Regular cost functions over finite trees

Thomas Colcombet
LIAFA, CNRS and Université Paris Diderot,
Case 7014, 75205 Paris Cedex 13, France
thomas.colcombet@liafa.jussieu.fr

Christof Löding
RWTH Aachen, Informatik 7,
52056 Aachen, Germany
loeding@cs.rwth-aachen.de

Abstract

We develop the theory of regular cost functions over finite trees: a quantitative extension to the notion of regular languages of trees: Cost functions map each input (tree) to a value in $\omega+1$, and are considered modulo an equivalence relation which forgets about specific values, but preserves boundedness of functions on all subsets of the domain.

We introduce nondeterministic and alternating finite tree cost automata for describing cost functions. We show that all these forms of automata are effectively equivalent. We also provide decision procedures for them. Finally, following Büchi's seminal idea, we use cost automata for providing decision procedures for cost monadic logic, a quantitative extension of monadic second order logic.

1 Introduction

Since the seminal works of Kleene [20] and Rabin and Scott [24], the theory of regular languages is one of the cornerstones in computer science. The theory has then been extended to infinite words [6], to finite trees [27], and to infinite trees [23]. This latter result is of such importance that it is sometimes called the 'mother of all decidability results'.

Recently, the notion of regular cost function of words has been presented as a candidate for being a quantitative extension to the notion of regular languages [7], while retaining most of the fundamental properties of the original theory such as the equivalence with logic and decidability[1]. A cost function is an equivalence class of the functions from the domain (e.g. words or trees) to $\omega + 1$, modulo an equivalence relation $\approx$ which allows some distortion, but preserves the existence of bounds over each subset of the domain. The model of cost functions is a strict extension to the notion of languages. The objective of this paper is to extend this theory to finite trees.

[1] Regular cost functions differ from other quantitative extensions to regular languages in the sense that they cannot be reduced to such other extensions, and that at the same time they retain very strong closure and decidability properties.

Related works and motivating examples

A prominent question in this theory is the star-height problem. This story begins in 1963 when Eggan formulates the star-height decision problem [10], i.e., decide if a regular language of words can be represented by a regular expressions using at most k nesting of Kleene stars. This problem was quickly considered as central in language theory, and as the most difficult problem in the area. It took twenty-five years before Hashiguchi came up with a proof of decidability spreading over four papers [14, 13, 15, 16]. Hashiguchi used in his proof the model of *distance automata*. A distance automaton is a finite state non-deterministic automaton running over words which can count the number of occurrences of some 'special' states, and hence attach a value to each input. The proof of Hashiguchi relies on a very difficult reduction to the *limitedness* problem, i.e., the existence of a bound over the function computed by an automaton over its domain. Hashiguchi established the decidability of this problem [13]. In 2005, Kirsten gave a much simpler and self-contained proof to the star-height problem [19]. It relied on a reduction to the limitedness problem for a more general class of automata, the nested distance desert automata. The notion of distance automata and its relationship with the tropical semiring has also been the source of many investigations [14, 17, 21, 26, 29].

Other decision problems can also be reduced to limitedness questions over words: in language theory the *finite power property* [25, 12] and the *finite substitution problem* [2, 18], and in model theory the *boundedness problem* of monadic formulas over words [3]. Distance automata are also used in the context of databases and image compression. Automata similar to the ones of Kirsten have also been introduced independently in the context of verification [1].

The automata used above compute a minimum over all their runs over the input of some function. In [4], a new

dual form of automata, computing a maximum over all runs of some functions, was introduced. The paper [4] was not focused on those automata by themselves, but at infinitary variants that also used asymptotic considerations. This makes the results difficult to compare (though all limitedness results can be deduced from it). However, the principle of using a dual form of automata plays a very important role in the present work.

Finally, the theory of those automata over words has been unified in [7], in which cost functions are introduced, and suitable models of automata, algebra, and logic for defining them are presented and shown equivalent. Corresponding decidability results are provided. The resulting theory is a neat extension of the standard theory of regular languages to a quantitative setting. All the limitedness problems from the literature appear as special instances of those results, as well as all the central results known for regular languages.

Besides the large number of results for distance automata and related models on words, there are also some results over trees. In [8], the *star-height problem over trees* has been solved by a reduction to the limitedness problem of nested distance desert automata over trees. The later problem was shown decidable in the more general case of alternating automata. In [9] a similar attempt has been tried for deciding the Mostowski hierarchy of non-deterministic automata over infinite trees (the hierarchy induced by the alternation of fixpoints). The authors show that it is possible to reduce this problem to the limitedness problem for a combination of nested distance desert automata with parity tree automata, while the decidability of this limitedness problem for automata on infinite trees remains open. To obtain a better understanding and to develop techniques for the case of infinite trees, we start by studying cost functions over finite trees in this paper.

Contributions

The present paper is the natural continuation of [7]. The objective in [7] was to raise the standard theory of regular languages of (finite) words to the level of cost functions. This paper aims at the same goal, but for (finite) trees.

We define in this paper two dual forms of alternating finite tree automata, namely B- and S-automata (B-automata were already used in [8]). We show that these automata are equivalent and are also equivalent to their non-deterministic variant (simulation and duality theorem). We also provide closure and decidability results for those automata.

Establishing the duality/simulation theorem is very similar to establishing the complementation [23, 11] or simulation [22] results for automata over infinite trees. Our proofs follow similar techniques, hence, we use games together with a fine analysis of the shape of strategies in our proofs. We also use new notions such as history-determinism, that have no counterpart in the classical theory.

The remainder of the paper is organised as follows. Cost functions are presented in Section 2. Cost automata over words are introduced in Section 3. Games are presented in Section 4. All results are combined in Section 5, where we study cost tree automata. Finally, all the theory is used in Section 6 for presenting cost monadic logic, an extension of monadic logic equivalent to cost tree automata.

2 Cost functions

We are interested in representing functions f over some set E assigning to each element of E a cost in $\omega + 1$, i.e., each $x \in E$ is mapped to a natural number or to the first infinite ordinal ω. When using such functions for modeling boundedness problems, the specific value of the function is not of importance. Instead, we are interested in the behavior of such functions on subsets of the domain, namely on which subsets of the domain the functions are bounded. Using this abstract view we can compare functions: A function $f : E \to \omega+1$ is below some other function $g : E \to \omega+1$, written as $f \preccurlyeq g$, if on each set $X \subseteq E$ on which g is bounded, f is also bounded. This defines a pre-order on functions from E to $\omega + 1$. The corresponding equivalence is denoted by $\approx$, i.e., $f \approx g$ if f and g are bounded on the same subsets of their domain. The equivalence classes for $\approx$ are called *cost-functions* (over E).

An alternative definition uses standard comparison of functions modulo a "stretching factor" α. Here $\alpha : \omega \to \omega$ is a non-decreasing mapping that we extend with $\alpha(\omega) = \omega$. For such α we define:

$$f \preccurlyeq_\alpha g \quad \text{iff} \quad f \leq \alpha \circ g \,.$$

It is easy to verify that $f \preccurlyeq g$ iff $f \preccurlyeq_\alpha g$ for some α. We also compare single values using $\preccurlyeq_\alpha$ with the semantics $n \preccurlyeq_\alpha m$ if $n \leq \alpha(m)$. Throughout the paper, α, α' etc denote such stretching factors, also called *correction functions*.

Cost functions over a set E can be seen as a refinement of the subsets of the base set E. Indeed, given some subset $A \subseteq E$, one defines its *characteristic function* χ_A which maps elements in A to 0, and other elements to ω. We then have for all $A, B \subseteq E$, $A \subseteq B$ iff $\chi_B \preccurlyeq \chi_A$. Consequently, all the information concerning a set is preserved in the cost function of its characteristic function: sets can be seen as particular cases of cost functions. Note also that $\chi_{A\cap B} = \max(\chi_A, \chi_B)$ and $\chi_{A\cup B} = \min(\chi_A, \chi_B)$.

3 Cost automata over words

We need a special notion of words in this work introduced in Section 3.1. In Section 3.2 we introduce objec-

tives, which are the counterpart of accepting conditions in the theory of automata over infinite objects. Word automata are presented in Section 3.3 and their history-deterministic variant in Section 3.4.

3.1 Words

It is necessary in our framework to have special symbols that identify the end of words. For this reason, a *word alphabet* $\mathbb{A} = \langle \mathbb{A}_1, \mathbb{A}_0 \rangle$ consists of two disjoint sets of letters. The letters in $\mathbb{A}_0$ are called *final letters*, as opposed to *non-final letters* in $\mathbb{A}_1$. A *word* over $\mathbb{A}$ is a sequence in $\mathbb{A}_1^*\mathbb{A}_0$, i.e., consisting of a sequence of non-final letters, and terminating with a final letter. In order to avoid confusion, an element in $\mathbb{A}_1^*$ will be called a *sequence of non-final letters*. For coding words in the usual sense, we append the final letter $\Box$ to their end.

3.2 Objectives and basic objectives

An objective is a triple $\langle \mathbb{C}, f, goal \rangle$ in which:

- $\mathbb{C}$ is a word alphabet of *actions*, letters in $\mathbb{C}_0$ are called *final actions*,
- the *value mapping* f maps $\mathbb{C}_1^*\mathbb{C}_0$ to $\omega + 1$,
- $goal \in \{\min, \max\}$ is the *goal*.

Intuitively, an objective in the context of a game is assigned to a player, and tells what is the function to optimise (f), over what domain ($\mathbb{C}_1^*\mathbb{C}_0$), and whether the player's aim is to minimise or maximise this value ($goal$). The *dual* $\overline{O}$ of an objective O is obtained by changing the goal, i.e., exchanging $\min$ for $\max$, or $\max$ for $\min$. In a game, this represents the goal of the opponent. This notion of objective will be used for games as well as for automata.

We use some basic objectives, that can be seen as the counterpart to basic acceptance conditions, like Büchi, Muller, Streett, Rabin, or parity, in the theory of automata over infinite objects (cf. [28]). The general mechanism for defining our basic objectives is to use a counter that can be incremented by one (i), reset to zero (r) or checked (c). Elements in $\{\mathsf{i}, \mathsf{r}, \mathsf{c}\}$ are called *atomic actions*. We read a sequence of atomic actions over the counter from left to right, and the counter, starting with value 0, evolves according to the actions (the action c does not change the value of the counter). From such a sequence u, one computes the set $C(u) \subseteq \omega$ which collects all the values of the counter when checked (i.e., when the action c is encountered). For example $C(\mathsf{iriiicicri}) = \{3, 4\}$ because the counter values at the two occurrences of c are 3 and 4, respectively.

This base mechanism is instantiated in different ways depending on the situation. In particular, one uses several counters and non-atomic actions (such as ic or cr). Hence, consider a finite set of *counters* Γ, and a set of actions $\mathbb{C}_1 \subseteq \{\mathsf{i}, \mathsf{r}, \mathsf{c}\}^*$ (those can be non-atomic). For each sequence u over the alphabet $\mathbb{C}_1^\Gamma$, we define $C(u)$ as $\bigcup_{\gamma \in \Gamma} C(u_\gamma)$, in which $u_\gamma \in \{\mathsf{i}, \mathsf{r}, \mathsf{c}\}^*$ is obtained by projecting u to its γ-component. In other words, each action in $\mathbb{C}_1^\Gamma$ tells for each counter what sequence of actions has to be performed, and all the values collected by all counters along a sequence of actions u are gathered into the unique set $C(u)$.

The *B-objective* (over counters Γ) is $Cost_B^\Gamma = \langle \langle \{\varepsilon, \mathsf{ic}, \mathsf{r}\}^\Gamma, \{[0], [\omega]\} \rangle, cost_B^\Gamma, \min \rangle$ in which for all $u \in (\{\varepsilon, \mathsf{ic}, \mathsf{r}\}^\Gamma)^*$ and $[x] \in \{[0], [\omega]\}$,

$$cost_B^\Gamma(u[x]) = \sup\left(C(u) \cup \{x\}\right) .$$

This corresponds to the definition of B-automata as in [7], except for the final letter in $\{[0], [\omega]\}$. This difference comes from the fact that we use here this last letter for coding accepting and rejecting states of an automaton: The letter $[0]$ should be understood as representing an *accepting state*, while the letter $[\omega]$ corresponds to a non-accepting state because it cancels all the computation before by making the value of $cost_B^\Gamma(u[\omega])$ equal to ω. The examples are given in the context of one counter. The corresponding goal is written $Cost_B^1$, the value mapping being $cost_B^1$. In this case, we simplify all the notations and use the alphabet $\{\varepsilon, \mathsf{ic}, \mathsf{r}\}$ without explicit reference to the counter.

