bypass the restriction in the previous transformation. So, take $q \in Q$, a new symbol $\hat{q} \notin Q \cup X$ and define

$$\mathcal{A}_{start(q)} = (Q \cup \{\hat{q}\}, X, \mathbb{A}, q, Cost_B^{\{1\},P}, \delta''),$$

where δ'' is equal to δ on Q, $\delta''(\hat{q}) = \delta(q)$ and $\Omega(\hat{q}) = \Omega(q)$. Remark that $\mathcal{A}_{start(q_0)}$ accepts the same structures as $\mathcal{A}$.

If in our automaton $\mathcal{A}$ we find $q_0 \in R$, we consider in the rest of the proof the semantically equivalent formula $\mathcal{A}_{start(q_0)}$ as $\mathcal{A}$ instead of the original $\mathcal{A}$. Because the result of this transformation is that the new starting state is not contained in a strongly connected component, this transformation works.

We now consider the automaton $\mathcal{A}_{free(R)}$ and for all $i \in \{1, \ldots, k\}$ the automaton $\mathcal{A}_{start(q_i)free(R)}$. Since all nodes with priority $\max(P)$ are changed to variables in all these automata, we can apply the induction hypothesis to find cost modal μ-calculus formulas $\varphi_R, \varphi_1, \ldots, \varphi_k$ such that for all binary labelled infinite trees t we have

$$[\![\mathcal{A}_{free(R)}]\!](t) \approx value(G^t(\varphi_R)) \tag{1}$$

and

$$[\![\mathcal{A}_{start(q_i)free(R)}]\!](t) \approx value(G^t(\varphi_i)) \tag{2}$$

for all $i \in \{1, \ldots, k\}$.

So, remember that we have these φ_i's. Name the outermost fixed point variable in the formula φ_i X_i, as follows: $\varphi_i = \sigma X_i.\psi_i(X_i)$ for all $i \in \{1, \ldots, k\}$, where $\sigma \in \{\mu, \nu\}$. Construct the formula φ as follows: start with the formula φ_R and replace every variable V_{q_i} in there with the formula $\varphi_{i,i}$, which is the same formula as φ_i, but with some extra information, as we will explain. Keep replacing until there is no variable of the form V_q left, as follows: if we want to replace a variable within $\varphi_{i,N}$,

replace V_{q_j} with X_j if $j \in N$, and replace with X_j with $\varphi_{j,N\cup i}$, which is equal to the formula φ_j, but combined with some information of how to interpret the variables in that formula. This process terminates, since there is only a finite amount of different numbers, namely k, that can be considered for the set N.

Remark that the outer part of this formula, φ_R, is corresponding to the first part of a run in $\mathcal{A}$; until we get to the first state with priority $\max(P)$. After that, the formulas $\varphi_1, \ldots, \varphi_k$ correspond to how the run can go from a state with priority $\max(P)$ to the next with priority $\max(P)$. If a particular run does not encounter a state with priority $\max(P)$ anymore, we just stay in the formula where we were. Like this, the easier formulas can be combined to construct the wanted formula. All that is left to show, is that the cost function defined by this combination is the right one.

Suppose that Eloise and Abelard have fixed optimal strategies, meaning that the run ρ through $\mathcal{A}$ we are considering is fixed. We have to show that the highest checked value in the automaton and in our formula are similar.

Define a subrun as a part of a run. If we now cut ρ into two every time we encounter a state in R, we can see these subruns as runs in $\mathcal{A}_{free(R)}$ for the first subrun and in $\mathcal{A}_{start(q_i)free(R)}$ for some $i \in \{1, \ldots, k\}$ for all other subruns. By construction, we know that when we are in a subrun that can be seen as a run of $\mathcal{A}_{start(q_i)free(R)}$, then in φ, we are in a subformula corresponding to φ_i. Thus, what happens to the counter if following an optimal strategy is similar by (1) and (2), meaning that the highest checked value in $\mathcal{A}$ and in φ will be similar.

Next to this, by construction it is ensured that when we loop infinitely often through a particular μ or ν subformula of φ, it will always be a subformula of a φ_i for some i. Which means that the parity condition is now indeed correctly reflected in this translation. ♣

To clarify this construction more, let us look at the following example.

Example 4.10. Recall the automaton introduced in example 3.32. This automaton recognises the function $h(t) = |t|_a$, which counts how often the label a occurs in t. This means that the formula that we will construct in this example will also recognise this function.

First note that the given automaton is already presented in the required format: we can define the priority function Ω as $\Omega(q) = \Omega(r) = 2$ and $\Omega(s) = 1$.

If we follow the construction, we start with an induction step, so something we need to know is what the strongly connected components are in this automaton. This is directly visible in the following graphical representation of $\mathcal{A}$. We can conclude that $R = \{q, r\}$, which means that from now on we will consider the automaton $\mathcal{A}' = \mathcal{A}_{start(q)}$. Because $\hat{q}$ is not in a strongly connected component, it does not really matter what the priority of this state is, since we cannot get back there anyway. By construction we know that $\Omega(\hat{q}) = 2$, which

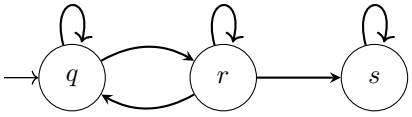

Figure 4.3: The automaton $\mathcal{A}$.

means that the corresponding formula is a ν-formula, but this can just as well be a μ.

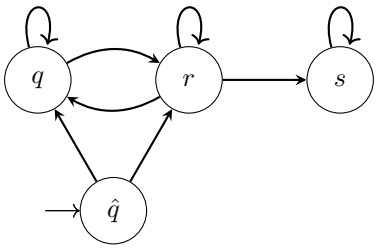

Figure 4.4: The automaton $\mathcal{A}_{start(q)}$.

The next step will be considering the automata $\mathcal{A}'_{free(R)}$, $\mathcal{A}'_{start(q)free(R)}$ and $\mathcal{A}'_{start(r)free(R)}$.

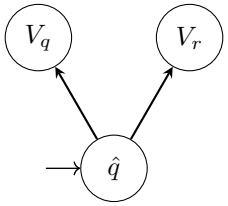

Figure 4.5: The automaton $\mathcal{A}'_{free(R)}$.

