Safra-like constructions for regular cost functions over finite words

Thomas Colcombet

Safra-like constructions for regular cost functions over finite words

Thomas Colcombet
thomas.colcombet@liafa.jussieu.fr

Liafa/Cnrs/Université Paris Diderot–Paris 7

Unpublished. Version Mars 2011.

Abstract. Regular cost functions have been introduced recently as an extension of the notion of regular languages with counting capabilities, which retains strong closure, equivalence, and decidability properties. Cost functions are functions ranging over $\mathbb{N} \cup \{\infty\}$, and considered modulo an equivalence which preserves the existence of bounds over each subset of the domain.
A central result in the theory of regular cost functions over words is the duality theorem stating that the two dual forms of automata, namely B- and S- automata, are equivalent. The only known proof of this result relies on the translation into an equivalent algebraic formalism.
In this paper, we describe direct translations from hierarchical B-automata to history-deterministic S-automata and from hierarchical S-automata to history deterministic B-automata. We furthermore describe how to optimize the number of counters of the output automaton, and how to make them hierarchical. This construction is reminiscent of the famous construction of Safra for the determinization of automata over infinite words.

1 Introduction

This paper aims at better understanding the theory of regular cost functions, and at providing efficient constructions and decision procedures for them.

The theory of regular languages dates from the fifties with the work of Kleene, Rabin and Scott. Recently, the notion of regular cost functions of words has been presented as a candidate for being a quantitative extension to the notion of regular languages which differs from other attempts in that it retains most of the fundamental properties of the theory of regular languages such as the strong closure properties, the equivalence with a logic (extending monadic second-order logic) as well as decidability procedures[5].

A cost function is an equivalence class of the functions from the domain (words in our case) to $\mathbb{N} \cup \{\infty\}$ for an equivalence relation $\approx$ which allows some distortion, but preserves the existence of bounds over each subset of the domain. The notion of cost functions is a refinement of the standard notion of languages. A result plays a central role in this theory: the duality theorem. It can be thought

as the counterpart to the closure under complementation for regular languages, and, when stated in its strong form, as a form of determinisation.

The goal of this paper is to provide simpler and more efficient constructions to the duality theorem in its strong form. Our constructions are reminiscent of the seminal construction of Safra for regular languages of infinite words.

Related works. Distance automata, which contain some core principles used in the theory of regular cost functions, where introduced by Hashiguchi [9]. These non-deterministic word automata describe mappings from words to $\mathbb{N} \cup \{\infty\}$, defined as the minimum over all possible accepting runs of the number of times a distinguished set of states is met along the run (∞ if there is no run). Hashiguchi proved the decidability of the existence of a bound over such functions. He used this result for solving the famous (restricted) *star height problem* [10]. This problem was raised by Eggan 25 years before, and was considered among the most important and difficult open problems in language theory. Other decision problems can also be reduced to Hashiguchi's result over words: in language theory the *finite power property*, and in model theory the *boundedness problem* for monadic formulas over words [2]. The notion of distance automata and its relationship with the tropical semiring has been the source of many investigations. Distance automata are also used in the context of databases and image compression.

Extensions and variations to the notion of distance automata have been introduced more recently. Related automata where used by Bala and Kirsten for solving the *finite substitution problem* [1, 11]. Kirsten also proposed extended automata for giving a more comprehensive and elegant proof of the star height problem [12].

More general forms of automata were used in [3] for solving an extension of monadic second-order logic over infinite words. These automata are not strictly speaking computing functions, but contain a significant part of the machinery involved in the theory of regular cost functions.

Regular cost functions. The theory of distance automata over words has been unified with the theory of regular languages in [5]. In this work, cost functions are introduced as an extension to the notion of language. Suitable models of automata, algebra, and logic for defining cost functions are presented and shown equivalent. Cost functions definable by one of these means are called *regular*. Regular cost functions also come with decidability procedures subsuming the standard result for regular languages. In particular, the equivalence is decidable, as well as the existence of a bound.

Borrowing ideas from [3], the automata manipulate counters that evolve along the run and are used for computing the output value. The automata come in two dual variants: B-automata which compute a minimum over all their accepting runs, and S-automata which compute a maximum over all their accepting runs. The central *duality theorem* states the equivalence of B-automata and S-automata. This theorem, when used for languages is a complementation result. The *strong duality theorem* further states that these automata are equivalent

to their *history-deterministic* variant: History-determinism is a weak notion of determinism which is a key ingredient in the extension of the theory to trees [7].