The *S-objective* (over counters Γ) is $Cost_S^\Gamma = \langle \langle \{\varepsilon, \mathsf{i}, \mathsf{r}, \mathsf{cr}\}^\Gamma, \{[0], [\omega]\} \rangle, cost_S^\Gamma, \max \rangle$ in which for all $u \in (\{\varepsilon, \mathsf{i}, \mathsf{r}, \mathsf{cr}\}^\Gamma)^*$ and $[x] \in \{[0], [\omega]\}$,

$$cost_S^\Gamma(u[x]) = \inf\left(C(u) \cup \{x\}\right) .$$

The same comment applies to the final letters. This time, $[0]$ should be understood as *rejecting*, and the final letter $[\omega]$ as *accepting*.

For technical reasons we also define a hierarchical version of the B-objective, the *hB-objective*. In this case, the set of counters Γ is totally ordered, and one sets $H_\Gamma \subseteq \{\varepsilon, \mathsf{ic}, \mathsf{r}\}^\Gamma$ to be the set of counter actions such that whenever a counter is touched, all smaller counters are reset, i.e., $c \in H_\Gamma$ if for all $\gamma' < \gamma$, $c(\gamma) \neq \varepsilon$ implies $c(\gamma') = \mathsf{r}$. We set $Cost_{hB}^\Gamma$ to be $\langle \langle H_\Gamma, \{[0], [\omega]\} \rangle, cost_B^\Gamma, \min \rangle$. The hB-objective has particularly good properties when used in the context of games, as shown by Theorem 8 below.

3.3 Cost-automata over words

A (non-deterministic word) *cost-automaton* $\mathcal{A} = \langle Q, \mathbb{A}, I, O, \Delta_1, F \rangle$ consists of a finite set of *states* Q, a word alphabet $\mathbb{A}$, a set of *initial states* I, an objective $O = \langle \mathbb{C}, f, goal \rangle$, a set of *non-final transitions* $\Delta_1 \subseteq Q \times \mathbb{A}_1 \times \mathbb{C}_1 \times Q$, and a *final transition* mapping $F : Q \times \mathbb{A}_0 \to \mathbb{C}_0$. The set Δ_0 of final transitions is $\{(q, b, F(q, b)) \ : \ q \in$

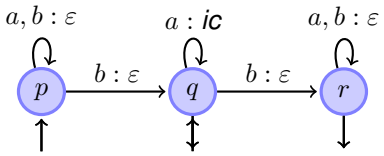
$Q,\ b \in \mathbb{A}_0\}$. It is convenient to see Δ as a word alphabet consisting of Δ_1 and Δ_0. For short, we call *B-automata* (resp. S-automata, hB-automata) the cost automata using B-objectives (resp. S-objectives, hB-objectives).

We assume that our automata are *complete* in the sense that for all states q and all letters $a \in \mathbb{A}_1$, there exists a transition of the form (q, a, c, r) in Δ_1. In the case of B-objective and S-objective, completeness can be obtained simply by adding a trap state from which every non-final transition is possible and is a loop, and only rejecting final transitions are possible (this construction is consistent with the definition of the semantics which follows).

A *run* ρ of the automaton is a word $(q_0, a_1, c_1, q_1) \dots (q_{n-1}, a_n, c_n, q_n)(q_n, b, d)$ over the alphabet Δ such that q_0 is initial. The corresponding *input word* $In(\rho)$ is $a_1 \dots a_n b$. One also says that ρ is a run over $a_1 \dots a_n b$. The *output word* $Out(\rho)$ is $c_1 \dots c_n d$. The *value* $f(\rho)$ of a run ρ is $f(Out(\rho))$. The value of a word u over $\mathbb{A}$ depends on the goal and is denoted by $[\![\mathcal{A}]\!]$:

- if $goal = \min$, then $[\![\mathcal{A}]\!](u)$ is the infimum[2] of $f(\rho)$ for all runs ρ over u,
- if $goal = \max$, then $[\![\mathcal{A}]\!](u)$ is the supremum of $f(\rho)$ for all runs ρ over u.

Example 1 *The following one counter B-automaton accepts the function* minseg*, which, to each word over the alphabet $\{a, b\}$ associates the minimal length of a maximal segment of consecutive occurrences of a. States are represented by circles, and each transition (p, a, c, q) by an edge from p to q labelled by $a : c$. Multiple transitions that differ only by the input letter, e.g., $(p, a, c, q), (p, b, c, q)$ are represented by a single edge labelled $a, b : c$. Initial states are marked by ingoing arrows.*



Our automata do not have accepting states, but a final function F. In our case, it maps $\square$ to $[0]$ for the states marked by an outgoing edge, and to $[\omega]$ otherwise. Given an input word, one constructs the optimal run as follows: the automaton guesses non-deterministically the beginning of the shortest a-segment, and jumps to state q at this moment (it can be at the beginning of the word). It then proceeds by counting the length of this interval, until it reaches the end of the word, or a letter b, in which case it goes to the trap state r.

[2]We use infimum and supremum and not min and max for handling the case when there are no runs, using $\inf \emptyset = \omega$ and $\sup \emptyset = 0$.

3.4 History-determinism

In general, the automata we consider cannot be made deterministic, even modulo $\approx$. For instance, the above Example 1 requires to guess the interval of minimal length, and this is 'unavoidable'. In replacement of determinism, we use the notion of *history-determinism* which is a semantic driven weakening of the standard (syntactic) definition of determinism.

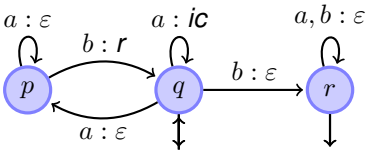
Let us fix a cost automaton $\mathcal{A} = \langle Q, \mathbb{A}, I, O, \Delta, F\rangle$. A *translation strategy*[3] for $\mathcal{A}$ is a mapping δ which maps $\mathbb{A}_1^* \times \mathbb{A}_1$ to Δ_1 and $\mathbb{A}_1^* \times \mathbb{A}_0$ to Δ_0. This mapping tells how to deterministically construct a run of $\mathcal{A}$ over a word. This map is transformed into a mapping $\tilde{\delta}$ from $\mathbb{A}_1^*$ to Δ_1^* and from $\mathbb{A}_1^*\mathbb{A}_0$ to $\Delta_1^*\Delta_0$ by $\tilde{\delta}(\varepsilon) = \varepsilon$, and $\tilde{\delta}(va) = \tilde{\delta}(v)\delta(v, a)$ for all $v \in \mathbb{A}_1^*$ and $a \in \mathbb{A}$. Given a word u over $\mathbb{A}$, if $\tilde{\delta}(u)$ is a valid run of $\mathcal{A}$ over u, it is called the run *driven by* δ over u. In the following, we assume that $\tilde{\delta}(u)$ is a valid run for every word u.

An automaton is called *history-deterministic* if there exists α and for all $n \in \omega$ a translation strategy δ_n such that for all words u,

- if $goal = \min$ and $[\![\mathcal{A}]\!](u) \leq n$, $f(\tilde{\delta}_n(u)) \leq \alpha(n)$,
- if $goal = \max$ and $[\![\mathcal{A}]\!](u) \geq \alpha(n)$, $f(\tilde{\delta}_n(u)) \geq n$.

In other words, when driven by δ, the automaton computes a function $\approx_\alpha$-equivalent to its normal semantics. The trick is that δ may require unbounded memory, and hence cannot be implemented directly by the automaton in general. One says that the automaton is α-*history-deterministic* when one wants to make the correction function α explicit.

Example 2 *The following automaton is an example of a history-deterministic B-automaton accepting the function* minseg *from Example 1:*



We use the same convention as in the previous example concerning accepting states. Let us assume that there is a run of value at most n over the word u. This means that the counter value never exceeds n. Thus the automaton was in state q with counter value 0 after some b or at the beginning of the word, then read at most n consecutive occurrences of letter a, followed by either the end of the word $\square$

[3]The name comes from a more general presentation of the notion, in which the notion of a translation strategy can be unified with the standard notion of strategy in games.

or letter b allowing it to jump to state r. This witnesses that $\mathrm{minseg}(u) \leq n$.

Conversely, assume that $\mathrm{minseg}(u) \leq n$. *We describe the translation strategy δ_n as a deterministic process for constructing the accepting run reaching r of value at most n. The sole cause of non-determinism in this automaton occurs when in state q, while reading letter a. The automaton can choose either to go to state p, and skip the remaining of the a-segment (call this choice 'skip'), or to stay in state q and increment and check the counter (choice 'continue'). There is no freedom in the definition of the translation strategy δ_n, but in this case. The translation strategy resolves this non-determinism by choosing the option 'continue' as long as possible, i.e., as long as the value of the counter is less than n, and by choosing to 'skip' only when it is unavoidable, i.e., when the counter has value n. It is clear that following this translation strategy, the counter will never exceed value n. It is also easy to see that following this translation strategy, a run will terminate in state q or r iff it contains an a-segment of length at most n.*

Theorem 3 states the equivalence of all forms of automata.

Theorem 3 (duality, Theorem 1 in [7]) *It is equivalent for a cost function to be accepted by a B-automaton, an S-automaton or an hB-automata, as well as by their history-deterministic variants.*

We call such cost functions *regular*. The proof of Theorem 3 relies on algebraic techniques and the transformations have a very high complexity. For simpler cases it is possible to provide direct constructions with better complexity, stated in the following lemmas.

Lemma 4 *The function $cost_S^\Gamma$ (resp. $cost_B^\Gamma$) is accepted by an id-history-deterministic B-automaton (resp. S-automaton) of size $2^{|\Gamma|}+1$ (whith id the identity function).*

The constructions use similar ideas as in Example 2.

Lemma 5 *The cost function $cost_B^\Gamma$ is accepted by a deterministic hB-automaton of size $|\Gamma|!$.*

The construction of the hB-automaton uses the idea of the latest appearance record construction known from the translation between acceptance conditions for ω-automata (see for instance [28]).

4 Cost games

Our results for tree automata require the use of games. Their definition is presented in Section 4.1. We show in Section 4.2 how games can be composed with history-deterministic automata, and in Section 4.3 we present results concerning the shape of winning strategies.

4.1 Definition

A cost game is a game involving two players, Adam and Eva, the result of which is a value in $\omega+1$. As opposed to what is often done, we do not partition the positions in the game into positions belonging to Eva and positions belonging to Adam. Instead we directly see the moves issued from a position as a positive Boolean combination of possible moves. One can see each such positive Boolean combination as a subgame in which Eva plays for each occurrence of a disjunction, and Adam for each conjunction. Another specificity is that actions label moves in our game (and not positions), and that a special treatment is given to final positions, i.e., positions at which the play ends. Apart from those specificities, all notions introduced here faithfully correspond to the usual ones.

From now, $\mathcal{B}^+(X)$ represents the set of positive Boolean combinations of elements in X. Given some $\varphi \in \mathcal{B}^+(X)$ and some function h from X to $\mathcal{B}^+(Y)$, $\varphi[x \leftarrow h(x)]$ represents the formula φ in which $h(x)$ has been substituted for each occurrence of x for $x \in X$. One also denotes by $\overline{\varphi}$ the *dual* of φ, i.e., φ in which disjunctions and conjunctions have been exchanged. Given $\varphi, \varphi' \in \mathcal{B}^+(X)$, $\varphi \Rightarrow \varphi'$ holds when φ' is a consequence of φ for the usual meaning.

A *cost game* $\mathcal{G} = \langle V, v_0, \delta, O\rangle$ consists of the following components:

- V is the set of *positions*.
- $v_0 \in V$ is the *initial* position.
- $O = \langle \mathbb{C}, f, goal\rangle$ is a basic *objective* (for Eva).
- $\delta : V \to \mathcal{B}^+(\mathbb{C}_1 \times V) \cup \mathbb{C}_0$ is the *control function*. The *non-final moves* in the game are the triples $(v, c, v') \in V \times \mathbb{C}_1 \times V$ such that some (c, v') appears in $\delta(v)$. M_1 is the set of non-final moves. The *final moves* are the pairs $(v, c) \in V \times \mathbb{C}_0$ such that $\delta(v) = c$. M_0 is the set of final moves. One requires that every position either has a successor in M_1, or is final. One finally assumes that the game is of finite duration, i.e., that the graph $\langle V, M_1\rangle$ does not contain any infinite path (and in particular no cycles).