The next step would be to work out the formulas that correspond to these automata. We only work out the formula for $\mathcal{A}'_{free(R)}$:

$$\begin{aligned}
\varphi_R &= \nu X_{\hat{q}}.((p_a \wedge \delta'(\hat{q}, a)) \vee (p_b \wedge \delta'(\hat{q}, b)) \\
&= (p_a \wedge (\Diamond(p_0 \wedge ic\varphi_{q,\{\hat{q}\}}) \wedge \Diamond(p_1 \wedge ic\varphi_{q,\{\hat{q}\}})))\vee \\
&\quad (p_b \wedge (\Diamond(p_0 \wedge ic\varphi_{q,\{\hat{q}\}}) \wedge \Diamond(p_1 \wedge ic\varphi_{q,\{\hat{q}\}}))\vee \\
&\qquad (\Diamond(p_0 \wedge \varphi_{r,\{\hat{q}\}}) \wedge \Diamond(p_1 \wedge \varphi_{r,\{\hat{q}\}}))\vee \\
&\qquad (\Diamond(p_0 \wedge \varphi_{q,\{\hat{q}\}}) \wedge \Diamond(p_1 \wedge \varphi_{r,\{\hat{q}\}}))\vee \\
&\qquad (\Diamond(p_0 \wedge \varphi_{r,\{\hat{q}\}}) \wedge \Diamond(p_1 \wedge \varphi_{q,\{\hat{q}\}}))).
\end{aligned}$$

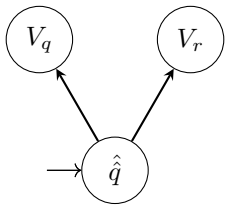

Figure 4.6: The automaton $\mathcal{A}'_{start(q)free(R)}$.

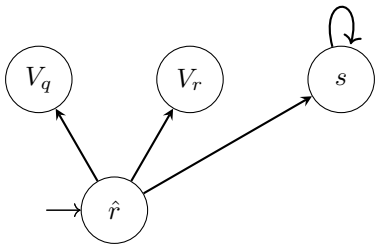

Figure 4.7: The automaton $\mathcal{A}'_{start(r)free(R)}$.

Because both q and r are states that are moved into variables. Every occurrence of $\varphi_{q,\{\hat{q}\}}$ will be replaced by V_q, and thus the formula corresponding to $\mathcal{A}'_{start(q)free(R)}$ will come in those places and the similar thing happens for every occurrence of $\varphi_{r,\{\hat{q}\}}$, but then for r instead of q.

We can proof a similar theorem for S-automata. The only things we have to do are changing some words in the induction step, that is, swap words like minimal and maximal, increase and decrease, zero and infinite, and define the transformations $\mathcal{A}_{free(X)}$ and $\mathcal{A}_{start(q)}$ for S-automata, which boils down to changing all 'B's to 'S's.

Theorem 4.11. *For every cost automaton $\mathcal{A} = (Q, \mathbb{A}, q_0, Cost_S^{\{1\},P}, \delta)$ over infinite trees, we can construct a cost modal μ-calculus formula $\varphi_{\mathcal{A}}$ such that*

$$[\![\mathcal{A}]\!](t) \approx value_S(G^t(\varphi_{\mathcal{A}})).$$

4.3 Cost logic to cost automata

For the next translation it is useful that given a tree with labels in $\mathbb{A}$, we also encode whether a node is a left or a right child with the new set of labels $\mathbb{A}' = \mathbb{A} \times \{0, 1\}$. Formally, we can define the new labelling function $t' : \{0,1\}^* \to \mathbb{A}'$ given the old labelling function $t : \{0,1\}^* \to \mathbb{A}$ as follows.

$$t'(x) = \begin{cases} t(x) \times 0 & \text{if } x \in \rho(p_0), \\ t(x) \times 1 & \text{if } x \in \rho(p_1), \end{cases}$$

for every $x \in \{0,1\}^*$. When a set of labels is of the form $\mathbb{A} \times \{0,1\}$ we will denote such a set with $\mathbb{A}'$.

Say $\alpha(\varphi)$ is the alternation depth of φ. The alternation depth of a formula is in fact the number of alternations between μ- and ν-operators. For some applications a more advanced definition of alternation depth is used, but that is superfluous at this point.

Definition 4.12. Given φ with a B-semantics, let us define

$$\mathcal{A}(\varphi) = (Q, \mathbb{A}', \langle\varphi\rangle, Cost_B^{\{1\},P}, \delta),$$

and given φ with an S-semantics, define

$$\mathcal{A}(\varphi) = (Q, \mathbb{A}', \langle\varphi\rangle, Cost_S^{\{1\},P}, \delta),$$

where in both cases, P is the set of priorities used,

$$Q = \{\langle\varphi'\rangle : \varphi' \text{ subformula of } \varphi\} \cup \{\langle\bot\rangle, \langle\top\rangle\},$$

and

$$\begin{aligned}
\delta(\langle\bot\rangle, (e,*)) &:= (0, (\epsilon, 1), \langle\bot\rangle),\\
\delta(\langle\top\rangle, (e,*)) &:= (0, (\epsilon, 0), \langle\top\rangle),\\
\delta(\langle p_d\rangle, (e,*)) &:= \begin{cases} (0, (\epsilon, 0), \langle\top\rangle) & \text{if } d = e,\\ (0, (\epsilon, 1), \langle\bot\rangle) & \text{if } d \neq e,\end{cases}\\
\delta(\langle \neg p_d\rangle, (e,*)) &:= \begin{cases} (0, (\epsilon, 1), \langle\bot\rangle) & \text{if } d = e,\\ (0, (\epsilon, 0), \langle\top\rangle) & \text{if } d \neq e,\end{cases}\\
\delta(\langle p_0\rangle, (e,0)) &:= (0, (\epsilon, 0), \langle\top\rangle),\\
\delta(\langle p_0\rangle, (e,1)) &:= (0, (\epsilon, 1), \langle\bot\rangle),\\
\delta(\langle p_1\rangle, (e,0)) &:= (0, (\epsilon, 1), \langle\bot\rangle),\\
\delta(\langle p_1\rangle, (e,1)) &:= (0, (\epsilon, 0), \langle\top\rangle),\\
\delta(\langle c\psi\rangle, (e,*)) &:= (\epsilon, (c, 0), \langle\psi\rangle),\\
\delta(\langle \chi \wedge \chi'\rangle, (e,*)) &:= (\epsilon, (\epsilon, 0), \langle\chi\rangle) \wedge (\epsilon, (\epsilon, 0), \langle\chi'\rangle),\\
\delta(\langle \chi \vee \chi'\rangle, (e,*)) &:= (\epsilon, (\epsilon, 0), \langle\chi\rangle) \vee (\epsilon, (\epsilon, 0), \langle\chi'\rangle),\\
\delta(\langle \Box\chi\rangle, (e,*)) &:= (0, (\epsilon, 0), \langle\chi\rangle) \wedge (1, (\epsilon, 0), \langle\chi\rangle),\\
\delta(\langle \Diamond\chi\rangle, (e,*)) &:= (0, (\epsilon, 0), \langle\chi\rangle) \vee (1, (\epsilon, 0), \langle\chi\rangle),\\
\delta(\langle Z\rangle, (e,*)) &:= (\epsilon, (\epsilon, 0), \langle\psi\rangle),\\
\delta(\langle \sigma Z.\psi\rangle, (e,*)) &:= (\epsilon, (\epsilon, \Omega(\sigma Z.\psi)), \langle\psi\rangle),
\end{aligned}$$