Regular cost functions are interesting in various contexts, and we believe in their interest in the context of verification for controlling the use of ressources by a system. However, in order to be really usable in practice, the constructions and decisions procedures must be as efficient as possible. This is not the case so far for the duality result: the proof in [5] makes a detour using algebra which costs a doubly exponential blowup (extracting the result from [3] would yield a non-elementary blow-up, and not the strong form of the result). Ideally the constructions for regular cost functions should behave exactly as standard constructions when counters are not used. The present paper is motivated by these considerations.

Contributions. In this paper, we give new direct proofs of the strong duality theorem. This contains two directions: from B-automata to history-deterministic S-automata, and from S-automata to history-deterministic B-automata. Furthermore, each of those constructions can be refined, if we are interested in reducing the number of counters and have them nested. Indeed the number of counters is a parameter of major importance in a way similar to the role of Rabin pairs/priorities in the theory of automata over infinite words and trees. This optimization furthermore produces a hierarchical automaton, i.e., an automaton for which the use of counters respects some nesting policy (similar to the role of parity condition compared to Streett/Rabin conditions over infinite words).

We gather those constructions in the following theorem (precise definitions will be given in the course of the paper):

Theorem 1. *For every B-automaton with n states and $k \geq 1$ hierarchical counters, there exists effectively an equivalent history-deterministic S-automaton of size at most $(4k(n+1))^{n+1}$ (resp., of size at most $n!(k(n+1))^{n+1}$) using at most $k2^{n-1}$ counters (resp., using kn hierarchical counters).*

For every S-automaton with n states and k hierarchical counters, there exists effectively an equivalent history-deterministic B-automaton of size at most $(4k(2^k n+1))^{2^k n+1}$ (resp., of size at most $(2^k n)!(k(2^k n+1))^{2^k n+1}$) using at most $2^{2^k n-1}$ counters (resp., using $k2^k n$ hierarchical counters).

Our constructions are inspired from the seminal construction of Safra used over infinite words [15], and of the recent state of the art improvements [14].This proof is radically different from the only existing one from [5]. Furthemore, motivated by the resemblance with Safra-like constructions, it is reasonnable to believe that those constructions are essentially optimal [8].

Structure of the paper. Regular cost functions are introduced in Section 2. In this short paper, we only present the construction in the one counter B-case. The description of this construction is the subject of Section 3. We give some examples and explanations in Section 4. We conclude in Section 5.

2 Regular cost functions

2.1 Cost functions

We are interested in representing functions f over some set E assigning to each element of E a cost in $\mathbb{N} \cup \{\infty\}$, i.e., each $x \in E$ is mapped to a natural number or to the first infinite ordinal ∞. 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 $f : E \to \mathbb{N} \cup \{\infty\}$ is below some other function $g : E \to \mathbb{N} \cup \{\infty\}$, 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 $\mathbb{N} \cup \{\infty\}$. 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 $\alpha : \mathbb{N} \to \mathbb{N}$ is a non-decreasing mapping that we extend with $\alpha(\infty) = \infty$. 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. 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[1] to ∞. 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.

2.2 Cost automata

A *cost automaton* (that can be either a *B-automaton* or an *S-automaton*) is a tuple $\langle Q, \mathbb{A}, In, Fin, \Gamma, \Delta \rangle$ in which Q is a finite set of *states*, $\mathbb{A}$ is the alphabet, In and Fin are respectively the set of *initial* and *final* states, Γ is a finite set of *counters*, and $\Delta \subseteq Q \times A \times (\{\mathtt{i}, \mathtt{r}, \mathtt{c}\}^*)^\Gamma \times Q$ is a finite set of transitions. A cost automaton is said *deterministic* if for all $p \in Q$ and $a \in \mathbb{A}$, there is at most one transition of the form (p, a, c, q).

Each counter takes *values* that are non-negative integers. Letters in $\{\mathtt{i}, \mathtt{r}, \mathtt{c}\}$ are called *atomic actions* and are used to update the counter values. The counter can be incremented by one ($\mathtt{i}$), be reset to 0 ($\mathtt{r}$), or be checked ($\mathtt{c}$). Quite naturally, given a value v, one sets $\mathtt{i}(v) = v + 1$, $\mathtt{r}(v) = 0$ and $\mathtt{c}(v) = v$ (the use of check is made explicit below). This definition is extended to sequence of atomic actions by $\varepsilon(v) = v$ and $\zeta\zeta'(v) = \zeta'(\zeta(v))$ (actions are applied from left

[1] The choice $0/\infty$ in this definition is arbitrary.

to right). Such sequences are called *actions*. A *valuation of the counters* is a mapping from Γ to $\mathbb{N}$. One uses composed actions $c \in (\{\mathtt{i}, \mathtt{r}, \mathtt{c}\}^*)^\Gamma$ for updating the valuations of the counters: $c(V) = V'$ in which $V'(\gamma) = c(\gamma)(V(\gamma))$. The *initial* valuation of the counters is constant equal to 0.