The *dual* $\overline{\mathcal{G}}$ of a game $\mathcal{G}$ is obtained by dualizing the objective and the control relation. Dualization amounts to exchanging the roles of the two players. In particular, all definitions below are given for Eva, but their counterparts for Adam is obtained by dualization of the game.

As for the transitions of automata, we see M_0 and M_1 as a word alphabet. A *play* π is a word of the form $(v_0, a_1, v_1)(v_1, a_2, v_2)\dots(v_n, a_n) \in M_1^* M_0$ (in which v_0 is indeed the initial position). The *output* of the play π is $Out(\pi) = a_1 \dots a_n$. The *cost* $f(\pi)$ is $f(Out(\pi))$. A strict prefix of a play is called a *partial play*. Its output

is defined accordingly. A *strategy for Eva* σ_{E} is a set of plays such that for every partial play $\pi \in M_1^*$ ending in a position v,

$$\bigwedge\{m \in M(v) \ : \ \pi m \in \mathit{pref}(\sigma_{\mathrm{E}})\} \quad \Rightarrow \quad \delta(v)\ ,$$

in which $M(v)$ is the set of moves of origin v, $\mathit{pref}(\sigma_{\mathrm{E}}) = \{u \ : \ uv \in \sigma_{\mathrm{E}},\ v \in M_1^* M_0\}$, and $\bigwedge S$ denotes the conjunction over all elements from the set S.

When $\mathit{goal} = \min$, Eva aims at minimising over all strategies the maximum value of all plays compatible with the strategy. In other words, the value $\mathit{value}(\sigma_{\mathrm{E}})$ of a strategy for Eva σ_{E} (with respect to objective O) is defined as the supremum of $f(\pi)$ for $\pi \in \sigma_{\mathrm{E}}$, and the value of a game is the infimum of $\mathit{value}(\sigma_{\mathrm{E}})$ for σ_{E} ranging over the strategies for Eva. Dually, when $\mathit{goal} = \max$, $\mathit{value}(\sigma_{\mathrm{E}})$ is defined as the infimum of $f(\pi)$ for $\pi \in \sigma_{\mathrm{E}}$, and the value of a game is the supremum of $\mathit{value}(\sigma_{\mathrm{E}})$ for σ_{E} ranging over the strategies for Eva.

It is well known that games of finite duration are determined, i.e., that the best value that can be obtained by one player is the same as the best value which can be obtained by its opponent. In our case, this is formalised by the following proposition.

Proposition 6 *For all cost games,* $\mathit{value}(\mathcal{G}) = \mathit{value}(\overline{\mathcal{G}})$.

The two following sections give some key arguments for working with games.

4.2 History-deterministic reduction

We now show how we can compose word automata with games. The goal is to transform a game into an "equivalent" one with a different objective (as it is known from the theory of infinite games, e.g., the transformation of Muller into parity games by the latest appearance record construction, cf. [28]). For this purpose we take the product of a game with the automaton. The game outputs a word, which is read by the automaton, which in turn yields a new word, the non-determinism of the automaton being controlled by player Eva. Hence, given a game $\mathcal{G} = \langle V, \delta, \langle \mathbb{A}, f, \mathit{goal}\rangle\rangle$, and a cost automaton $\mathcal{A} = \langle Q, \mathbb{A}, I, \langle \mathbb{C}, g, \mathit{goal}\rangle, \Delta, F\rangle$, one defines the product $\mathcal{A} \times \mathcal{G} = \langle Q \times V, \delta', \langle \mathbb{C}, g, \mathit{goal}\rangle\rangle$ in which one sets $\delta'((q,v))$ to be

$$\delta(v)\left[(a,v') \leftarrow \bigvee\{(c,(q',v')) \ : \ (q,a,c,q') \in \Delta\}\right]$$

if v is non-final, and $F(\delta(v))$ otherwise.

This construction is standard. However, it is also well known that it fails to have the correct semantics in general: it is not true that, when $\mathcal{A}$ accepts the function f, the game $\mathcal{A} \times \mathcal{G}$ has the same value as $\mathcal{G}$. It is classical that this property holds either if the automaton is deterministic, or if Adam is never allowed to play in the game (in our case if all Boolean formulas are disjunctions). However, the following lemma shows that when composing with history-deterministic automata, good properties are recovered.

Lemma 7 *Let $\mathcal{A}$ be an α-history-deterministic cost automaton over alphabet $\mathbb{A}$ and $\mathcal{G} = \langle V, \delta, \langle \mathbb{A}, [\![\mathcal{A}]\!], \mathit{goal}\rangle\rangle$ be a cost game, then $\mathit{value}(\mathcal{G}) \approx_\alpha \mathit{value}(\mathcal{A} \times \mathcal{G})$.*

Proof Let $\mathcal{A} = \langle Q, \mathbb{A}, I, \langle \mathbb{C}, g, \mathit{goal}\rangle, \Delta, F\rangle$. Let us treat the case $\mathit{goal} = \min$. We claim that $\mathit{value}(\mathcal{G}) \leq \mathit{value}(\mathcal{A} \times \mathcal{G})$ (this part does not use the history-determinism of the automaton). For this, consider a strategy for Eva σ_{E} in the game $\mathcal{A} \times \mathcal{G}$. We would like to project this strategy into a strategy $\tilde{\sigma_{\mathrm{E}}}$ in the original game $\mathcal{G}$. For this, for each non-final move $m = ((q,v),c,(q',v'))$ in the game $\mathcal{A} \times \mathcal{G}$ one associates a move in $\mathcal{G}$ $\tilde{m} = (v,a,v')$ such that $(q,a,c,q') \in \Delta$, and to each final move $m = ((q,v),c)$, one associates a final move in $\mathcal{G}$ $\tilde{m} = (v,a)$ such that $F(q,a) = c$ (in both cases $\tilde{m}$ exists by definition of $\mathcal{A} \times \mathcal{G}$). One then constructs $\tilde{\sigma_{\mathrm{E}}}$ from σ_{E} by applying this transformation to each move occurring in the strategy. One can check that $\tilde{\sigma_{\mathrm{E}}}$ is a strategy for Eva in $\mathcal{G}$. Furthermore, it is straightforward that $\mathit{value}(\tilde{\sigma_{\mathrm{E}}}) \geq \mathit{value}(\sigma_{\mathrm{E}})$ because the plays in $\tilde{\sigma_{\mathrm{E}}}$ are combined with a run of $\mathcal{A}$, and the cost of plays in σ_{E} is the minimal value of a run of $\mathcal{A}$. Hence $\mathit{value}(\mathcal{G}) \leq \mathit{value}(\mathcal{A} \times \mathcal{G})$.

Conversely, let σ_{E} be a strategy for Eva in the game $\mathcal{G}$ such that $[\![\mathbb{A}]\!](\sigma_{\mathrm{E}}) = n$. Let δ_n be the translation strategy for $\mathcal{A}$. Let $\pi = (v_0,a_1,v_1)\ldots(v_k,a_k)$ be a play in σ_{E}. Let $(p_0,a_1,c_1,p_1)\ldots(p_k,a_k,c_k)$ be the run driven by δ_n over the word $a_1 \ldots a_k = \mathit{Out}(\pi)$, i.e., $\tilde{\delta}_n(\mathit{Out}(\pi))$. Define $\tilde{\pi}$ to be $((p_0,v_0),c_1,(p_1,v_1))\ldots((p_k,v_k),c_k)$, which is a play in the game $\mathcal{A} \times \mathcal{G}$. By assumption of history-determinism, one knows that $g(\tilde{\pi}) \leq \alpha([\![\mathcal{A}]\!](\pi)) \leq \alpha(n)$. Finally set $\tilde{\sigma_{\mathrm{E}}} = \{\tilde{\pi} \ : \ \pi \in \sigma_{\mathrm{E}}\}$. It is not difficult to check that $\tilde{\sigma_{\mathrm{E}}}$ is a strategy for Eva in the game $\mathcal{A} \times \mathcal{G}$. Furthermore, by the above remark, $g(\sigma_{\mathrm{E}}) \leq \alpha(n)$. Hence, overall, we have established $\mathit{value}(\mathcal{A} \times \mathcal{G}) \preccurlyeq_\alpha \mathit{value}(\mathcal{G})$.

One uses the same argument when $\mathit{goal} = \max$, replacing $\leq$ by $\geq$ and $\preccurlyeq$ by $\succcurlyeq$. □

What is really interesting in this statement is that it is possible that every winning strategy for Eva in the game $\mathcal{G}$ may require an unbounded quantity of memory (this is the case for S-games in general), while at the same time it is possible to win the game $\mathcal{A} \times \mathcal{G}$ with a bounded quantity of memory (this is the case for B-games). In this respect, this result differs a lot from the standard composition with deterministic automata used in the literature.

4.3 On the shape of strategies

Given a strategy for Eva σ_{E} in some game $\mathcal{G}$, and some $u \in \mathit{pref}(\sigma_{\mathrm{E}})$, $u^{-1}\sigma_{\mathrm{E}}$ denotes the set $\{v : uv \in \sigma_{\mathrm{E}}\}$. The strategy σ_{E} is called *positional* if for all $u, v \in \sigma_{\mathrm{E}}$ ending in the same position, $u^{-1}\sigma_{\mathrm{E}} = v^{-1}\sigma_{\mathrm{E}}$. Given a stretching function α, a game $\mathcal{G}$ is *α-positional*, if there exists a positional strategy for Eva σ_{E} in $\mathcal{G}$ such that $\mathit{value}(\sigma_{\mathrm{E}}) \approx_\alpha \mathit{value}(\mathcal{G})$. In other words, by playing positionally, Eva commits an error which is bounded by α. The following result has been established in [8] for $\mathit{Cost}^{\Gamma}_{hB}$-objectives and the argument can easily be adapted for $\overline{\mathit{Cost}^{\Gamma}_{hB}}$-objectives.

Theorem 8 *For all finite hierarchical sets of counters Γ, the $\mathit{Cost}^{\Gamma}_{hB}$- and $\overline{\mathit{Cost}^{\Gamma}_{hB}}$-games of finite duration are α-positional, in which $\alpha(n) = n^{|\Gamma|}$.*

Note finally an asymmetry here: this result does only hold for the hierarchical B-condition. Using Lemma 5 one can also show that strategies with finite memory are sufficient in B-games (where the size of the memory depends on the number of counters), whereas winning strategies in S-games may require an unbounded quantity of memory in general.

5 Cost tree automata

We introduce here our models of cost automata over trees, and study their properties.

5.1 Trees

A *ranked alphabet* $\mathbb{A}$ consists of a finite set of *letters*, each of them having a rank in ω. For $r \in \omega$, we denote by $\mathbb{A}_r$ the set of letters in $\mathbb{A}$ of rank r. Remark that this notation is compatible with the notion of word alphabet, which is equivalent to a ranked alphabet that uses only ranks 0 and 1. The set $\mathcal{T}_{\mathbb{A}}$ of *trees* over the ranked alphabet $\mathbb{A}$ is the least set containing $\mathbb{A}_0$, and such that if $t_1, \ldots, t_r$ are trees, and $a \in \mathbb{A}_r$, $a(t_1, \ldots, t_r)$ is a tree. A *position* in a tree is a sequence in ω^* such that ε is a position in every tree, and ix is a position in a tree $a(t_1, \ldots, t_r)$ iff $1 \leq i \leq r$ and x is a position in t_i. Given a position x in a tree $t = a(t_1, \ldots, t_r)$, $t(x)$ denotes the letter at position x, i.e., a when $x = \varepsilon$, and $t_i(y)$ when $x = iy$. A position such that $t(x)$ has rank 0 is called a leaf. The set of all positions of t is denoted by $\mathit{pos}(t)$.

5.2 Cost alternating tree automata

A *cost alternating tree automaton* $\mathcal{A} = \langle Q, \mathbb{A}, q_{in}, O, \delta\rangle$ consists of a finite set of states Q, a ranked alphabet $\mathbb{A}$, an *initial state* $q_{in} \in Q$, an objective $O = \langle \mathbb{C}, f, \mathit{goal}\rangle$ and a transition function $\delta : \left[\bigcup_{i>0} Q \times \mathbb{A}_i \to \mathcal{B}^+([1,i] \times \mathbb{C}_1 \times Q)\right] \cup [Q \times \mathbb{A}_0 \to \mathbb{C}_0]$, where $[1,i]$ denotes the set $\{1, \ldots, i\}$. The semantics of cost alternating automata is defined in terms of games. Given a tree t over $\mathbb{A}$, one defines the game $\mathcal{A} \times t = \langle Q \times \mathit{pos}(t), (q_{in}, \varepsilon), \delta', O\rangle$ by setting $\delta'(p, x) = \delta(p, t(x)))$ for $t(x) \in \mathbb{A}_0$, and

$$\delta'((p, x)) = \delta(p, t(x))\,[(n, c, q) \leftarrow (c, (q, xn))]$$

otherwise. One defines $[\![\mathcal{A}]\!](t)$ to be $\mathit{value}(\mathcal{A} \times t)$.