with $d, e \in \mathbb{A}$, $\sigma \in \{\mu, \nu\}$ and $c \in \mathbb{C}$. The $\Omega(\sigma Z.\psi)$ is defined as follows:

$$\Omega(\sigma Z.\psi) = \begin{cases} 2\alpha(\sigma Z.\psi) & \text{if } \sigma = \mu,\\ 2\alpha(\sigma Z.\psi) + 1 & \text{if } \sigma = \nu.\end{cases}$$

Notice that this is not per se the most efficient use of priorities: in most cases less priorities would be enough if assigned in a smarter way. This is only important when the complexity of an automaton plays a role, which is not the case in this thesis.

Another remark is that in this definition we use so called 'silent transitions': transitions that do not pick the left or the right child, but stay where we are. For most formulas, this boils down to postponing the choice for the left or the right child. In general, the expressability of automata does not change when adding silent transitions.

We have not shown yet that this is a correct translation, that is, that the cost functions given by this automaton and the cost formula correspond to each other. The following theorem states this formally.

Theorem 4.13.

$$[\![\mathcal{A}(\varphi)]\!](t) \approx value(\mathcal{G}^t(\varphi)).$$

Proof. The proof goes by induction on the construction of the cost modal μ-calculus formula. Notice that it is enough to show for each induction step that players can make a similar choice and that what happens to the counter in making those choices is similar.

The only case we discuss is $\varphi = p_d$ for some $d \in \mathbb{A}$, with a B-semantics: for the S-semantics we can give a similar argument. In the automaton this would mean that we either end up in an infinite $(\epsilon, 0)$ or $(\epsilon, 1)$ chain, meaning that we either get ∞ as outcome, because of the priorities, or nothing happens to the counter. On the formula side, the correction function $f(\pi)$ mimics the result or nothing happens as well. Clearly, all other base cases are similar.

For all other cases, there is an immediate one-to-one correspondence between both games. Since the $\Omega(\sigma Z.\psi)$ is designed to reflect the condition of infinitely often seen fixed point operators, also the parity condition works. Thus, this is indeed a working translation. ♣

Since we now have two working translations, we can conclude this section with the following theorem.

Corollary 4.14. *A cost function over infinite trees is regular if and only if it is definable in cost modal μ-calculus.*

4.4 Properties of cost logic

We see some useful things in this part of the thesis. First of all, we prove directly that B- and S-logic are equivalent in expressability. This of course already follows directly from the given translations between the logic and cost automata over infinite trees and the fact that B- and S-automata over infinite trees are equivalent in expressability. However, proving directly gives more insight in how this logic works.

Next to this, we show that the B-logic has a normal form. From the direct translations given in the first part, also a normal form for S-logic follows, although a less neater one.

4.4.1 B- and S-logic are equivalent

The following two theorems give the two translations between B- and S-logic. After that, we can conclude the desired result.

Theorem 4.15. *Given a cost modal μ-calculus formula φ with B-semantics, we can construct a cost modal μ-calculus formula ψ with S-semantics such that*

$$value_B(\mathcal{G}^t(\varphi)) \approx value_S(\mathcal{G}^t(\psi)),$$

for all labelled trees t.

Proof. Fix some tree t. Start with reversing the role of the players in the formula. This means switching conjunctions and disjunctions, and boxes and diamonds.

If we would give this updated formula an S-semantics, Eloise would become the maximising player, meaning that the goal of the players reverses too. Thus far, it is quite straightforward. The only thing that is still problematic is that instead of taking the supremum of checked values, we now take the infimum. In other words, we have not yet corrected for the change in cost function.

First of all, we want the correction function $\overline{f}(\pi)$ to give similar results as $f(\pi)$ in the B-semantics. This can be achieved by replacing all predicates p with the opposite predicate $\overline{p}$ and by switching least to greatest fixed point and greatest to least fixed points. Next to this, change all ic-counter actions to i's and let the maximising player, Eloise, decide when to check the counter, after which nothing happens anymore. Moreover, add some greatest fixed point formulas that function as the infinite loops where nothing happens. More precisely, replace X by $X \vee cr\nu Z.Z$, where Z is a variable that has not been used before. Name the formula that we get after these transformations ψ.

This is indeed the formula we are looking for. First of all, remark that both the role and the goal of the players has reversed, so what was an optimal strategy for Eloise, becomes one for Abelard and the other way around. The only difference is that Eloise in every loop now has the option to check the counter. It follows that if in optimal strategies for the game given by φ a counter is checked at a certain value, a counter in the game given by ψ will get to approximately the same value if also here optimal (and thus similar) strategies are followed. Note that there can be small differences, bounded by the maximal amount of ic's that occur in a loop, since the reset option can occur in a loop. Clearly, Eloise can then check the value, meaning that if $value_B(\mathcal{G}^t(\varphi)) = n$ for some natural number n, $\alpha(value_S(\mathcal{G}^t(\psi))) \geq n$ for some correction function α.

If now $value_S(\mathcal{G}^t(\psi)) = n$ for some natural number n, it follows by similar reasoning that there exists some α such that $\alpha(value_B(\mathcal{G}^t(\varphi))) \geq n$. Thus,

$$value_B(\mathcal{G}^t(\varphi)) \approx value_S(\mathcal{G}^t(\psi)),$$

for all labelled trees t. ♣

We can also proof the other direction.

Theorem 4.16. *Given a cost modal μ-calculus formula φ with S-semantics, we can construct a cost modal μ-calculus formula ψ with B-semantics such that*

$$value_S(\mathcal{G}^t(\varphi)) \approx value_B(\mathcal{G}^t(\psi)),$$

for all labelled trees t.

Proof. Again, fix some tree t. The basic idea of this proof is similar: we start by taking $\overline{\varphi}$ as initial formula. However, the rest of the translation is a bit more complicated. The idea is that when $value_S(\mathcal{G}^t(\varphi)) = n$ for some natural number n, Eloise has to find the best moment to start keeping track of the counter actions after which we finish when reading a cr. This is to mimic the S-semantics. If Eloise thinks it is not the right moment yet, she can decide to go to a sort of resting part of the formula that brings us back to a part of the formula that can keep track of what is happening as soon as the counter is reset.