An *run* σ of an automaton over a word $a_1 \dots a_n$ is a sequence $(q_0, V_0), \dots, (q_n, V_n)$ in which V_0 is the initial valuation for all $i = 1 \dots n$, there exists a transition (q_{i-1}, a_i, c_i, q_i) such that $V_i = c_i(V_{i-1})$. A run is accepting if q_0 is initial and q_n is final.

The difference between B-automata and S-automata comes from their dual semantics. An *n-run* of a B-automaton $\mathcal{A}$ for some $n \in \mathbb{N}$ is a run σ such that all counters have value at most n when checked. An *n-run* of an S-automaton $\mathcal{A}$ for some $n \in \mathbb{N}$ is a run σ such that all counters have value at least n when checked. We define for all B-automata $\mathcal{B}$, all S-automata $\mathcal{S}$, and all words u:

$$[\![\mathcal{B}]\!]_B(u) = \inf\{n \ : \ \text{there exists an accepting } n\text{-run of } \mathcal{B} \text{ over } u\} \ ,$$
$$\text{and} \quad [\![\mathcal{S}]\!]_S(u) = \sup\{n \ : \ \text{there exists an accepting } n\text{-run of } \mathcal{S} \text{ over } u\} \ .$$

(There is a slight ambiguity here since the definition of an n-run depends also on whether we consider the B-semantics or the S-semantics. In practice, this is always clear from the context.)

A B-automaton of counters Γ is *simple* if the actions it uses belong to $\{\varepsilon, \mathtt{ic}, \mathtt{r}\}^\Gamma$. The above example is of this form. An S-automaton of counters Γ is *simple* if the actions that can be performed on counters belong to $\{\varepsilon, \mathtt{i}, \mathtt{r}, \mathtt{cr}\}^\Gamma$. Once more, these automata, even in their simple form, happen to be also equivalent to the general model. However, in general, the automata we consider cannot be made deterministic, even modulo $\approx$. In replacement of determinism, we use the notion of *history-determinism* which is a semantics driven weakening of the standard (syntactic) definition of determinism.

Let us fix a cost automaton $\mathcal{A} = \langle Q, \mathbb{A}, I, \Delta, F\rangle$. A *translation strategy*[2] for $\mathcal{A}$ is a mapping δ which maps $\mathbb{A}^* \times \mathbb{A}$ to Δ. 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 $\mathbb{A}^*$ to Δ^* by $\tilde{\delta}(\varepsilon) = \varepsilon$, and $\tilde{\delta}(va) = \tilde{\delta}(v)\delta(v, a)$ for all $v \in \mathbb{A}^*$ 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 always assume that $\tilde{\delta}(u)$ is a valid run for every word u.

Given a B-automaton $\mathcal{B}$, an S-automaton $\mathcal{S}$, and families of translation strategies for them $\delta^{\mathcal{B}} = (\delta^{\mathcal{B}}_n)_{n\in\omega}$ and $\delta^{\mathcal{S}} = (\delta^{\mathcal{S}}_n)_{n\in\omega}$, one defines the *semantics of $\mathcal{B}$ and $\mathcal{S}$ driven by $\delta^{\mathcal{B}}$ and $\delta^{\mathcal{S}}$* respectively by:

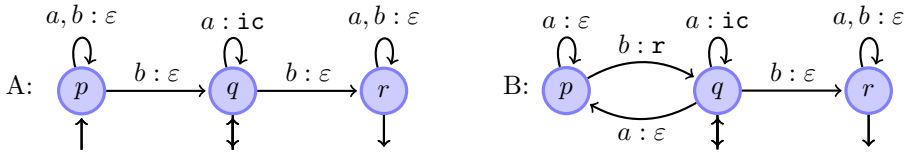
$$[\![\mathcal{B}]\!]^{\delta^{\mathcal{B}}}_B(u) = \inf\{n \ : \ \tilde{\delta}^{\mathcal{B}}_n(u) \text{ is an accepting } n\text{-run of } \mathcal{B}\}$$
$$\text{and} \quad [\![\mathcal{S}]\!]^{\delta^{\mathcal{S}}}_S(u) = \sup\{n \ : \ \tilde{\delta}^{\mathcal{S}}_n(u) \text{ is an accepting } n\text{-run of } \mathcal{S}\} \ .$$

A B-automaton (resp S-automaton) $\mathcal{A}$ is *history-deterministic* if there exists a familiy of translation strategies $(\delta_n)_{n\in\omega}$ for it such that $[\![\mathcal{A}]\!]_B \approx [\![\mathcal{A}]\!]^{\delta}_B$ (resp.