A *cost (non-deterministic) tree automaton* is a cost alternating tree automaton $\mathcal{A} = \langle Q, \mathbb{A}, q_{in}, O, \delta\rangle$ such that there exists $\Delta \subseteq \cup_{i>0} Q \times \mathbb{A}_i \times (\mathbb{C}_1 \times Q)^i$ such that for all states q and all $a \in \mathbb{A}_i$ with $i > 0$,

$$\delta(q, a) = \bigvee_{(q,a,(c_1,q_1),\ldots,(c_i,q_i))\in\Delta} \bigwedge_{n\in[1,i]} (n, c_n, q_n)\ .$$

An equivalent definition of $[\![\mathcal{A}]\!]$ that is used in the example below can be given when $\mathcal{A}$ is non-deterministic, say defined by the transition relation Δ. A *run over t* is a pair $\sigma = (r, c)$ consisting of the mapping r from $\mathit{pos}(t)$ to Q and the mapping c from $\mathit{pos}(t) \setminus \{\varepsilon\}$ to $\mathbb{C}$, and such that for all non-leaf $x \in \mathit{pos}(t)$, $(r(x), t(x), (c(x1), r(x1)), \ldots, (c(xr), r(r))) \in \Delta$, $r(\varepsilon) = q_0$, and for all leaf $x \in \mathit{pos}(t)$, $c(x) = \delta(r(x), t(x))$. Given a *branch* $x_0 < x_1 < \cdots < x_k$, i.e., a maximal sequence of positions ordered by prefix, its *cost* for σ is $f(c(x_1) \ldots c(x_k))$. The cost of a run σ is the supremum if $\mathit{goal} = \min$ (otherwise the infimum) of $f(\tau)$ when τ ranges over all branches of the tree. The value $[\![\mathcal{A}]\!](t)$ is the infimum (resp. the maximum if $\mathit{goal} = \max$) over the values of all runs over t.

Example 9 *Consider the alphabet $\mathbb{A}$ consisting of $\mathbb{A}_0 = \{a, b\}$ and $\mathbb{A}_2 = \{f\}$ (all other $\mathbb{A}_i$'s are empty). One aims at counting the number of occurrences of leaves labeled by a using a non-deterministic B-automaton. Our automaton uses two states p and q, and the following set of transitions (the $\star$ is for later reference to the transition):*

$$\Delta = \left\{ \begin{array}{l} (p, f, (\varepsilon, p), (\varepsilon, p)) \\ (q, f, (\varepsilon, q), (\varepsilon, p)) \\ (q, f, (\varepsilon, p), (\varepsilon, q)) \\ (q, f, (\mathsf{ic}, q), (\mathsf{ic}, q)) \end{array} \right\} \star \quad \left\{ \begin{array}{ll} \delta(p, a) & = [\omega] \\ \delta(p, b) & = [0] \\ \delta(q, a) & = [0] \\ \delta(q, b) & = [\omega] \end{array} \right.$$

We assume that both states are initial (this is not strictly speaking in the definition, but can be simulated in an easy way by introducing a new initial state).

This automaton is simpler to read as a bottom-up deterministic one. The objective of Eva is to minimize the maximum value over all branches. As a consequence, the state p

must necessarily be used for every b-labeled leaf, and the state q over every a-labeled leaf (otherwise the $cost^1_B$ value of the corresponding branch is immediately ω). Then, the transitions force the state p to be used if and only if the subtree rooted in the corresponding position does not contain any a-labeled leaf. Conversely, q is used iff there exist an a-labeled leaf below. Now, the cost of a branch of the run is exactly the number of occurrences of the transition $\star$. This transition is used iff state q is assumed by the run at both children. In other words, the value $S(t)$ computed by the automaton over a tree t is the maximal number of separating positions in a branch, where a separating position is a position below which both subtrees contain an a. It is easy to check that $S(t) \leq |t|_a \leq 2^{S(t)}$ where $|t|_a$ is the number of occurrences of a-leaves.

Lemma 10 *Functions computed by alternating B-automata, alternating S-automata, and alternating hB-automata are effectively equivalent.*

Proof From alternating S-automata to alternating B-automata: Consider an S-automaton $\mathcal{A}$ over counters Γ. By Lemma 4, let $\mathcal{S}$ be an id-history-deterministic B-automaton which accepts $cost^\Gamma_S$. One then easily constructs by dualizing $\mathcal{A}$ and product with $\mathcal{S}$, an automaton $\mathcal{B}$ such that for all trees t, $\mathcal{B} \times t$ is a game isomorphic to $\mathcal{S} \times \overline{(\mathcal{A} \times t)}$. We then directly get that for all trees t, $[\![\mathcal{B}]\!](t) = value(\mathcal{B} \times t) = value(\mathcal{S} \times \overline{(\mathcal{A} \times t)}) \overset{(1)}{=} value(\overline{\mathcal{A} \times t}) \overset{(2)}{=} value(\mathcal{A} \times t) = [\![\mathcal{A}]\!](t)$, where (1) is by Lemma 7 for $\alpha = id$, and (2) is by Proposition 6. The same composition principle allows similarly to go from B-automata to S-automata and from B-automata to hB-automata (using Lemma 5). □

The non-deterministic automata have different 'natural' closure properties. In particular, one uses the two new operations of $\inf$-projection and $\sup$-projection. Given two ranked alphabets $\mathbb{A}$ and $\mathbb{B}$, a translation from $\mathbb{A}$ to $\mathbb{B}$ is a mapping h from $\mathbb{A}_n$ to $\mathbb{B}_n$ for each n. It is naturally extended into a mapping $\tilde{h}$ from $\mathcal{T}_\mathbb{A}$ to $\mathcal{T}_\mathbb{B}$ by $\tilde{h}(a(t_1, \ldots, t_r)) = h(a)(\tilde{h}(t_1), \ldots, \tilde{h}(t_r))$. Given a mapping f from $\mathcal{T}_\mathbb{A}$ to $\omega + 1$, the $(\inf, h)$-*projection* of f is the mapping $f_{\inf,h}$ from $\mathcal{T}_\mathbb{B}$ to $\omega + 1$ defined by:

$$f_{\inf,h}(t) = \inf \left\{ f(t') \ : \ \tilde{h}(t') = t \right\} \quad (= \inf f(\tilde{h}^{-1}(t)))$$

The $(\sup, h)$-*projection* of f is defined similarly by:

$$f_{\sup,h}(t) = \sup \left\{ f(t') \ : \ \tilde{h}(t') = t \right\} \ (= \sup f(\tilde{h}^{-1}(t)))$$

Lemma 11 *B-automata are closed under* $\min, \max$ *and* $\inf$*-projection, hB-automata are closed under* $\min$ *and* $\inf$*-projection, and S-automata are closed under* $\min, \max$ *and* $\sup$*-projection.*

5.3 Simulation and duality result

We are now able to state and prove the main result of the paper. It shows the simulation result, i.e., that alternating automata can be transformed into equivalent non-deterministic automata, as well as the duality result, which states that non-deterministic B-automata and non-deterministic S-automata are equivalent. The proof method is inspired from modern presentations (see e.g., [28]) of similar results for automata on infinite trees: the simulation theorem of Muller and Schupp [22] and Rabin's complementation lemma [23].

Theorem 12 (simulation and duality) *It is effectively equivalent for a cost function to be accepted by a tree B-automaton, S-automaton, or hB-automaton, as well as with their alternating versions.*

Proof (sketch) By Lemma 10, alternating tree S-, B-, and hB-automata are effectively equivalent. Furthermore, hB-automata are B-automata over a restricted output alphabet. Therefore it is sufficient for us to show how to transform an alternating tree hB-automaton into (1) a non-deterministic tree hB-automaton and (2) a non-deterministic tree S-automaton. We sketch the proof of (1), the proof of (2) uses the same technique.

Consider an alternating tree hB-automaton $\mathcal{A} = \langle Q, \mathbb{A}, q_{in}, Cost_{hB}, \delta \rangle$. Given a tree t, the value $[\![\mathcal{A}]\!](t)$ is defined as the infimum over the values of all strategies σ_E for Eva in $\mathcal{A} \times t$. According to Theorem 8 it is sufficient to consider positional strategies. Now note that we can code such a positional strategy by annotating t at each inner node x with all the tuples (p, c, q, n) such that $(c, (q, xn))$ is a possible move from (p, x) according to σ_E, and similarly the leaf nodes with tuples (p, c) for the possible σ_E-moves from (p, x). Denote this annotated tree by t_{σ_E}. We construct a tree hB-automaton $\mathcal{B}$ such that $[\![\mathcal{B}]\!](t_{\sigma_\mathrm{E}}) \approx_\alpha value(\sigma_\mathrm{E})$ for some correction function α. We obtain the desired automaton by applying an $\inf$-projection to $\mathcal{B}$ defined by the mapping that removes the strategy annotations.

The construction of $\mathcal{B}$ works as follows: Consider some path τ through t_{σ_E} and define its cost to be the supremum over the costs of all σ_E-plays that stay on this path. This defines a cost function over words. It is not very difficult to see that this cost function is regular and therefore there exists a history-deterministic hB-automaton $\mathcal{D}$ computing it (according to Theorem 3). The automaton $\mathcal{B}$ is constructed by simulating $\mathcal{D}$ over all branches of the tree (in each direction $\mathcal{B}$ takes a transition that $\mathcal{D}$ could have taken when reading the corresponding path coded as a word). Since $\mathcal{D}$ is history-deterministic, we obtain that $[\![\mathcal{B}]\!](t_{\sigma_\mathrm{E}})$ is the supremum over the costs of the paths through τ computed by $\mathcal{D}$ (formally we apply Lemma 7). This corresponds to the value of σ_E, as desired. □

5.4 Decidability

In the spirit of algorithms for automata on infinite trees ([28]), we can use games to decide $\preccurlyeq$.

Theorem 13 *The relation $\preccurlyeq$ is decidable over regular cost functions of finite trees.*

In particular, the uniform universality problem (whether the function is bounded on the whole domain) is decidable, since it amounts to test whether $f \preccurlyeq 0$. This result was already known from [8] for alternating tree hB-automata.

6 Cost monadic logic

In this section, we briefly state/recall the consequences of our results in logical terms. Let us recall that monadic second-order logic (monadic logic for short) is the extension of first-order logic with the ability to quantify over sets (i.e., monadic relations). Formally monadic formulae use *first-order variables* ($x, y, \ldots$, ranging over elements of the structure), and *monadic variables* ($X, Y, \ldots$ ranging over sets of elements), existential and universal quantification over both first-order and monadic variables, boolean connectives, the membership predicate ($x \in X$), as well as all the predicates in the relational structure.

In cost monadic logic, one uses a single extra variable N of a new kind, called the *bound variable*, which ranges over non-negative integers. *Cost monadic* logic is obtained from monadic logic by allowing the extra predicate $|X| \leq N$ – in which X is some monadic variable and N is the bound variable – iff it appears *positively* in the formula (i.e., under the scope of an even number of negations). The semantics of $|X| \leq N$ is, as one may expect, to be satisfied if (the valuation of) X has cardinality at most (the valuation of) N. If we push negations to the leaves, one obtains the following syntax:

$$\begin{array}{l} \phi ::= \exists x.\phi \quad | \quad \forall x.\phi \quad | \quad \exists X.\phi \quad | \quad \forall X.\phi \\ \quad | \quad \phi \vee \phi \quad | \quad \phi \wedge \phi \quad | \quad x \in X \quad | \quad x \notin X \\ \quad | \quad R(x_1, \ldots, x_r) \quad | \quad \neg R(x_1, \ldots, x_r) \quad | \quad |X| \leq N \end{array}$$

in which $x, x_1, \ldots, x_r$ are first-order variables, X is a monadic variable, and R is some predicate symbol of arity r.