Notice that what we are doing in this case is in fact treating the formula as a automaton with three states: q, r and s. We start in state r. In this state we ic the counter and stay in r, or decide to go to the resting state q while doing nothing with the counter if the original counter is increased. If originally the counter is reset, we still do this. When the counter is cr, we go to the state s where nothing can happen to the counter anymore. If in state q, we do nothing until we read either an r or an cr action; if so, we reset the counter and go back to state r. Graphically, this looks like the following automaton.

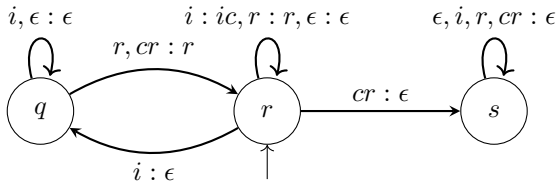

Figure 4.8: Translate counters from S-semantics to B-semantics

Thus, what is left to do, is adapting $\overline{\varphi}$ in a way that captures this: take $\psi = (\overline{\varphi})_{r,\emptyset}$, where we define $(.)_{r,V}$ and $(.)_{q,V}$ as follows. Name the formula

inside a fixed point $\sigma X\ \psi_X$. Take $t \in \{q, r\}$ and $\sigma \in \{\mu, \nu\}$, and define

$$
\begin{aligned}
(\varphi \wedge \varphi')_{t,V} &:= (\varphi)_{t,V} \wedge (\varphi')_{t,V}, & (\Box\varphi)_{t,V} &:= \Box(\varphi)_{t,V},\\
(\varphi \vee \varphi')_{t,V} &:= (\varphi)_{t,V} \vee (\varphi')_{t,V}, & (\Diamond\varphi)_{t,V} &:= \Diamond(\varphi)_{t,V},\\
(p)_{t,V} &:= p, & (\neg p)_{t,V} &:= \neg p,\\
(i\varphi)_{q,V} &:= (\varphi)_{q,V}, & (i\varphi)_{r,V} &:= (\varphi)_{q,V} \vee ic(\varphi)_{r,V},\\
(r\varphi)_{q,V} &:= r(\varphi)_{r,V}, & (r\varphi)_{r,V} &:= r(\varphi)_{r,V},\\
(cr\varphi)_{q,V} &:= r(\varphi)_{r,V}, & (cr\varphi)_{r,V} &:= \nu Z.Z,\\
(\sigma X.\psi_X)_{t,V} &:= \sigma X_t.(\psi_X)_{r,V\cup\{X_t\}},
\end{aligned}
$$

$$
(X)_{t,V} := \begin{cases} X_t & \text{if } X_t \in V,\\ \sigma X_t.(\psi_X)_{r,V\cup\{X_t\}} & \text{if } X_q \notin V \wedge X_r \notin V,\\ (\sigma X_t.(\psi_X)_{r,V\cup\{X_t\}})^* & \text{otherwise,}\end{cases}
$$

where $(\sigma X.\psi_X)^*$ is an operation that gives all fixed point variables in $\{Z \in Var : \sigma Z.\psi_Z \in Sub(\psi_X)$ a name that has not been used before. This is done to ensure that the order of the fixed point variables stays fixed. Namely, because we take in fact two copies of a fixed point, a q- and an r-version, what can happen is that the second copy of the fixed point ends up inside a first copy of a fixed point that is originally a subformula. Clearly, this can mix up the process of acceptance.

All that is left to do, is convince ourselves that ψ recognises the cost function we started with. So, suppose that $value_S(\mathcal{G}^t(\varphi)) = n$ for some natural number n. This means that the lowest checked number is n, or that the correction function $\overline{f}$ has 0 as result. In the first case, there exists a strategy for Eloise, namely stay in state r as soon as the series of i's starts that resulted in the lowest checked number, such that the value of that strategy is n. Thus, $value_B(\mathcal{G}^t(\varphi)) \leq n$. If the correction function gives 0 as result, a strategy for Eloise is to always go to state q as soon as possible. If so, the correction function f will also give 0 as result, resulting in what we wanted to find. All other cases have similar solutions and work as well. Therefore, we can conclude

$$value_S(\mathcal{G}^t(\varphi) \approx value_B(\mathcal{G}^t(\psi)),$$

for all labelled trees t. ♣

Corollary 4.17. *Cost modal μ-calculus with B-semantics is exactly as expressive as cost modal μ-calculus with S-semantics.*

4.4.2 Counter normal form theorem

Let us, for $\sigma \in \{\mu, \nu\}$, all variables X and all cost modal μ-calculus formulas ψ, use $\sigma X^N.\psi$ as a shorthand for $\sigma X.ic\psi$. The idea is that the counter N counts how often we loop through the given fixed point operator. Name this use of counters 'placing a counter on a least or greatest fixed point operator'. Now we are ready to define a normal form.

Definition 4.18 (Counter normal form)**.** Say that a cost modal μ-calculus formula is in *counter normal form* when all counters are placed on least fixed points.

We can continue with the following theorem, which means that we also have the right to call it a normal form.

Theorem 4.19. *Given any cost modal μ-calculus formula φ with B-semantics, there exists a formula ψ in counter normal form that recognises the same cost function.*

Proof. First, we argue why $\mathbb{B}' = \{\epsilon, ic\}$ is enough as the set of used counter actions. After that, we discuss why we can also restrict the area where ic can occur.

The reset action is redundant: the only use of the reset action is to start counting all over again. Since we have players with a minimising and a maximising goal, we can let them play a game to determine when to start counting instead of counting everything and reset in the mean time.

For the placement of the counter actions, it only makes sense to place them inside fixed points. That is, the game visits them at most once if outside, meaning that the same function where such a counter is left away recognises the same cost function with correction function $\alpha(n) = n + 1$.

Furthermore, it is needless to have more than one ic in one loop in a play. To see this, remark that if for a particular formula the maximum amount of ic's that can be seen in a loop is k, we can construct a similar formula where there is maximal one ic occurring directly after a fixed point operator such that they recognise the same cost function. This time the correction function $\alpha(n) = kn$ suffices. Remark that we might need to add more fixed point operators if we only want to increase the counter if the players make a particular choice.