[2] 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.

$[\![\mathcal{A}]\!]_S \approx [\![\mathcal{A}]\!]_S^\delta$). In other words, the automaton is not-deterministic, but can be deterministically driven while still computing the same cost function.

Example 1. The B-automaton A below computes the function minseg which to $a^{n_0}b \dots ba^{n_k}$ associates $\min(n_0, \dots, n_k)$. Each transition (p, a, ζ, q) is depicted by an edge from p to q labeled $a : \zeta$ (the automaton uses only one counter). Initial states use ingoing arrows, and final states outgoing arrows.



The automaton A is not history-deterministic, since it requires to know when in state p while reading letter b, if we are entering the segment of a of minimal size. This is future-dependant and hence cannot be implemented by a translation strategy. One can also prove that no determinstic B-automaton can accept the function minseg, even modulo $\approx$.

However, the B-automaton B accepts the same function, but is history-deterministic. 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 letters a, followed by either the end of the word or the letter b allowing it to jump to state r. This witnesses that $\text{minseg}(u) \leq n$.

Conversely, fix a natural number n. We describe a translation strategy δ_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 2 (strong duality [5]). *It is effectively equivalent for a regular cost function over words to be accepted by a B-automaton or an S-automaton as well as by their simple history-deterministic variant.*

The only known proof of this result makes use of the equivalent algebraic characterisation by means of stabilisation monoids [5, 4]. We describe in this paper a direct automata-theoretic construction.

3 The construction for B-automata with one counter

For simplicity, we just describe the one-counter B-automaton case in this paper. We take as input a one-counter B-automaton, and aim at producing an equivalent history-deterministic S-automaton. The principle of the construction is to keep track of certain possible runs of the automaton, and to preserve a sufficient amount of information concerning them, and in particular concerning the possible values of the counter. It is not possible to maintain the exact value of the counter, but, since we are interested only in cost functions, i.e., in working modulo $\approx$, it is sufficient to maintain an 'estimation' of the values. For estimating the possible values of the counter for all runs we are tracking, two techniques are used. The first one consists in arranging the states in a tree-like structure, *à la Safra*, and the second is to heavily use the counters of the resulting automaton.

We now fix ourselves a one-counter B-automaton $\mathcal{A} = \langle Q, \mathbb{A}, I, F, \{\gamma\}, \Delta\rangle$ (in fact, we simplify all notations concerning γ). As in the original Safra construction, the automaton that we construct maintains its information arranged in a tree like fashion. We start by making precise the terminology used for trees. It can be good to keep an eye at the same time on the examples in the following section for having illustrations of most aspects of the construction.

A *tree* T is a prefix-closed subset of $\mathbb{N}^*$ [3] such that whenever $u(i+1)$ belong to T, ui also belongs to T. The elements of the tree are called *nodes*. A *forest* T is a tree with its root removed, and the *roots* of the forests are the nodes of length 1. A node ui is called a *child* of u, while u is called the *parent* of ui. The *arity* of a node is the number of its children. Nodes of the form ui and uj with $i \neq j$ are called *siblings*. Given a node ui, its *successor sibling* is the node $u(i+1)$ when it exists. Conversely, the *predecessor sibling* of $u(i+1)$ is ui. A node of the form uv with v nonempty is said *below* u, while u is said *above* uv. The relation $\leq_{\text{rlex}}$ denotes the lexicographic ordering of nodes in which longer words (i.e., below in the tree) are considered as smaller. The strict variant is $<_{\text{rlex}}$.

A node x is said to *dominate* a node y if y is a right sibling of x, or y is below a right sibling of x. Remark that the domination relation is transitive: if x dominates y and y dominates z, then x dominates z. Given a position x which is not leftmost, i.e., of the form $ui0^*$ with $i \geq 1$, its *domination predecessor* is the position $u(i-1)$. In other words, it is the rightmost node which dominates x.

A *pre-structure* S (over the states Q) is a forest together with a mapping from its nodes to subsets of Q. One sees S as a mapping from a forest to sets of states. Given a node x, one says that p *appears* at node x if $p \in S(x)$. A very informal intuition is that the pre-structure containing p keeps track of a possible run of the original automaton that would end in state p, and the node in which p appears gives some information on the value of the counter at the end of this run. A *structure* is a pre-structure such that furthermore:

1. Every state appears at most once in the structure, and;
2. No node is empty (i.e., has no state appearing in it).

[3] Where $\mathbb{N}^*$ denotes the set of words over the alphabet of natural numbers.