Given a sentence ϕ of cost monadic logic (i.e., with N as sole free variable), let us write $\mathcal{S}, n \models \phi$ when the formula ϕ holds over the relational structure $\mathcal{S}$ when the bound variable N takes value n. From the positivity requirement on the occurrences of the predicates $|X| \leq N$, it is clear that $\mathcal{S}, n \models \phi$ implies $\mathcal{S}, m \models \phi$ for all $m \geq n$. We use a sentences of cost monadic for defining values over structures as follows. Given a cost monadic sentence ϕ and a relational structure $\mathcal{S}$, one defines $[\![\phi]\!](\mathcal{S}) \in \omega + 1$ as follows:

$$[\![\phi]\!](\mathcal{S}) = \inf\{n \ : \ \mathcal{S}, n \models \phi\} \ .$$

Example 14 *When representing a digraph as a structure, the elements are the vertices of the digraph, and the predicate* $\mathrm{edge}(x, y)$ *expresses the existence of an edge of source x and target y. The monadic formula* $\mathtt{reach}(x, y, X)$*:*

$$\begin{array}{l} \mathtt{reach}(x, y, X) ::= \forall Z. \\ (x \in Z \wedge \forall z, z'.\ ((z \in Z \wedge z' \in X \wedge \mathrm{edge}(z, z')) \rightarrow z' \in Z)) \\ \qquad\qquad\qquad\qquad\qquad\qquad \rightarrow \quad y \in Z \end{array}$$

describes the existence of a path in a (directed) graph from vertex x to vertex y such that all vertices appearing in the path, but the first one, belong to X. Indeed, it expresses that every set Z containing x and closed under taking edges ending in X, also contains y. Consider now the following cost monadic sentence:

$$\mathtt{diameter} ::= \forall x, y. \exists X. |X| \leq N \wedge \mathtt{reach}(x, y, X).$$

It defines the diameter of a graph: the diameter of a graph is the least n such that for all pair of states x, y, there exists a set of size at most N allowing to reach y from x. Remark that the formula produces value ω if the graph is not strongly connected.

We are interested in using cost monadic logic for defining values over finite trees. In the case of trees over a ranked alphabet $\mathbb{A}$, the elements of the structure are the positions in the tree, and there is a predicate a of arity $r + 1$ for each letter a of rank r. The statement $a(x, x_1, \ldots x_r)$ holds if the letter at position x is a, and its children are, from left to right, $x_1, \ldots, x_r$.

Over (finite or infinite) words as well as (finite or infinite) trees, the expressiveness of monadic logic coincide with standard forms of automata [5, 6, 27, 23]. Those fundamental results are all established in the same way (cf. [28]). In our case, we obtain the following result.

Theorem 15 *A cost function over finite trees is regular if and only if it is definable in cost monadic logic.*

Proof From logic to automata. As in the case of monadic logic, one shows that to each connector of the logic corresponds an operation under which regular cost functions are closed. For instance, consider a cost monadic formula $\phi \vee \psi$, then one easily shows that $[\![\phi \vee \psi]\!] = \min([\![\phi]\!], [\![\psi]\!])$. It follows that disjunction corresponds to the $\min$ operation over cost functions. Pushing further this relationship, one gets that conjunction corresponds to the $\max$ operation, monadic existential quantification (and also first-order existential quantification as a particular case) corresponds to

$\inf$-projection, and universal quantification corresponds to $\sup$-projection. Concerning the constants, the only novelty compared to standard monadic logic is the predicate $|X| \leq N$. However, by definition, $[\![|X| \leq N]\!]$ evaluates to the cardinal of X. The corresponding cost function $|\cdot|_{\mathbb{B}}$ associates to each $\mathbb{A}$-tree the number of positions labelled by letters in $\mathbb{B} \subseteq \mathbb{A}$. This cost function is regular, using a slight extension of Example 9.

From automata to logic. One writes a formula guessing a run and computing its value, as usual. □

Corollary 16 *The relation $\preccurlyeq$ is decidable over cost monadic definable functions over finite trees.*

7 Conclusion

In this paper we have extended the theory of regular cost functions to the case of finite trees, showing all equivalence, closure and decidability results we could expect. The techniques involved are game-theoretic (in a way similar to the theory of languages of infinite trees), and require the use of new notions such as history-determinism. A challenging continuation would be the extension of those results to infinite trees. This would imply the decidability of the Mostowski hierarchy by [9].

References

[1] P. A. Abdulla, P. Krcál, and W. Yi. R-automata. In *CONCUR*, pages 67–81. Springer, 2008.

[2] S. Bala. Regular language matching and other decidable cases of the satisfiability problem for constraints between regular open terms. In *STACS*, volume 2996 of *Lecture Notes in Computer Science*, pages 596–607. Springer, 2004.

[3] A. Blumensath, M. Otto, and M. Weyer. Boundedness of monadic second-order formulae over finite words. In *36th ICALP*, Lecture Notes in Computer Science, pages 67–78. Springer, July 2009.

[4] M. Bojańczyk and T. Colcombet. Bounds in ω-regularity. In *LICS 06*, pages 285–296, Aug. 2006.

[5] J. R. Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik Grundl. Math.*, 6:66–92, 1960.

[6] J. R. Büchi. On a decision method in restricted second order arithmetic. In *Proceedings of the International Congress on Logic, Methodology and Philosophy of Science*, pages 1–11. Stanford Univ. Press, 1962.

[7] T. Colcombet. The theory of stabilisation monoids and regular cost functions. In *36th ICALP*, number 5556 in Lecture Notes in Computer Science, pages 139–150, Rhodos, July 2009. Springer.

[8] T. Colcombet and C. Löding. The nesting-depth of disjunctive μ-calculus for tree languages and the limitedness problem. In *CSL*, number 5213 in Lecture Notes in Computer Science, pages 416–430, Bertinoro, Sept. 2008. Springer.

[9] T. Colcombet and C. Löding. The non-deterministic Mostowski hierarchy and distance-parity automata. In *35th ICALP*, number 5126 in Lecture Notes in Computer Science, pages 398–409, Reykjavik, July 2008. Springer.

[10] L. C. Eggan. Transition graphs and the star-height of regular events. *Michigan Math. J.*, 10:385–397, 1963.

[11] Y. Gurevich and L. Harrington. Trees, automata and games. pages 60–65, 1982.

[12] K. Hashiguchi. A decision procedure for the order of regular events. *Theoretical Computer Science*, 8:69–72, 1979.

[13] K. Hashiguchi. Limitedness theorem on finite automata with distance functions. *J. Comput. Syst. Sci.*, 24(2):233–244, 1982.

[14] K. Hashiguchi. Regular languages of star height one. *Information and Control*, 53(3):199–210, 1982.

[15] K. Hashiguchi. Representation theorems on regular languages. *J. Comput. Syst. Sci.*, 27(1):101–115, 1983.

[16] K. Hashiguchi. Relative star height, star height and finite automata with distance functions. In *Formal Properties of Finite Automata and Applications*, pages 74–88, 1988.

[17] K. Hashiguchi. New upper bounds to the limitedness of distance automata. *Theor. Comput. Sci.*, 233(1–2):19–32, 2000.

[18] D. Kirsten. Desert automata and the finite substitution problem. In *STACS*, volume 2996 of *Lecture Notes in Computer Science*, pages 305–316. Springer, 2004.

[19] D. Kirsten. Distance desert automata and the star height problem. *RAIRO*, 3(39):455–509, 2005.

[20] S. C. Kleene. Representation of events in nerve nets and finite automata. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 3–42. Princeton University Press, Princeton, New Jersey, 1956.

[21] H. Leung and V. Podolskiy. The limitedness problem on distance automata: Hashiguchi's method revisited. *Theoretical Computer Science*, 310(1-3):147–158, 2004.

[22] D. Muller and P. E. Schupp. Alternating automata on infinite objects, determinacy and Rabin's theorem. In M. Nivat and D. Perrin, editors, *Automata on Infinite Words*, volume 192 of *Lecture Notes in Computer Science*, pages 100–107. Springer, 1985.

[23] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. soc.*, 141:1–35, 1969.

[24] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM J. Res. and Develop.*, 3:114–125, Apr. 1959.

[25] I. Simon. Limited subsets of a free monoid. In *FOCS*, pages 143–150. IEEE, 1978.

[26] I. Simon. On semigroups of matrices over the tropical semiring. *ITA*, 28(3-4):277–294, 1994.

[27] J. W. Thatcher and J. B. Wright. Generalized automata theory with an appliucation to a decision problem in second-order logic. *Math. Syst. Theory*, 2:57–81, 1968.

[28] W. Thomas. Languages, automata and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of language theory*, volume 3, chapter 7, pages 389–455. Springer Verlag, 1997.

[29] A. Weber. Finite-valued distance automata. *Theoretical Computer Science*, 134(1):225–251, 1994.

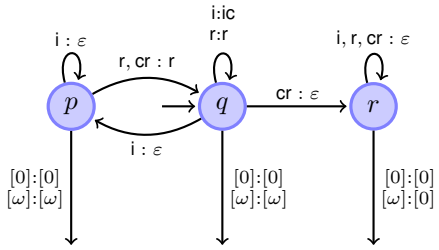


Figure 1. B-automaton $\mathcal{A}^B_{cost^1_S}$ accepting $cost^1_S$

Appendix

Proof of Lemma 4

We start by showing that the function $cost^\Gamma_S$ can be accepted by an id-history-deterministic B-automaton. The construction that we present is such that the B-automaton uses the same set Γ of counters. Such an automaton $\mathcal{A}^B_{cost^\Gamma_S}$ reads sequences over the word alphabet on which the S-objective is defined, namely the non-final letters $\{\epsilon, \mathsf{i}, \mathsf{r}, \mathsf{cr}\}^\Gamma$ and the final letters $\{[0], [\omega]\}$ such that for each word w over this alphabet we have $[\![\mathcal{A}^B_{cost^\Gamma_S}]\!](w) = cost^\Gamma_S(w)$. In fact, it would be sufficient if $[\![\mathcal{A}^B_{cost^\Gamma_S}]\!] \approx cost^\Gamma_S$ but for our construction even equality holds.

For a single counter the corresponding B-automaton is very similar to the automaton from Example 2. It is depicted in Figure 1.

Before showing that the automaton is history-deterministic let us show that it indeed accepts $cost^\Gamma_S$. For this purpose, let $u[x]$ be an input word. If $x = 0$, then $cost^1_S(u[x]) = 0$. A corresponding run of $\mathcal{A}^B_{cost^1_S}$ that has cost 0 always outputs ε for input i (moving from q to p or from p to p). The produced output sequence is of the form $\hat{u}[0]$ where $\hat{u}$ does not contain any check position. Therefore $\sup(C(\hat{u}) \cup \{0\}) = 0$.

Now consider the more interesting case that $x = \omega$ and thus $cost^1_S(u[x]) = \inf C(u)$. Each run of $\mathcal{A}^B_{cost^1_S}$ that does not end in the state r outputs $[\omega]$ at the end and therefore has cost ω. Thus, if u does not contain any check position, then all runs of $\mathcal{A}^B_{cost^1_S}$ have cost ω which is the correct value because $\inf \emptyset = \omega$.

Now assume that u does contain check positions. Consider such a check position and the block of increments before, i.e., the decomposition of u as either $u_1\mathsf{ri}\cdots\mathsf{icr}u_2$ or $\mathsf{i}\cdots\mathsf{icr}u_2$ if there is no reset before the check position under consideration.

For each such check position there is a run of $\mathcal{A}^B_{cost^1_S}$ that outputs ε on all increments in u_1 and is in state q with counter value 0 directly before the block of increments (either by moving from q to q or p to q on the reset before the block or because the block starts at the beginning of the word). Now the automaton outputs ic on each input i and moves to state r when cr is reached. The cost of this run exactly corresponds to the number of increments in the block before the check position under consideration. Note that these are all the runs that end up in state r. Thus the cost of $u[x]$ is the minimum over the costs of these runs, which is the same value as provided by $cost^1_S = \inf C(u)$.

This shows that the automaton indeed computes $cost^1_S$. It remains to show that it is history deterministic. We fix n and define a translation strategy δ_n (similar to the one from Example 2): Note that the only nondeterminism to resolve is on input i in state q. The strategy is to remain in state q until the counter value of the automaton has reached n. If another input i follows, then $\mathcal{A}^B_{cost^1_S}$ takes the transition from q to p.

Now let $u[x]$ be such that $cost^1_S(u[x]) \leq n$. If $x = 0$, then the final output of the automaton is $[0]$, and since the strategy δ_n ensures that the counter never exceeds n, we obtain that the cost of the run defined by δ_n is at most n. If $x = \omega$, then $\inf C(u) \leq n$. In particular, this means that there is a check position in u such that the block of increments before this position has length at most n. Consider the first check position with this property. The run provided by δ_n is in state q with counter value 0 directly before the block of increments is processed. Then it remains in q, and since the number of increments is no more than n, it moves to r when the check position is reached. The cost of this run is at most n.