The last thing we have to discuss is why we only need least fixed points with counters placed on them. This follows because the difference between least and greatest fixed points evaporates if a counter is placed on them. ♣

Combine this counter normal form theorem with the construction to make an S-logic formula out of a B-one, to find a restricted form of S-logic as well that is still as expressive as the logic without those restrictions.

4.5 A fragment of cost modal μ-calculus

In the paper [2] there is one section devoted to describing a cost version of modal μ-calculus. This logic is also equivalent in expressiveness to alternating cost automata, but is more restricted in how it incorporates the counters. Their idea is to put counters on the fixed points to count how often you loop through those fixed points. If we consider how cost monadic second order logic is defined by Colcombet, see for instance [6], [7] and [5], this seems the most straightforward way to apply counters in modal μ-calculus, until you realise that you can also put them directly into your formula.

In the rest of this section we consider a fragment of formulas of the cost modal μ-calculus that can be defined as

$$\varphi ::= \mu Z^N.\psi,$$

where

$$\psi ::= p_e \mid \neg p_e \mid Z \mid \psi \wedge \psi \mid \psi \vee \psi \mid \Box\psi \mid \Diamond\psi.$$

Here p_e is a predicate for every $e \in \mathbb{A}$, which is satisfied when a node has label e. The nice part about this fragment is that we do not need cost games to define the semantics, as we will see.

We use the notation νZ^N in the following way

$$[\![\nu Z^N.\psi(Z)]\!](t,V) \approx [\![\mu Z^N.\neg\psi(\neg Z)]\!](t,V).$$

Note that $\neg\psi$ is officially not an allowed formula in the cost modal μ-calculus. Thus, read $\neg\psi$ as an abbreviation of the formula where the $\neg$ is pushed to the leaves of the syntax tree.

Fix a labelled tree t, where $t(x)$ denotes the label of node $x \in \{0,1\}^*$, and a valuation $V : Var \to \mathcal{P}(\{0,1\}^*)$. In this fragment $Var = \{Z\}$. We define

$$\begin{aligned}
||p_e||_V^t &:= \{x \in \{0,1\}^* : t(x) = e\},\\
||\neg p_e||_V^t &:= \{x \in \{0,1\}^* : t(x) \neq e\},\\
||Z||_V^t &:= V(Z),\\
||\varphi \wedge \psi||_V^t &:= ||\varphi||_V^t \cap ||\psi||_V^t,\\
||\varphi \vee \psi||_V^t &:= ||\varphi||_V^t \cup ||\psi||_V^t,\\
||\Box\varphi||_V^t &:= \{x \in \{0,1\}^* : x0 \in ||\varphi||_V^t \wedge x1 \in ||\varphi||_V^t\},\\
||\Diamond\varphi||_V^t &:= \{x \in \{0,1\}^* : x0 \in ||\varphi||_V^t \vee x1 \in ||\varphi||_V^t\}.
\end{aligned}$$

So, $||\cdot||_V^t$, is a function that maps a modal formula to the set of nodes where that formula holds. It does not make sense to define the semantics of $\mu Z^N.\psi$ and $\nu Z^N.\psi$ in terms of $\mathcal{P}(\{0,1\}^*)$ as well, because the meaning of such a formula is the cost of the particular tree, i.e. a natural number or infinity.

We can now define the cost of a cost modal μ-calculus formula as follows:

$$\begin{aligned}
[\![\mu Z^N.\psi]\!]_V^t &:= \inf\{n \in \mathbb{N}_\infty : \epsilon \in \bigcup_{i \leq n} ||\mu^i Z.\psi||_V^t\},\\
[\![\nu Z^N.\psi]\!]_V^t &:= \sup\{n \in \mathbb{N}_\infty : \epsilon \in \bigcap_{i \leq n} ||\nu^i Z.\psi||_V^t\}.
\end{aligned}$$

The game always starts in the root, so it makes sense to require that the root is still in the approximation.

We find that this logic is indeed a fragment of the cost modal μ-calculus.

Proposition 4.20.

$$[\![\mu Z^N.\psi(Z)]\!]_V^t \approx value(\mathcal{G}^t(\mu Z.ic(\psi(Z)))).$$

Some examples conclude this section.

Example 4.21. Let us have a look at the formula $\varphi = \mu X^N.p_a \vee \Diamond X$. We want to evaluate this cost formula over the tree t from which the start can be found in figure 4.21. The rest of the tree only exists of nodes with label b.

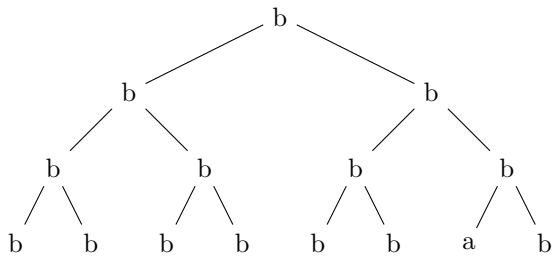

Figure 4.9: The start of tree t.

We find:

$$\begin{aligned}
||\mu^0 Z.\varphi_0||_V^t &= \emptyset \\
||\mu^1 Z.\varphi_0||_V^t &= ||p_a \vee \Diamond\emptyset||_V^t \\
&= \{110\} \\
||\mu^2 Z.\varphi_0||_V^t &= ||p_a \vee \Diamond\{110\}||_V^t \\
&= \{11, 110\} \\
||\mu^3 Z.\varphi_0||_V^t &= ||p_a \vee \Diamond\{11, 110\}||_V^t \\
&= \{1, 11, 110\} \\
||\mu^4 Z.\varphi_0||_V^t &= ||p_a \vee \Diamond\{1, 11, 110\}||_V^t \\
&= \{\epsilon, 1, 11, 110\}.
\end{aligned}$$

Thus, $[\![\varphi_0]\!]_V^t = 4$, which is how long it minimally takes before we see an a.

Example 4.22. The formula $\nu Z^N.p_a \wedge \Diamond Z$ measures the length of the longest a-path starting in ϵ.

5 Proof theory of cost logic

In the previous chapter we introduced a cost fixed point logic. In this chapter we make a start on setting up a sound and complete sequent calculus for this logic. We do this by considering a sequent calculus for the formulas in cost modal μ-calculus without counters, i.e. plain modal μ-calculus, and extending the calculus by adding rules for counter actions. Soundness and completeness are proved for a restricted form. An unrestricted acceptance condition is proposed and conjectured to work for the full logic. We end the chapter by connecting the story to the decidability problem.

5.1 Plain formulas

Before working on the cost version of modal μ-calculus, we review cost formulas without any counter actions; we call these plain formulas. Notice that were then in fact considering standard modal μ-calculus, which means that $t \models \varphi$ is defined. The equivalence becomes precise in the following proposition.