(so far, the construction is undistinguishable from Safra's one).

We construct the S-automaton $\mathcal{S}$ which has the following characteristics:

- the states are structures,
- the initial state is the structure consisting of a single node labeled with I,
- a state is final in $\mathcal{S}$ if no final state from $\mathcal{A}$ does appear in it,
- the counters are the positions of nodes in structures.
- the transition relation is described below.

Remark that counters are attached to nodes of the structure. In fact, at any moment in the construction, only the counters that are attached to an existing node (i.e., a node present in the (pre-)structure) are relevant. That is why, whenever one *destroys a node* during the construction, it implicitly means that the corresponding counter is reset. This ensures that whenever a new node is (re-)created later, its associated counter has the default value 0.

One needs now to define the transition relation of our automaton. When reading letter a from a structure S, the possible transitions of the automaton are described as the composition of three successive steps.

The first step is the *update step*. It consists in starting from a structure S, reading a letter a, and outputing a new pre-structure S'. At the beginning, the pre-structure S' has the same domain as S, but all its nodes are empty. It is then constructed using the following rules for all nodes x of S and all states $p \in S(x)$:

U1 For all transitions (p, a, ε, q), one inserts q in $S'(x)$.

U2 For all transitions $(p, a, \mathtt{ic}, q)$, one creates a new rightmost child y of x and inserts q in $S'(y)$.

U3 For all transitions $(p, a, \mathtt{r}, q)$, one creates a new rightmost tree to the forest which consists of an isolated node y and inserts q in $S'(y)$.

We reach at this point a pre-structure. Call it again S. The second step, the *normalization step* aims at making it a valid structure. It consists in the application of a sequence of atomic normalization steps. The three possible steps are:

N1 If $p \in S(x)$ and $p \in S(y)$ for some $x <_{\text{rlex}} y$, then p is removed from $S(x)$. This operation is called a *preference simplification*.

N2 If the node x as well as all nodes below are empty, one performs an *horizontal collapse at node* x:
- All nodes at x, below x, and dominated by x are destroyed. Let P be the set of states appearing in these nodes before being destroyed.
- If P is nonempty, a new node is created at position x and labeled P.

N3 If there are no states at x but there are states appearing below x, one performs an *increment collapse at node* x; it means that:
- The counter x is incremented.
- All nodes below x are destroyed. Let P be the set of states appearing at these nodes before being destroyed. The label of node x is set to P.
- An horizontal collapse of the right sibling of x (if relevant) is performed.

After sufficiently many normalization steps, the pre-structure eventually becomes a structure, and no more normalization steps are possible.

During the third step, called the *reduction step*, the automaton can choose non-deterministically to reduce some nodes. The *reduction of a node* x involves the successive application of the following transformations:

R1 The counter x is checked (this means requiring that it is sufficiently large).

R2 One removes all states appearing at x or below. Let P be these states.

R3 If x has a domination predecessor y, one inserts all nodes in P at node y. We call this an *advance of the states in* P.

If x has no domination predecessor, we say that this is the *death* of the states in P (indeed, these states disappear from the structure).

R4 One performs an horizontal collapse at x.

A reduction of a node transforms a structure into another structure.

The correctness result concerning this construction is the following:

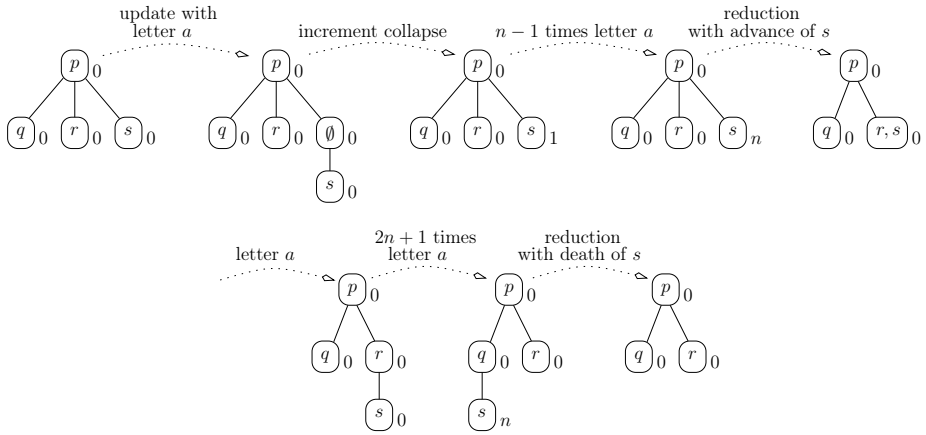
Theorem 3. *The automaton $\mathcal{S}$ is history-deterministic and is $\approx$-equivalent to $\mathcal{A}$.*

Though the construction is reminiscent of the construction of Safra, the proof of correctness is very different, and significantly more involved. One reason is that we must be very precise on the values of counters, while explaining Safra's construction can be done at a higher level. The other reason is that the construction is more complex and involves more steps of different nature. In this short paper, we will just illustrate the behavior of the construction (see appendix for the proofs and more general constructions).