Having settled the case of one counter, we observe that we can generalize the construction to several counters by taking one copy of $\mathcal{A}^B_{cost^1_S}$ for each counter and then taking the product of all these copies, where each copy processes the input counter instructions of the counter it is dealing with. The only thing that remains is to specify the output on the final letters. Note that the only interesting case is when some copies have reached state r but other copies are still in p or q at the end of the word, and the final letter is $[\omega]$. In fact, as soon as one of the copies has reached state r, then a small value of some counter has been checked, witnessing a small value of the input. Accordingly, we define the output for the final letters $[\omega]$ to be $[0]$ as soon as one of the copies of the automaton has reached reached state r.

The product automaton has size $3^{|\Gamma|}$, which is more than the claimed $2^{|\Gamma|} + 1$ states. The reason is that in the above construction we can merge all the states containing at least one copy of the state r. This is safe since as soon such a state is reached the final letter of the computation will be $[0]$ independently of the remaining input. Furthermore, one sequence of increments has been tested, and when applying δ_n this means a small sequence of increments has been found

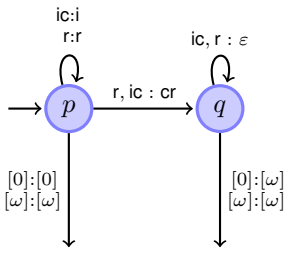


Figure 2. S-automaton $\mathcal{A}^S_{cost^1_B}$ accepting $cost^1_B$

in the input.

We now show the second part of the claim, namely that $cost^\Gamma_B$ can be accepted by an S-automaton. The idea is similar to the one presented above. In fact, the automaton is even simpler. It is shown in Figure 2 for a single counter.

Basically, the counter value of $\mathcal{A}^S_{cost^1_B}$ corresponds to the counter value of the input sequence as long as the run is in state p. Then $\mathcal{A}^S_{cost^1_B}$ can move at any time from p to q, thus producing a run with a single check whose cost is at most the cost of the input sequence. For an input word $u[\omega]$ we have $cost^1_B(u[\omega]) = \omega$. This can be achieved by $\mathcal{A}^S_{cost^1_B}$ by simply not moving to state q and thus producing a run without check positions. For $u[0]$, following the explanation above, we have runs of cost m for all counter values m that occur in u. Therefore $\mathcal{A}^S_{cost^1_B}$ computes the maximal counter value that occurs in u.

The translation strategy δ_n stays in p as long as the counter value is below n and then moves to q. One can easily check that this strategy has the desired property.

For the generalization to several counters we again take the product of $|\Gamma|$ many copies of $\mathcal{A}^S_{cost^1_B}$. The final output in the product always translates $[\omega]$ to $[\omega]$, and translates $[0]$ to $[\omega]$ if at least one of the copies has reached state q: If one copy was able to find a big sequence of increments, then this is sufficient for proving that the input has a big value. □

Proof of Lemma 5

Consider the set $\Gamma = \{1, \ldots, k\}$ of counters. The non-final letters for the B-condition are $\{\mathsf{ic}, \mathsf{r}, \varepsilon\}^\Gamma$. For simplicity, we denote the operations on counter j by ic_j and r_j.

The hB-automaton $\mathcal{A}$ that we construct uses the same set of counters with the natural ordering $1 < \cdots < k$. For better readability we denote the operations on the hierarchical counters, i.e., letters in $H_{\{1,\ldots,k\}} \subseteq \{\varepsilon, \mathsf{ic}, \mathsf{r}\}^{\{1,\ldots,k\}}$, by capital letters, i.e., by

IC_j for $j \in \{0, \ldots, k\}$ and R_j for $j \in \{1, \ldots, k\}$:

- R_j for $j \in \{0, \ldots, k\}$ maps every counter from 1 to j to r, and every counter from $j+1$ to k to ε. Remark that the case $j = 0$ corresponds to the case of all counters sent to ε.
- IC_j for $j \in \{1, \ldots, k\}$ maps every counter from 1 to $j-1$ to r, the counter j to ic, and every counter from $j+1$ to k to ε.

The construction of $\mathcal{A}$ uses the idea of latest appearance record known from the translation between acceptance condition for ω-automata (see, e.g., [28]).

- The states of $\mathcal{A}$ are permutations over the set of counters $\{1, \ldots, k\}$. The initial state is an arbitrary permutation, e.g., the identity $\langle 1, \ldots, k \rangle$.
- The final transitions map $[x]$ to $[x]$.
- To define the non-final transitions, consider a state $\langle j_1, \ldots, j_k \rangle$ and an input letter a from $\{\mathsf{ic}, \mathsf{r}, \varepsilon\}^\Gamma$. Let h be the maximal index such that $a(j_h) \neq \varepsilon$, i.e., the position of the rightmost counter in the permutation that is manipulated by a. (If no such h exists, we output ε and leave the state unchanged.) We define the transition

  $$\langle j_1, \ldots, j_k \rangle \xrightarrow{a:\mathrm{I}_h} \langle j_1, \ldots, j_k \rangle$$

  if $a(j_h) = \mathsf{ic}$, and

  $$\langle j_1, \ldots, j_k \rangle \xrightarrow{a:\mathsf{R}_h} \langle j_h, j_1, \ldots, j_{h-1}, j_{h+1}, \ldots j_k \rangle$$

  if $a(j_h) = \mathsf{r}$.

  Note that in the first case the permutation does not change, while in the second case we move j_h to the front of the permutation.

We prove that $[\![A]\!] \approx_\alpha cost^\Gamma_B$ for $\alpha(N) = kN^k$ by showing the following claim: Let $w \in \{\mathsf{ic}, \mathsf{r}, \varepsilon\}^\Gamma$ and let u be the output sequence produced by $\mathcal{A}$ for input w. Then

$$cost^\Gamma_{hB}(u) \leq cost^\Gamma_B(w) \leq k \cdot cost^\Gamma_{hB}(u)^k.$$

Assume that $cost^\Gamma_B(w) = k \cdot N^k$ and pick a subsequence v of w that contains a corresponding number of occurrences of some increment ic_j and no r_j. Consider the evolution of the position of j in the permutation (the state of $\mathcal{A}$). Because r_j does not occur in v, j can only move to the right, and this can happen at most k times. Hence, there must be a subsequence of v containing at least N^k times ic_j on which the position h of j in the permutation is stable. For each occurrence of ic_j in this subsequence the automaton outputs IC_h or a bigger increment. Since the position of j is stable, there is no output of a reset R_h or higher. A counting argument (or an induction on k) yields that at least one

counter reaches value N, i.e., $N \leq cost^{\Gamma}_{hB}(u)$. We obtain $cost^{\Gamma}_{B}(w) \leq k \cdot cost^{\Gamma}_{hB}(u)^k$.

Now assume that $cost^{\Gamma}_{hB}(u) = M$ and pick a subsequence u' of u that contains M times some increment IC_h and no higher counter operation. Since no higher counter operation occurs, the permutation is stable from position h to the right. Let j be at position h in the permutation. Each time IC_h is output by $\mathcal{A}$ there must be ic_j in the corresponding input letter of w. And since the position of j is stable in the permutation, there is no r_j occurring on this segment of the input. Thus, $cost^{\Gamma}_{hB}(u) \leq cost^{\Gamma}_{B}(w)$. □

Proof of Lemma 11 (closure of non-deterministic automata)

The proofs follow the standard approach: the minimum for B-automata, hB-automata as well as the maximum for S-automata, are obtained by taking the disjoint union of the two automata (and adding a new initial state for sticking to the definitions), while the maximum for B-automata, as well as the minimum for S-automata, are obtained by a product construction that simulates the two original automata concurrently. The inf-projections and sup-projections are obtained by simply translating the letters in the transition. We just formalize the constructions, the correctness proofs being very simple.

Let $\mathcal{A} = \langle Q, \mathbb{A}, q_{in}, O, \delta\rangle$ and $\mathcal{A}' = \langle Q', \mathbb{A}, q'_{in}, O', \delta'\rangle$ be two non-deterministic tree automata over the same ranked alphabet $\mathbb{A}$, which have objectives of same nature, i.e., either a B-objective, or an S-objective.

Consider first the case of two S-automata. Hence $O = \langle \{\varepsilon, \mathsf{ic}, \mathsf{r}\}^{\Gamma}, \{[0], [\omega]\}, cost^{\Gamma}_{B}, \min\rangle$ and $O' = \langle \{\varepsilon, \mathsf{ic}, \mathsf{r}\}^{\Gamma'}, \{[0], [\omega]\}, cost^{\Gamma'}_{B}, \min\rangle$.

Minimum of B-automata. Wlog, we assume that $\Gamma = \Gamma'$ (up to adding some useless counters to the automata when required), and by consequence $O = O'$. One constructs a non-deterministic tree B-automaton $\mathcal{B} = \langle Q \uplus Q' \uplus \{q^+_{in}\}, \mathbb{A}, q^+_{in}, O, \theta\rangle$ in which θ is defined by the transition relation:

$$\begin{aligned}\Theta &= \Delta \cup \Delta' \\ &\cup \{(q^+_{in}, a, (c_1, p_1), \dots) \,:\, (q_{in}, a, (c_1, p_1), \dots) \in \Delta\} \\ &\cup \{(q^+_{in}, a, (c_1, p_1), \dots) \,:\, (q'_{in}, a, (c_1, p_1), \dots) \in \Delta'\}\end{aligned}$$

and for each a of arity 0, each $q \in Q$, $\theta(p, a) = \delta(p, a)$, each $p \in Q'$, $\theta(p, a) = \delta'(p, a)$, and $\theta(q^+_{in}, a) = [\min(x, x')]$ in which $\delta(q_{in}, a) = [x]$ and $\delta(q'_{in}, a) = [x']$. One easily checks that $[\![\mathcal{B}]\!] = \min([\![\mathcal{A}]\!], [\![\mathcal{A}']\!])$.

Minimum of hB-automata. The same above construction applied to hB-automata yields an hB-automaton.

Maximum of B-automata. We construct a non-deterministic tree B-automaton $\mathcal{B} = \langle Q \times Q', \mathbb{A}, (q_{in}, q'_{in}), O'', \theta\rangle$ for the maximum, in which O'' is

$$\langle \{\varepsilon, \mathsf{ic}, \mathsf{r}\}^{\Gamma\uplus\Gamma'}, \{[0], [\omega]\}, cost^{\Gamma\uplus\Gamma'}_{B}, \min\rangle \,,$$

the transition relation Θ is

$$\begin{aligned}\Theta = \{((p, p'), a, (c_1c'_1, (q_1, q'_1)), \dots, (c_rc'_r, (q_r, q'_r))) \,:\, \\ (p, a, (c_1, q_1), \dots, (c_r, q_r)) \in \Delta, \\ (p', a, (c'_1, q'_1), \dots, (c'_r, q'_r)) \in \Delta'\}\end{aligned}$$

where $cc' \in \{\varepsilon, \mathsf{ic}, \mathsf{r}\}^{\Gamma\uplus\Gamma'}$ maps $\gamma \in \Gamma$ to $c(\gamma)$ and $\gamma \in \Gamma'$ to $c'(\gamma')$, and $\theta((p, p'), a) = [\max(x, x')]$ for $\delta(p, a) = [x]$ and $\delta(p', a) = [x']$. One easily checks that $[\![\mathcal{B}]\!] = \max([\![\mathcal{A}]\!], [\![\mathcal{A}']\!])$.

Inf-projection of B-automata. We start from the non-deterministic tree B-automaton $\mathcal{A}$ as above. Let h be a translation. One constructs a tree B-automaton $\langle Q, \mathbb{A}, q_{in}, O, \delta_h\rangle$ in which δ_h is defined by Δ_h over non-leaf symbols:

$$\Delta_h = \{(p, h(a), (c_1, q_1), \dots) \,:\, (p, h(a), (c_1, q_1), \dots) \in \Delta\} \,.$$

and by $\delta_h(p, b) = [\min\{x \,:\, \delta(p, a) = x,\ h(a) = b\}]$.

Inf-projection of hB-automata. The above construction applied to an hB-automaton yields an hB-automaton.

Maximum of S-automata. This is the same construction as for the minimum of B-automata. (Use S-objectives, and exchange $\min$ for $\max$ in the definition of the final transitions)

Minimum of S-automata. This is the same construction as for the maximum of B-automata. (Use S-objectives, and exchange $\max$ for $\min$ in the definition of the final transitions)

Sup-projection of S-automata. As for the inf-projection of B-automata (replace $\min$ by $\max$ in the definition of the transition function of leaf symbols).