Proposition 5.1. *Given a plain cost formula φ, we have*

1. $[\varphi]_B(t) < \infty$ *iff* $t \models \varphi$,
2. $[\varphi]_S(t) = \infty$ *iff* $t \models \varphi$, *and*
3. $[\varphi]_X(t) \in \{0, \infty\}$ *for* $X \in \{B, S\}$.

Proof. If there are no counter actions, then $value_B(\pi) = f(\pi)$ and $value_S(\pi) = \overline{f}(\pi)$. The result follows directly. ♣

In general, one direction of the implications in the previous proposition always holds.

Proposition 5.2. *Given a cost formula φ and the plain cost formula φ' obtained from φ by dropping the counter actions,*

1. *if* $[\varphi]_B(t) < \infty$, *then* $t \models \varphi'$, *and*
2. *if* $[\varphi]_S(t) = \infty$, *then* $t \models \varphi'$.

Proof. Suppose $t \not\models \varphi'$. This means that there exists a winning strategy for Abelard in the model checking game. This same strategy is an optimal strategy for Abelard in the game played to determine the cost of φ; by following the same ideas, he can make sure that the correction function becomes ∞, resp. 0, which makes the cost ∞, resp. 0. ♣

Given the observation above, it is not so hard to see the following connection.

Proposition 5.3. *Given φ, ψ plain formulas, then $[\varphi]_S \preceq [\psi]_S$ iff $\varphi \to \psi$ is valid.*

Proof. $[\varphi]_S \preceq [\psi]_S$ iff $[\varphi]_S(t) < \infty$ or $[\psi]_S(t) = \infty$ iff $t \not\models \varphi$ or $t \models \psi$ iff $\varphi \to \psi$ is valid. ♣

This suggests that we can design a sequent calculus for cost modal μ-calculus by taking the rules of the two-sided sequent calculus for modal μ-calculus together with some rules for the counter actions.

5.2 Sequent calculus

To define a sequent calculus, we first need to know how to interpret a sequent.

Definition 5.4. The interpretation $\mathcal{I}$ of a sequent is defined as

$$\mathcal{I}(\Gamma \Rightarrow \Delta) := [\bigwedge \Gamma]_S \preceq [\bigvee \Delta]_S.$$

The presentation of the sequent calculus denoted by $2DT$ and utilised below is taken from [23].

$$\frac{}{\Gamma, p \Rightarrow p, \Delta}\,(Ax)_1 \qquad \frac{}{\Gamma \Rightarrow p, \neg p, \Delta}\,(Ax)_2$$

$$\frac{}{\Gamma, \neg p \Rightarrow \neg p, \Delta}\,(Ax)_3 \qquad \frac{}{\Gamma, p, \neg p \Rightarrow \Delta}\,(Ax)_4$$

$$\frac{\Gamma, \varphi_0, \varphi_1 \Rightarrow \Delta}{\Gamma, \varphi_0 \wedge \varphi_1 \Rightarrow \Delta}\,(\wedge)_L \qquad \frac{\Gamma \Rightarrow \varphi_0, \Delta \qquad \Gamma \Rightarrow \varphi_1, \Delta}{\Gamma \Rightarrow \varphi_0 \wedge \varphi_1, \Delta}\,(\wedge)_R$$

$$\frac{\Gamma, \varphi_0 \Rightarrow \Delta \qquad \Gamma, \varphi_1 \Rightarrow \Delta}{\Gamma, \varphi_0 \vee \varphi_1 \Rightarrow \Delta}\,(\vee)_L \qquad \frac{\Gamma \Rightarrow \varphi_0, \varphi_1, \Delta}{\Gamma \Rightarrow \varphi_0 \vee \varphi_1, \Delta}\,(\vee)_R$$

$$\frac{\Gamma, \varphi \Rightarrow \Delta}{\Theta, \Box\Gamma, \Diamond\varphi \Rightarrow \Diamond\Delta, \Sigma}\,(\Box)_L \qquad \frac{\Gamma \Rightarrow \varphi, \Delta}{\Theta, \Box\Gamma \Rightarrow \Box\varphi, \Diamond\Delta, \Sigma}\,(\Box)_R$$

$$\frac{\Gamma, Z \Rightarrow \Delta}{\Gamma, \sigma Z.\varphi(Z) \Rightarrow \Delta}\,(\sigma)_L \qquad \frac{\Gamma \Rightarrow Z, \Delta}{\Gamma \Rightarrow \sigma Z.\varphi(Z), \Delta}\,(\sigma)_R$$

$$\frac{\Gamma, \varphi(Z) \Rightarrow \Delta}{\Gamma, Z \Rightarrow \Delta}\,(Z)_L \qquad \frac{\Gamma \Rightarrow \varphi(Z), \Delta}{\Gamma \Rightarrow Z, \Delta}\,(Z)_R,$$

where

- p is a proposition,
- φ is a formula,
- capital Greek letters denote finite sets of formulas, which can possibly be empty,
- $\Box\Gamma$ is an abbreviation for a (possibly emtpy) set of formulas of the form $\Box\varphi$,

- $\Diamond\Delta$ is an abbreviation for a (possibly empty) set of formulas of the form $\Diamond\varphi$, and
- $\sigma \in \{\mu, \nu\}$.

This is already enough to make derivations for plain formulas.

The notion of a *trace* is the standard one. A *μ-trace* is an infinite trace such that the unique variable that occurs infinitely often and subsumes every other variable that occurs infinitely often is a least fixed point. When it is a greatest fixed point, we call it a *ν-trace*. See for instance [23] for a detailed definition. We use the following property to capture validity.

Definition 5.5 (Cost proof)**.** Given φ and ψ plain cost formulas. A *proof of* $[\varphi]_S \preceq [\psi]_S$ is a (possibly infinite) tree of sequents with $\varphi \Rightarrow \psi$ as root following the rules of the calculus $2DT$ and satisfying the *global trace condition*: every infinite path has either an infinite ν-trace on the right or an infinite μ-trace on the left.

Now we are ready to define the full set of sequent rules for cost modal μ-calculus.

Definition 5.6. The calculus $2DTC$ is formed by taking all sequent rules of $2DT$ combined with:

$$\frac{\Gamma, \varphi \Rightarrow \Delta}{\Gamma, c\varphi \Rightarrow \Delta}\,(c)_L \qquad \frac{\Gamma \Rightarrow \varphi, \Delta}{\Gamma \Rightarrow c\varphi, \Delta.}\,(c)_R.$$

Clearly the property defined in definition 5.5 must be adjusted when one of the sides, or even both sides, contain counter actions. In this thesis we only consider the case $\varphi \Rightarrow \psi$ in which φ is a plain formula and ψ is allowed to have counter actions and both are interpreted as S-cost functions. First we need a notion of the value of a trace, after which we can focus on formulating a notion of a 'cost' proof.