4 Some examples and explanations

We consider here distance automata, i.e., one counter B-automata, without resets of the counter. Consider a distance automaton $\mathcal{A}$ with states p, q, r, s, and transitions (p, a, ε, p), (q, a, ε, q), (r, a, ε, r), and $(s, a, \mathtt{ic}, s)$. Let $\mathcal{S}$ be the automaton resulting from the construction of the above section.

As our first example, we present a possible run of $\mathcal{S}$ reading successively $3n-2$ times the letter a. For the sake of explanation, this run starts from an arbitrary configuration. This run is an n-run since the reductions are only performed on nodes for which the counter value is at least n. Each configuration of $\mathcal{S}$ being the pair of a structure (a tree), together with a valuation of counters: we write the value counters by directly annotating the nodes. We allow ourselves to present intermediate (pre-)configurations for the sake of illustration:



In this example, the state s starts at the right of the structure, and progressively moves toward the left while the run advances. This phenomenon is captured by the general (informal) principle of this construction that when a state s appears in some node x in a fixed structure then:

> 'the least is x for $\leq_{\text{rlex}}$, the higher is the value of the counter of the 'best' run of $\mathcal{A}$ reaching this state (after reading the same word)'

This principle explains the death of state s at the end of the run: The automaton $\mathcal{S}$ has detected that all runs ending in state s after $3n+2$ letters a need have seen a lot of increments. The death of state s means that it is not required to keep track of state s after that point. It is forgotten.

This principle also explains the preference simplification rule (N1): if two different runs reach the same state, then the semantic of a B-automaton is to keep track only of the one having the least counter value. According to the above principle, one chooses to keep the state occurring at the maximal position for $\leq_{\text{rlex}}$. This is the content of rule N1 (that we don't illustrate further).

Another point concerning this construction is that it is (non-deterministic and) history-deterministic. You can remark that the only non-determinism in the construction concerns the choice of the nodes to reduce. In the above example, we have made the choice to immediately reduce a node as soon as the corresponding counter has reached value n. The automaton could have chosen to wait longer before deciding to reduce the node. This choice would not be as good. In particular, the run would not manage to get rid of state s (i.e., reaching its death) after $3n+2$ occurrences of letter a. (Recall the automaton $\mathcal{S}$ tries to reach an 'accepting state', i.e., it tries to have as few states as possible in the structure of the current configuration. Hence, it aims into bringing as much states as possible to their death). In the proof of correctness, we define the translation strategy to be exactly this one: the strategy is to "always reduce as soon as possible", i.e., as soon as the counter has reached value n. we call this choice the *optimal translation strategy*. It resolves all non-deterministic choices in the best way (up to $\approx$).

Another important feature of this construction is that it works only modulo $\approx$. One can have an idea of the origin of this approximation by having a glimpse on the proof of correctness. In the following explanations, a word u is fixed, σ always denotes a run over u starting in an initial state of $\mathcal{A}$ and ending in a configuration $(p, v) \in Q \times \mathbb{N}$, and similarly Σ always denotes a run over u of $\mathcal{S}$ starting in an initial state, and ending in a configuration (S, V).

The first direction of the proof states that "no run of $\mathcal{A}$ can be erroneously forgotten by $\mathcal{S}$". This is formally stated as:

> *If Σ is any n-run, and p does not appear in S,*
> *then there are no $n-1$-runs σ ending in state p.*

The converse direction states that "if a state of $\mathcal{A}$ is kept track of by $\mathcal{S}$, then there is a reason for this." This is formalized by:

> *If Σ is the n-run driven by the optimal strategy, and p appears in S,*
> *then there is a $P(n)$-run σ ending in state p, where P is a polynomial depending on $|Q|$.*

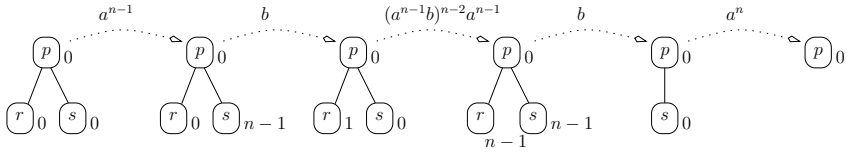
The combination of the two direction yields Theorem 3.