Proof of Theorem 12 (simulation and duality)

By Lemma 10 alternating tree S-, B-, and hB-automata are effectively equivalent. Furthermore, hB-automata are special cases of B-automata. Therefore it is sufficient for us to show how to transform an alternating tree hB-automaton into (1) a non-deterministic tree hB-automaton and (2) a non-deterministic tree S-automaton. We give the proof of (1) and then explain how to adapt it for (2).

Consider an alternating tree hB-automaton $\mathcal{A} = \langle Q, \mathbb{A}, q_{in}, Cost_{hB}, \delta\rangle$. Given a tree t, the value $[\![\mathcal{A}]\!](t)$ is defined as the infimum over the values of all strategies σ_{E} for Eva in $\mathcal{A} \times t$. According to Theorem 8 it is sufficient to consider positional strategies. Now note that we can code such a positional strategy by annotating t at each inner node

x with all the tuples (p, c, q, n) such that $(c, (q, xn))$ is a possible move from (p, x) according to σ_{E}, and similarly the leave nodes with tuples (p, c) for the possible σ_{E}-moves from (p, x). Denote this annotated tree by $t_{\sigma_{\mathrm{E}}}$. Below we show how to construct a tree hB-automaton $\mathcal{B}$ such that $[\![\mathcal{B}]\!](t_{\sigma_{\mathrm{E}}}) \approx_\alpha value(\sigma_{\mathrm{E}})$ for some correction function α. We obtain the desired automaton by applying the $(\inf, h)$-projection to $\mathcal{B}$ defined by the mapping h that removes the strategy annotations.

The construction of $\mathcal{B}$ works as follows: Consider some path τ through $t_{\sigma_{\mathrm{E}}}$. Since the domains of t and $t_{\sigma_{\mathrm{E}}}$ are equal, τ is also a path through t. Recall that σ_{E} is a set of plays in the game $\mathcal{A} \times t$, and a play in $\mathcal{A} \times t$ corresponds to a path through t together with states and outputs of the automaton. Denote the set of plays in σ_{E} in which the path through t is τ by $\sigma_{\mathrm{E}}|_\tau$.

We identify τ with its ending leaf node $n_1 \cdots n_m$. The path τ together with the tree $t_{\sigma_{\mathrm{E}}}$ induces a word $w(\sigma_{\mathrm{E}}, \tau) = (a_0, n_1) \cdots (a_{m-1}, n_{m-1}) a_m$ where $a_j = t_{\sigma_{\mathrm{E}}}(n_0 \cdots n_{j-1})$.

We define a cost function f over such words by associating to each word the supremum over the costs of all σ_{E}-plays that stay on τ, i.e., $f(w(\sigma_{\mathrm{E}}, \tau)) = \sup_{\pi \in \sigma_{\mathrm{E}}|_\tau}(value(\pi))$. It is not very difficult to see that this cost function is regular: The cost of a single play staying on τ is computed by $cost_B^\Gamma$, since $\mathcal{A}$ is a B-automaton. We can construct an S-automaton guessing a play and computing its $cost_B^\Gamma$ value (Lemma 4). By the semantics of S-automata the cost of the word is the supremum over all costs of possible runs, where each run computes the cost of a play. This shows that the cost function that we consider is regular and therefore there exists a history-deterministic hB-automaton $\mathcal{D}$ computing it (according to Theorem 3).

The automaton $\mathcal{B}$ is constructed by simulating $\mathcal{D}$ over all branches of the tree: A transition of $\mathcal{D}$ is of the form $(p, (a, n), c, q)$, where a is a letter from the label alphabet for $t_{\sigma_{\mathrm{E}}}$, and n is a direction in the tree. To define the transition function of $\mathcal{B}$ we let $\delta_{\mathcal{B}}(p, a)$ be the disjunction of all possible conjunctions of the form

$$\bigwedge_{n=1}^{i} (n, c_n, q_n)$$

where $(p, (a, n), c_n, q_n)$ is a transition of $\mathcal{D}$.

This construction results in a tree hB-automaton $\mathcal{B}$. We claim that the value that is computed by $\mathcal{B}$ on $t_{\sigma_{\mathrm{E}}}$ is the supremum over all values computed by $\mathcal{D}$ on paths through $t_{\sigma_{\mathrm{E}}}$. This is the value of σ_{E} because

$$\begin{aligned} &\sup_\tau [\![\mathcal{D}]\!](w(\sigma_{\mathrm{E}}, \tau)) \\ &= \sup_\tau \sup_{\pi \in \sigma_{\mathrm{E}}|_\tau} \{value(\pi)\} \\ &= value(\sigma_{\mathrm{E}}) \end{aligned}$$

To show this property we view the construction of $\mathcal{B}$ as a composition of $\mathcal{D}$ with a game as considered in Lemma 7: The game $\mathcal{G}(t_{\sigma_{\mathrm{E}}})$ is played over the nodes of $t_{\sigma_{\mathrm{E}}}$, and Adam, starting from the root, can choose in each move a successor of the current node (Eva has no choices). The output is the label of the current node together with the direction taken by Adam. The result of such a play is a word of the form $w(\sigma_{\mathrm{E}}, \tau)$. To define the objective of the game, we let the cost of a play be the cost of $w(\sigma_{\mathrm{E}}, \tau)$ as defined above. The goal function for Eva is $\min$.

Since Eva has no choices and therefore only a single strategy, we obtain that $value(\mathcal{G}(t_{\sigma_{\mathrm{E}}}))$ is the supremum over the cost of all $w(\sigma_{\mathrm{E}}, \tau)$, which is the value of σ_{E} as explained above. Since $\mathcal{D}$ computes the cost function from the objective of the game, we can apply Lemma 7 and obtain that $value(\mathcal{G}(t_{\sigma_{\mathrm{E}}})) \approx_{\alpha'} value(\mathcal{D} \times \mathcal{G}(t_{\sigma_{\mathrm{E}}}))$ for some α'.

We note further that $\mathcal{D} \times \mathcal{G}(t_{\sigma_{\mathrm{E}}})$ is the same game as $\mathcal{B} \times t_{\sigma_{\mathrm{E}}}$, and therefore $\mathcal{B}$ computes the value of σ_{E} on $t_{\sigma_{\mathrm{E}}}$, as desired.

This ends the proof of (1), the simulation of alternating tree hB-automata by nondeterministic ones.

The proof of (2), the simulation of alternating tree hB-automata by non-deterministic tree S-automata uses exactly the same technique. The main difference is that we have to use strategies $\overline{\sigma_{\mathrm{E}}}$ for Eva in the dual game $\overline{\mathcal{A} \times t}$. The cost of a word $w(\overline{\sigma_{\mathrm{E}}}, \tau)$ is now the infimum over the cost of all plays in $\overline{\sigma_{\mathrm{E}}}|_\tau$.

As before we construct a word automaton $\mathcal{D}$ computing this cost function over the $w(\overline{\sigma_{\mathrm{E}}}, \tau)$, but this time it is a history-deterministic S-automaton. Again we run it over all branches of the tree. The resulting tree S-automaton computes the infimum over all $w(\overline{\sigma_{\mathrm{E}}}, \tau)$ for all τ. Therefore, we obtain the correct value for the strategy $\overline{\sigma_{\mathrm{E}}}$ because its value is $value(\overline{\sigma_{\mathrm{E}}}) = \inf_{\pi \in \overline{\sigma_{\mathrm{E}}}}(value(\pi))$.

Proof of Theorem 13 (decidability)

We show that we can decide $f_1 \preccurlyeq f_2$ for regular cost functions over trees. The algorithm for deciding this problem works in a similar fashion as a standard algorithm for deciding the inclusion $L_1 \subseteq L_2$ for regular languages of infinite trees. For the latter problem one starts from an automaton for L_1 and an automaton for the complement of L_2. Then one decides the emptiness of the intersection using games (see, e.g., [28]).

We use Theorem 12 to represent f_1 by an S-automaton $\mathcal{A}_1$, and f_2 by a B-automaton $\mathcal{A}_2$, and then we also rely on games, as explained in the following.

The games that we consider are similar to the cost games defined in Section 4. Unfortunately, we need slightly more general games to solve the decision problem. But we can use the same techniques, in particular Lemma 7.

We want to decide $f_1 \preccurlyeq f_2$ for a tree S-automaton $\mathcal{A}_1$ computing f_1 and a tree B-automaton $\mathcal{A}_2$ computing f_2.

We have

$$f_1 \not\preccurlyeq f_2 \Leftrightarrow \exists n \forall m \exists t \ : \ f_1(t) \geq m \text{ and } f_2(t) \leq n. \quad (1)$$

To verify the property on the right hand side, it suffices to find a number n and a family of trees $(t_j)_j$ such that $f_1(t_j) \geq j$ and $f_2(t_j) \leq n$ for all j. We do this by exhibiting a family of strategies in a game that is derived from a product automaton $\mathcal{A}$ of $\mathcal{A}_1$ and $\mathcal{A}_2$.

Let $\mathcal{A}_1 = \langle Q_1, \mathbb{A}, q^1_{in}, Cost^{\Gamma_1}_S, \delta_1 \rangle$ and $\mathcal{A}_2 = \langle Q_2, \mathbb{A}, q^2_{in}, Cost^{\Gamma_2}_B, \delta_2 \rangle$ and denote by Δ_1 and Δ_2 the respective sets of inner node transitions.

In the following, we denote by $\mathbb{S}^{\Gamma_1}$ the output alphabet of $\mathcal{A}_1$, and by $\mathbb{B}^{\Gamma_2}$ the output alphabet of $\mathcal{A}_2$.

We now apply a standard product construction to $\mathcal{A}_1$ and $\mathcal{A}_2$. The output alphabet of the product is the product of the two output alphabets. We are interested in the objectives of $\mathcal{A}_1$ and $\mathcal{A}_2$ on this product alphabet. By abuse of notation we define for a word u over $\mathbb{S}^{\Gamma_1} \times \mathbb{B}^{\Gamma_2}$

- $cost^{\Gamma_1}_S(u) = cost^{\Gamma_1}_S(pr_1(u))$, where $pr_1(u)$ denotes the projection of u to the $\mathbb{S}^{\Gamma_1}$ components, and
- $cost^{\Gamma_2}_B(u) = cost^{\Gamma_2}_B(pr_2(u))$, where $pr_2(u)$ denotes the projection of u to the $\mathbb{B}^{\Gamma_2}$ components.

The product $\mathcal{A} = \mathcal{A}_1 \otimes \mathcal{A}_2$ is the tuple $\mathcal{A} = \langle Q, \mathbb{A}, q_{in}, O_1, O_2, \delta \rangle$ with the following components:

- $Q = Q_1 \times Q_2$
- $q_{in} = (q^1_{in}, q^2_{in})$
- $O_1 = \langle \mathbb{S}^{\Gamma_1} \times \mathbb{B}^{\Gamma_2}, cost^{\Gamma_1}_S, \max \rangle$
- $O_2 = \langle \mathbb{S}^{\Gamma_1} \times \mathbb{B}^{\Gamma_2}, cost^{\Gamma_2}_B, \min \rangle$
- The transition function is defined by the set Δ containing the inner transitions

  $$((p,q), a, [(c_1,d_1),(p_1,q_1)], \dots [(c_i,d_i),(p_i,q_i)])$$

  for all $(p, a, (c_1, p_1), \dots, (c_i, p_i)) \in \Delta_1$ and $(q, a, (d_1, q_1), \dots, (d_i, q_i)) \in \Delta_2$.

Note that $\mathcal{A}$ has two objectives and thus is not an automaton in the strict sense. Nevertheless, we can adapt the definition of the game $\mathcal{A} \times t$ with the only difference that the game now has two objectives inherited from $\mathcal{A}$. A strategy σ_{E} for Eva in this game $\mathcal{A} \times t$ has two values, one according to O_1 and one according to O_2. Denote these two values by $value_1(\sigma_{\mathrm{E}})$ and $value_2(\sigma_{\mathrm{E}})$.