Definition 5.7. The *value of a trace* is the infimum of all checked values on that trace.

So, suppose we have $[\varphi]_S \preceq [\psi]_S$. This holds if and only if

- for all trees t we have $[\varphi]_S(t) = \infty$ implies $[\psi]_S(t) = \infty$, if and only if
- for all trees t we find $t \not\models \varphi$ or $[\psi]_S(t) = \infty$.

Note that by proposition 5.1, we know that $[\varphi]_S(t) = \infty$ is equivalent to $t \models \varphi$. This motivates the following property for accepting derivations.

Definition 5.8 (Cost proof)**.** Given φ a plain cost formula and ψ any cost formula, a *proof of* $[\varphi]_S \preceq [\psi]_S$ is a (possibly infinite) tree of sequents with $\varphi \Rightarrow \psi$ as root, following the rules of the calculus $2DTC$ and satisfying the condition: every infinite path has either for every natural number n an infinite ν-trace that has value greater or equal to n on the right, or an infinite μ-trace on the left. We use the notation $2DTC \vdash \varphi \Rightarrow \psi$ if there exists such a proof.

If there is no restriction on the formulas we conjecture that the following global trace condition yields a sound and complete sequent calculus for the full cost logic.

Definition 5.9 (Cost proof)**.** Given two cost formulas φ and ψ, a *proof* of $[\varphi]_S \preceq [\psi]_S$ is a (possibly infinite) tree of sequents with $\varphi \Rightarrow \psi$ as root, following the rules of the calculus $2DTC$ and satisfying the condition: every infinite path has either for every natural number n an infinite ν-trace that has value greater or equal to n on the right, or an infinite μ-trace on the left, or there exists a natural number m such that all infinite ν-traces on the left have value at most m. We use the notation $2DTC \vdash \varphi \Rightarrow \psi$ if there exists such a proof.

Conjecture 5.10. *Given two cost formulas φ and ψ, we have $[\varphi]_S \preceq [\psi]_S$ if and only if there exists a derivation in $2DTC$ satisfying the global trace condition described in definition 5.9.*

5.3 Soundness and completeness

In this subsection we prove that a restricted form of our proof system is sound and complete.

Theorem 5.11 (Soundness)**.** *Given a plain cost formula φ and a cost formula ψ, if $2DTC \vdash \varphi \Rightarrow \psi$, then $[\varphi]_S \preceq [\psi]_S$.*

Proof. Suppose we have p, a proof of $\varphi \Rightarrow \psi$, and suppose that $[\varphi]_S \not\preceq [\psi]_S$. We derive a contradiction.

We know that $[\varphi]_S \not\preceq [\psi]_S$, so let t be such that $[\varphi]_S(t) = \infty$, and $[\psi]_S(t) = N < \infty$. Thus, if both players play optimal strategies, respectively ∞ and N will be the resulting values. Name the plays resulting in these values π_φ and π_ψ.

There are three possible reasons for $[\psi]_S(t) = N < \infty$:

1. either π_ψ is finite and ending in a proposition that does not hold at that point,
2. or π_ψ is infinite, but does not satisfy the parity condition,
3. or there exists an n such that we check the counter before it is increased more than n times.

In the first two cases the counters do not play a role at all, which means that when we can consider ψ', the plain version of ψ. We find that $t \not\models \psi$ and thus that there does not exist a proof of $\varphi \preceq \psi$, by the soundness of $2DT$.

Let us now exclude the first two options. Look at the proof p and decide which path to consider by mimicking what happens with the formulas in π_φ and π_ψ: if in π_φ the play continues with the left disjunct, also follow the left path at that point in p. Like this, we find an infinite path through the proof such that every formula on the left side can be reached in a play consistent with Eloise's strategy and every formula on the right in a play consistent with

Abelard's strategy. It is possible that a modal rule causes one of the sides to be empty, but that will not cause any problems, as there will be no traces. Also, the path does not end in an axiom, because that implies that the players did not play optimally: if we for instance end in $(Ax)_2$, this means that in the play π_ψ Eloise gets to choose between p and $\neg p$ for some proposition p, which means that the value of that play can become infinite: a contradiction.

So, we have an infinite path through p consistent with the strategies of both players. There are no counters to check on the left, so we know that the only way $[\varphi]_S(t) = \infty$ in an infinite play is when the parity condition is satisfied, thus, all the traces on the left of this path, which are consistent with Eloises strategy, must be satisfying the parity condition. We can conclude that every trace on the left must be a ν-trace.

On the right, we know that we cannot have increased the counter more than n times before we check it, which means that there exists an N such that we cannot find a trace with a higher value.

We can conclude that this path in p does not satisfy the condition for being a proof. We can conclude that p is not a proof, which completes the contradiction. ♣

Theorem 5.12 (Completeness)**.** *Given a plain cost formula φ and a cost formula ψ, if $[\varphi]_S \preceq [\psi]_S$, then $2DTC \vdash \varphi \Rightarrow \psi$.*

Proof. Just like in the proof of completeness of the infinite sequent calculus, we can view $\Gamma \Rightarrow \Delta$ as a two-player game: Eloise, or prover, selects which rule to apply, and Abelard, the refuter, can choose which premise to continue with. Since there are only two rules with multiple premises, $(\vee)_L$ and $(\wedge)_R$, Abelard does not have many places where he can act.

A strategy for Eloise consists of a tree of sequents with $\Gamma \Rightarrow \Delta$ as root following the rules of $2DTC$. A play is winning for Eloise if every path satisfies the path condition for being a path in a proof. We can conclude that $2DTC \vdash \Gamma \Rightarrow \Delta$ if and only if Eloise has a winning strategy for the game $\Gamma \Rightarrow \Delta$.

Now suppose $[\varphi]_S \preceq [\psi]_S$, but there is no proof of $\varphi \Rightarrow \psi$. This means there is no winning strategy for Eloise in the provability game. Since the game is determined, a direct result of Martin's theorem, Abelard must have a winning strategy. We claim this winning strategy induces a counter model, i.e. a tree t, such that $t \models \varphi$, but $\psi_S(t) = \infty$. t is constructed in the usual way for showing completeness of ill-founded proofs for plain formulas [18]. ♣

5.4 Boundedness

In the first part of this chapter, we showed that there exists a sound and complete sequent calculus with which you can proof whether $[\varphi]_S \preceq [\psi]_S$ for plain φ and any cost formula ψ. This is already a nice result. However, suppose we have a proof of conjecture 5.10. This means that we can start asking questions about the decidability of $[\varphi]_S \preceq [\psi]_S$: can we show the known decidability results, see chapter 3 for an overview, in this calculus? And can we maybe learn other variations of the decidability result that are not so visible from automata theory?