One sees from these statements that some precision is lost. Indeed, in the first direction, the n-runs of Σ proves the non-existence of $n-1$-runs of $\mathcal{A}$. In the other direction, a (specific) n-run of Σ proves the existence of a $P(n)$-run of $\mathcal{A}$. However, the combination of the two directions does not assert anything on the existence of an m-run σ for $n \leq m < P(n)$. This "gray zone" is the source of the approximation $\approx$. The two above directions can be rephrased as:

$$[\![\mathcal{S}]\!]_S \leq [\![\mathcal{A}]\!]_B \leq P \circ [\![\mathcal{S}]\!]_S^{\texttt{optimal-strategy}} ,$$

which proves both that $[\![\mathcal{S}]\!]_S \approx [\![\mathcal{A}]\!]_B$ and that $\mathcal{S}$ is history deterministic.

According to the first example, it may seem that it is sufficient to take P to be linear. This is not the case as shown by the following run over the word $(a^{n-1}b)^n a^n$:



where the automaton $\mathcal{A}$ is the one of the previous example expanded with transitions (p, b, ε, p), $(r, b, \texttt{ic}, r)$, and (s, b, ε, s). One sees on this example that the automaton reads n^2 times the letter a before reaching the death of state s. This shows that the polynomial P needs to be at least quadratic. In practice, for being able to *amortize* each such phenomena, the polynomial P has a degree linear in $|Q|$.

5 Conclusion

We have presented in this short paper the core construction, which retains most of the ideas. Our more general results in Theorem 1 involve generalizing to several counters, optimizing the number of counters, making them hierarchical, as well as providing the other direction, from S to B (see appendix for proofs). From those results, we can derive new proofs of older results such as the PSPACE decidability of limitedness for distance automata [13], and for nested distance desert automata (i.e., hierarchical B-automata) [12]. Considering the analogy with the standard Safra construction, it is also reasonable to believe that those constructions are (essentially) optimal.

Ackowledgments

I am grateful to Nathanaël Fijalkov, Christof Löding, Sylvain Perifel, Adeline Pierrot and Dominique Rossin for helpful discussions and comments during the development of this work.

References

1. Sebastian 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.
2. Achim Blumensath, Martin Otto, and Mark Weyer. Boundedness of monadic second-order formulae over finite words. In *36th ICALP*, Lecture Notes in Computer Science, pages 67–78. Springer, July 2009.
3. Mikolaj Bojańczyk and Thomas Colcombet. Bounds in ω-regularity. In *LICS 06*, pages 285–296, 2006.
4. Thomas Colcombet. Regular cost functions, part I: logic and algebra over words. submitted, 2009.
5. Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *36th ICALP*, number 5556 in Lecture Notes in Computer Science, pages 139–150. Springer, 2009.
6. Thomas Colcombet and Christof 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. Springer, 2008.
7. Thomas Colcombet and Christof Löding. Regular cost functions over finite trees. In *LICS*, pages 70–79, 2010.
8. Thomas Colcombet and Konrad Zdanowski. A tight lower bound for determinization of transition labeled büchi automata. In *36th ICALP*, number 5556 in Lecture Notes in Computer Science, pages 151–162. Springer, 2009.
9. Kosaburo Hashiguchi. Regular languages of star height one. *Information and Control*, 53(3):199–210, 1982.
10. Kosaburo Hashiguchi. Relative star height, star height and finite automata with distance functions. In *Formal Properties of Finite Automata and Applications*, pages 74–88, 1988.
11. Daniel Kirsten. Desert automata and the finite substitution problem. In *STACS*, volume 2996 of *Lecture Notes in Computer Science*, pages 305–316. Springer, 2004.

12. Daniel Kirsten. Distance desert automata and the star height problem. *RAIRO*, 3(39):455–509, 2005.
13. Hing Leung. Limitedness theorem on finite automata with distance functions: An algebraic proof. *Theoretical Computer Science*, 81(1):137–145, 1991.
14. Nir Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. *Logical Methods in Computer Science*, 3(3), 2007.
15. Shmuel Safra. On the complexity of ω-automata. In *29th FOCS*, pages 319–327. IEEE, 1988.

A Hierarchical and full automata

We give here some complementary definitions for the appendix. Namely, we introduce the formal definitions of hierarchical automata. We also introduce the notion of full automata.

Hierarchical automata. An automaton is called *hierarchical* if its set of counters Γ is $\{1, \dots, K\}$, and whenever an action modifying a counter is performed on some counter, i.e., $\mathtt{r}$ pr $\mathtt{i}$, then all counters below are reset. Hierarchical simple B-automata corresponds to the nested distance desert automata introduced by Kirsten[12].