Since in the product $\mathcal{A}$ the two automata $\mathcal{A}_1$ and $\mathcal{A}_2$ move independently, each strategy σ_{E} in $\mathcal{A} \times t$ corresponds to a pair of strategies σ^1_{E} in $\mathcal{A}_1 \times t$ and σ^2_{E} in $\mathcal{A}_2 \times t$ such that

$$value_1(\sigma_{\mathrm{E}}) = value(\sigma^1_{\mathrm{E}}) \text{ and } value_2(\sigma_{\mathrm{E}}) = value(\sigma^2_{\mathrm{E}}).$$

We now define a game $\mathcal{G}_{\mathcal{A}}$ from $\mathcal{A}$ such that each strategy τ_{E} in this game corresponds to a tree t and a strategy σ_{E} in $\mathcal{A} \times t$. To verify (1), it will be sufficient to find a family $(\tau^j_{\mathrm{E}})_j$ of strategies in this game such that $value_1(\tau^j_{\mathrm{E}}) > j$ and $value_2(\tau^j_{\mathrm{E}}) \leq n$ for some n. The game $\mathcal{G}_{\mathcal{A}}$ corresponds to the emptiness game known from automata on infinite trees ([28]).

Let $\mathcal{G}_{\mathcal{A}} = \langle Q, q_{in}, \eta, O_1, O_2 \rangle$ where the control relation η is defined as $\eta(q) = \bigvee_{a \in \mathbb{A}} \delta(q, a)$. Note that in this definition we have final and non-final moves from the same state. This simplifies the definition and does not cause any difficulties when treating such games. In fact, in Section 4 we never use the property that a position allows only final or only non-final moves.

The difference to cost games as treated in Section 4 is that we have two objectives, and that the game is not of finite duration. The notion of strategy is adapted to this setting without any problem: in particular strategies in general form can contain infinite plays (i.e., infinite sequences of non-final moves) as it is standard in the theory of infinite games. To clearly distinguish strategies in $\mathcal{G}_{\mathcal{A}}$ and in $\mathcal{A} \times t$ we denote the strategies in $\mathcal{G}_{\mathcal{A}}$ by τ_{E}. A *finite duration strategy* is one that does not contain an infinite play.

For finite duration strategies τ_{E} we define two values $value_1(\tau_{\mathrm{E}})$ and $value_2(\tau_{\mathrm{E}})$ according to the two objectives, as in Section 4.

The game $\mathcal{G}_{\mathcal{A}}$ corresponds to a game $\mathcal{A} \times t$, where the positions of t are removed and the choice of the label $t(x)$ (used to define the control relation of $\mathcal{A} \times t$) is given to Eva. Hence, it is not surprising, that each finite duration strategy τ_{E} in $\mathcal{G}_{\mathcal{A}}$ corresponds to a tree t and a strategy σ_{E} in $\mathcal{A} \times t$, and vice versa.

Summarizing the above considerations, the following lemma reformulates (1) in terms of strategies in $\mathcal{G}_{\mathcal{A}}$.

Lemma 17 *There exists a number n and a family $(\tau^j_{\mathrm{E}})_j$ of finite duration strategies for Eva in $\mathcal{G}_{\mathcal{A}}$ with $value_1(\tau^j_{\mathrm{E}}) \geq j$ and $value_2(\tau^j_{\mathrm{E}}) \leq n$ for all j iff $f_1 \not\preccurlyeq f_2$.*

To find such a family of strategies we transform $\mathcal{G}_{\mathcal{A}}$ into a game with an ω-regular objective. Note that the strategy τ^j_{E} has to increment all the counters from Γ_1 at least j times before checking them on each play if $value_1(\tau^j_{\mathrm{E}}) \geq j$ is satisfied. On the other hand, since we want $value_2(\tau^j_{\mathrm{E}}) \leq n$, all the counters from Γ_2 have to be reset before they are incremented more than n times. For growing j the length of the plays of the strategy also has to grow to satisfy the first condition. We solve this by computing one strategy in an ω-regular game.

It turns out that the B-objective is very similar to a Streett condition but the S-objective is difficult to capture using ω-regular games. We tackle this problem by first transforming

the S-objective into a $\overline{\text{B}}$-objective over Γ_1 using the automaton $\mathcal{A}^B_{cost_S^{\Gamma_1}}$ from the proof of Lemma 4, and the construction from Lemma 7 for composing history-deterministic automata and games.

In fact, we consider the game

$$\tilde{\mathcal{G}}_{\mathcal{A}} = \overline{\mathcal{A}^B_{cost_S^{\Gamma_1}} \times \overline{\mathcal{G}_{\mathcal{A}}}}$$

where $\mathcal{A}^B_{cost_S^{\Gamma_1}}$ is only used to transform the output of $\overline{\mathcal{G}_{\mathcal{A}}}$ from the alphabet $\mathbb{S}^{\Gamma_1}$. (See Section 4 for the definition of the product of an automaton with a game.)

That is, we dualize $\mathcal{G}_{\mathcal{A}}$, thus turning O_1 into a $\overline{\text{S}}$-objective. Then we take the product with $\mathcal{A}^B_{cost_S^{\Gamma_1}}$ which transforms O_1 into a B-objective. Finally, we dualize the resulting game again and obtain a game with two objectives O_1' and O_2, where O_2 is the same B-objective as in $\mathcal{G}_{\mathcal{A}}$, and O_1' is a $\overline{\text{B}}$-objective over Γ_1.

Adapting the proof of Lemma 7 to this specific setting, we can conclude that we can transfer strategies between $\tilde{\mathcal{G}}_{\mathcal{A}}$ and $\mathcal{G}_{\mathcal{A}}$ while preserving the two values.

We can rewrite Lemma 17 as follows:

Lemma 18 *There exists a number n and a family $(\tau_{\text{E}}^j)_j$ of finite duration strategies for Eva in $\tilde{\mathcal{G}}_{\mathcal{A}}$ with $value_1(\tau_{\text{E}}^j) \geq j$ and $value_2(\tau_{\text{E}}^j) \leq n$ for all j iff $f_1 \not\preccurlyeq f_2$.*

As announced earlier, we now view $\tilde{\mathcal{G}}_{\mathcal{A}}$ as an ω-regular game. We use some standard notions and constructions from the theory of regular games of infinite duration. The reader who is not familiar with this theory is referred to [28].

Basically, we are looking for a strategy σ that ensures that all counters of Γ_2 are either reset infinitely often or incremented only finitely often, and furthermore that at least one of the counters of Γ_1 is incremented infinitely often and reset only finitely often (note that we are working with a $\overline{\text{B}}$-condition over Γ_1 now, where on each increment we also check the counter). From such a strategy we can synthesize the desired family of finite duration strategies for Eva, provided that at each point Eva can ensure to terminate the game such that the values are not "spoiled" by the final letters. This means that Eva has to ensure that she can always reach positions of the game that produce $([\omega], [0])$ as final letters.

Thus, in a first step we compute the set of all positions in $\tilde{\mathcal{G}}_{\mathcal{A}}$ from which Eva can ensure to reach a position that allows a final move with output $([\omega], [0])$. This can be done by a simple construction (called attractor construction in [28]). We remove all other positions from $\tilde{\mathcal{G}}_{\mathcal{A}}$, obtaining $\tilde{\mathcal{G}}'_{\mathcal{A}}$.

We now consider the following winning condition for outputs of infinite plays in $\tilde{\mathcal{G}}'_{\mathcal{A}}$: Eva wins an infinite play if

1. at least one counter from Γ_1 is incremented infinitely often and reset only finitely often, and
2. all counters from Γ_2 are incremented finitely often or are reset infinitely often,

Additionally, Eva loses every finite play.

Using the standard terminology of infinite games, the first condition is a Streett condition, and the second condition is a Rabin condition (see, e.g., [28]). In any case it is possible to formalize the conjunction of the two conditions as a Muller condition.

Lemma 19 *There exists a number n and a family $(\tau_{\text{E}}^j)_j$ of finite duration strategies for Eva in $\tilde{\mathcal{G}}_{\mathcal{A}}$ with $value_1(\tau_{\text{E}}^j) \geq j$ and $value_2(\tau_{\text{E}}^j) \leq n$ for all j iff Eva has a winning strategy in the Muller game $\tilde{\mathcal{G}}'_{\mathcal{A}}$.*

Proof We first show how to construct $(\tau_{\text{E}}^j)_j$ from a winning strategy σ in the Muller game $\tilde{\mathcal{G}}'_{\mathcal{A}}$. A standard result from the theory of infinite duration games says that if Eva has a winning strategy in a Muller game, then she also has a winning strategy using finite memory. Let σ be such a finite memory winning strategy for Eva.

Consider an infinite play in $\tilde{\mathcal{G}}'_{\mathcal{A}}$ according to σ. Because σ uses finite memory, there cannot be a segment in the play

- that starts and ends in the same position,
- the memory of σ is the same at the start and the end of the segment, and
- a counter from Γ_2 is incremented in this segment but not reset.

Otherwise there would be a play according to σ that loops on this segment, and in this play the counter γ would be incremented infinitely often but reset only finitely often. We can conclude that all the counters from Γ_2 are incremented at most n times before being reset for some n that does not depend on the play but only on the size of $\tilde{\mathcal{G}}'_{\mathcal{A}}$ and the memory used by σ.

We now define τ_{E}^j: In τ_{E}^j we play according to σ. From the above consideration we already know that all counters from Γ_2 stay below n. Furthermore, since at least one counter from Γ_1 is incremented infinitely often and reset only finitely often, one of these counters eventually reaches a value greater than j. Once this happens, we use the attractor strategy of Eva to reach a final position that outputs $([\omega], [0])$.

For the other direction we assume that Eva does not have a winning strategy in the Muller game $\tilde{\mathcal{G}}'_{\mathcal{A}}$. By finite memory determinacy of Muller games Adam has a winning strategy $\overline{\sigma}$ using finite memory.

Now assume that we have a family $(\tau_{\text{E}}^j)_j$ of strategies for Eva in $\tilde{\mathcal{G}}_{\mathcal{A}}$ with the desired properties. First note that all

plays according to τ_{E}^j end with final letter $([\omega],[0])$ (otherwise $value_1(\tau_{\mathrm{E}}^j) = 0$ or $value_2(\tau_{\mathrm{E}}^j) = \omega$). Therefore, all plays of τ_{E}^j stay within the sets of positions of $\tilde{\mathcal{G}}'_{\mathcal{A}}$.

Thus, we can consider the plays π_j that are produced when Eva plays according to τ_{E}^j and Adam plays according to $\overline{\sigma}$.

In π_j we call a segment that starts and ends with the same position in the game and the same memory content of $\overline{\sigma}$ a looping segment. Assume that there is a looping segment in π_j that has the following two properties:

- Some counter from Γ_1 is incremented and not reset.
- Each counter from Γ_2 is reset or not incremented.

By repeating such a looping segment Eva could force a play against $\overline{\sigma}$ in $\tilde{\mathcal{G}}'_{\mathcal{A}}$ that infinitely often increments some counter from Γ_1 without resetting it, and in which each counter from Γ_2 is either infinitely often reset or only finitely often incremented. This contradicts the choice of $\overline{\sigma}$ as a winning strategy for Adam.

Hence, on each looping segment in π_j

- each counter from Γ_1 is either reset or not incremented, or
- some counter from Γ_2 is incremented and not reset.

We refer to this property as $(\star)$

Now pick a large j and some segment β in π_j on which some counter γ from Γ_1 is incremented j times without being reset. Let k be the product of the size of $\tilde{\mathcal{G}}_{\mathcal{A}}$ and the size of the memory of $\overline{\sigma}$.

Pick a looping segment inside β that contains the largest number of increments of γ among all looping segments inside β. Such a looping segment still contains at least $\frac{j}{k}$ increments of γ.

On this looping segment some counter ζ_1 from Γ_2 is incremented and not reset by $(\star)$. By the property of τ_{E}^j this counter is incremented at most n times. Since we have chosen $j >> n$ very big, we can find a subsegment β_1 of β that still contains $j_1 = \frac{j}{kn}$ increments of γ and no increment of ζ_1. Again we find a looping subsegment of β_1 with many increments of γ, and a counter ζ_2 from Γ_2 that is incremented and not reset on this subsegment. We continue this construction $\ell = |\Gamma_2|$ times, resulting in a segment that contains $\frac{j}{(nk)^\ell}$ increments of γ but no increment of any counter from Γ_2, contradicting $(\star)$.

Hence, the family $(\tau_{\mathrm{E}}^j)_j$ cannot exists if Adam has a winning strategy in the Muller game $\tilde{\mathcal{G}}'_{\mathcal{A}}$. □

Combining Lemma 19 and Lemma 18 we obtain an algorithm for solving the problem $f_1 \preccurlyeq f_2$: We construct the Muller game $\tilde{\mathcal{G}}'_{\mathcal{A}}$ and decide if Eva has a winning strategy in this game.