It is known that provability of $2DT$, that is the sequent calculus for plain formulas, is decidable, so it is natural to ask whether for some classes of cost formulas derivability in $2DTC$ is also decidable. Because of the soundness and completeness, this decidability would yield an algorithm for deciding boundedness statements.

If we want to express the known decidability results, remember that we were comparing the cost functions of a nondeterministic B- and nondeterministic S-automaton. The calculus that is suggested in this chapter however works with two times an S-semantics. This shortcoming is easily resolved. Recall that in the previous chapter, we proved that we can translate B-logic into S-logic. If we name the operation defined in that proof $\sim$, we find

1. $[\varphi]_B \approx [\sim\varphi]_S$, and
2. on plain formulas $\sim\varphi$ is equivalent to the negation of φ.

The first point coincides with the result of the proposition, while the second point follows because of the lack of counter actions.

6 Concluding remarks and further work

In this thesis, we have defined an extension of μ-calculus that corresponds nicely with alternating cost automata over infinite trees. Unknown to us until recently, a cost fixed point logic was already proposed in [2]. However, the cost logic presented in this thesis is more general in the sense that counters can occur at any level in the formula and are not restricted to fixed point subformulas, as is the case in [2]. We observed through our counter normal form theorem, theorem 4.19, that this difference does not impact the expressability. Nevertheless, the more flexible syntax of our framework would appear to provide a more transparant setting for a proof-theoretic study of cost logic.

Completely new is the idea of a sequent calculus for cost fixed point logics. In the section on proof theory, we only got to the point of giving an accepting condition and proving soundness and completeness for a restricted form, in which one of the formulas in the relation must be plain. We expect that these results can be extended to a sound and complete sequent calculus for the full cost modal μ-calculus, and further hope that the logical framework developed can contribute solving the open problem of deciding $f \preceq g$ for cost functions f and g that are recognised by alternating cost automata over infinite trees.

References

[1] Bahareh Afshari. "Logic, games and automata (slides)". University of Amsterdam, 2021.

[2] Achim Blumensath et al. "Two-Way Cost Automata and Cost Logics over Infinite Trees". In: CSL-LICS '14. Vienna, Austria: Association for Computing Machinery, 2014, pp. 1–9.

[3] Mikołaj Bojańczyk and Thomas Colcombet. "Bounds in ω-regularity". In: *Logics In Computer Science* (2006), pp. 285–296.

[4] Julian Bradfield and Colin Stirling. "Modal Mu-Calculi". In: *Handbook of Modal Logic*. Ed. by Patrick Blackburn, Johan Van Benthem, and Frank Wolter. Vol. 3. Studies in Logic and Practical Reasoning. Elsevier, 2007. Chap. 12, pp. 721–756.

[5] Thomas Colcombet. "Fonctions régulières de coût". Université Paris Diderot, 2013.

[6] Thomas Colcombet. "Regular cost functions over words". In: (2009).

[7] Thomas Colcombet. "Regular Cost Functions, Part I: Logic and Algebra over Words". In: *Logical methods in computer science* 9.3 (2013).

[8] Thomas Colcombet. "The Theory of Stabilisation Monoids and Regular Cost Functions". In: *Automata, Languages and Programming*. Ed. by Susanne Albers et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 139–150.

[9] Thomas Colcombet and Christof Löding. "Regular Cost Functions over Finite Trees". In: *Proceedings of the 2010 25th Annual IEEE Symposium on Logic in Computer Science*. LICS '10. IEEE Computer Society, 2010, pp. 70–79.

[10] Thomas Colcombet and Christof Löding. "The Non-deterministic Mostowski Hierarchy and Distance-Parity Automata". In: *Automata, Languages and Programming*. Ed. by Luca Aceto et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 398–409.

[11] Stéphane Demri, Valentin Goranko, and Martin Lange. *Temporal Logics in Computer Science: Finite-State Systems*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2016.

[12] E. Allen Emerson and Charanjit S. Jutla. "Tree automata, mu-calculus and determinacy". In: *Proceedings 32nd Annual Symposium of Foundations of Computer Science, FoCS '91* (1991), pp. 368–377.

[13] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, eds. *Automata, Logics, and Infinite Games A Guide to Current Research*. eng. 1st ed. 2002. Lecture Notes in Computer Science, 2500. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002.

[14] Kosaburo Hashiguchi. “Relative star height, star height and finite automata with distance functions”. In: *Formal Properties of Finite Automata and Applications*. Ed. by J. E. Pin. Berlin, Heidelberg: Springer Berlin Heidelberg, 1989, pp. 73–88.

[15] Daniel Kirsten. “Distance desert automata and the star height problem”. In: *RAIRO - Theoretical Informatics and Applications - Informatique Théorique et Applications* 39.3 (2005), pp. 455–509.

[16] Dexter Kozen. “Results on the propositional μ-calculus”. In: *Theoretical Computer Science* 27.3 (1983), pp. 333–354.

[17] Andrzej W. Mostowski. “Games with forbidden positions”. Technical Report 78, Uniwersytet Gdánski, Instytut Matematyki, 1991.

[18] Damian Niwinski and Igor Walukiewicz. “Games for the mu-Calculus”. In: *Theoretical Computer Science* 163 (1996), pp. 99–116.

[19] Robert S. Streett and E. Allen Emerson. “An automata theoretic decision procedure for the propositional mu-calculus”. In: *Information and Computation* 81.3 (1989), pp. 249–264.

[20] Michael Vanden Boom. “Weak Cost Automata over Infinite Trees”. PhD thesis. Baliol College, University of Oxford, 2012.

[21] Michael Vanden Boom. “Weak Cost Monadic Logic over Infinite Trees”. In: *Mathematical Foundations of Computer Science 2011*. Ed. by Filip Murlak and Piotr Sankowski. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 580–591.

[22] Igor Walukiewicz. “Notes on the Propositional mu-calculus: Completeness and Related Results”. In: *Basic Research in Computer Science Notes Series 95*. 1995.

[23] Lukas Zenger. “Proof theory for fragments of the modal mu-calculus”. MA thesis. Amsterdam: Master of Logic, ILLC, University of Amsterdam, 2021.