The definition of (simple) *hierarchical B-automata* (*hB-automata* for short) enforces that there are only $2K + 1$ way to perform action on counters.

- For $l = 1 \dots, K$, the action $\mathtt{IC}_l$ increments and checks counter l, and resets all counters $1, \dots, l-1$,
- For $l = 0 \dots, K$, action $\mathtt{R}_l$ resets all counters almong $1, \dots, l$. Remark that in the case $l = 0$, then no action is performed on any counter, and in this case, we legitimately simply write ε instead of $\mathtt{R}_0$.

Let us denote $\{\mathtt{R}_0, \mathtt{IC}_1, \mathtt{R}_1, \dots, \mathtt{IC}_K, \mathtt{R}_K\}$ simply as $\mathtt{Act}_K^{\mathtt{hB}}$. Remark that a full automaton can be safely normalized by removing every transition of the form (p, ζ, q) whenever there exists another transition (p, ζ', q) such that $\zeta \sqsubset \zeta'$ for the order:

$$\mathtt{IC}_K \sqsubset \mathtt{IC}_{K-1} \sqsubset \cdots \sqsubset \mathtt{IC}_1 \sqsubset \varepsilon \sqsubset \mathtt{R}_1 \sqsubset \cdots \sqsubset \mathtt{R}_K \ .$$

This simplification operation does not change the cost function accepted by the automaton. The resulting automata then have at most one transition of the form (p, ζ, q) for each pair of states p, q. Thus such automata can be represented as $Q \times Q$ matrices over $\mathtt{Act}_K^{\mathtt{hB}} \cup \{\bot\}$, in which $\bot$ is a new symbol representing the absence of a transition.

In a similar way, (simple) *hierarchical S-automata* (*hS-automata* for short) uses only $3K + 1$ actions on counters, namely:

- For $l = 1 \dots, K$, the action $\mathtt{I}_l$ increments counter l, and resets all counters $1, \dots, l-1$,
- For $l = 0 \dots, K$, action $\mathtt{R}_l$ resets all counters almong $1, \dots, l$. Remark that in the case $l = 0$, then no action is performed on any counter, and in this case, we legitimately simply write ε instead of $\mathtt{R}_0$.
- For $l = 1 \dots, K$, action $\mathtt{c}_l$ checks counter l, and leaves all counters invariants.

One denotes $\{\mathtt{R}_0, \mathtt{IC}_1, \mathtt{c}_1, \mathtt{R}_1, \dots, \mathtt{IC}_K, \mathtt{c}_K, \mathtt{R}_K\}$ simply as $\mathtt{Act}_K^{\mathtt{hS}}$. This is not strictly speaking a simple automaton as defined above, but this definition is more general, and simplify a bit the construction by not having a combined 'check-reset' action.

We described in the hB-case some optimization/normalization arguments, based on a good total order on the possible actions on counters. No such order exists in the hS-case. Several complications of the construction below are indirectly resulting from this fact.

Remark that any B-automaton is equivalent to a hierarchical hB-automaton using the same number of counter [7], as well as each S-automaton is equivalent to a hierarchicak hS-automaton using the same number of counters. This is a simple adaptation of the standard latest appearance record technique, used in the context of regular cost-functions.

Full automaton. A *full automaton* is an automaton over a special alphabet of letters, namely the alphabet which contains all sets of possible transitions as letters. It has the property that it is in some sense 'the most complex' automaton of his size. In our case, if one is able to perform the translation from a B-automaton to an S-automaton starting from the full automaton, then this construction can be used for all automata using the same set of states, and the same number of counters.

In the case of a hierarchical B-automata, the alphabet contains each set

$$\mathcal{P}(Q \times \mathtt{Act}_K^{\mathtt{hB}} \times Q)$$

as a letter. The transition function is then defined to implement the content of the letter:

$$\Delta = \{(p, A, \zeta, q) \ : \ (a, \zeta, q) \in A\} \ .$$

One denotes by $\mathcal{F}^B_{Q,K}$ the full simple hierarchical B-automaton over states Q and using K-counters (the initial and final states are not fixed). We use the notation $\mathcal{F}^B_{Q,K,I,F}$ in order to make the initial and final states explicit. We use the similar notations $\mathcal{F}^S_{Q,K}$ and $\mathcal{F}^S_{Q,K,I,F}$ for the full S-automaton.

B The general construction from B to S

In this section, we provide the complete construction from hierarchical B-automata to S-automata, i.e., the generalization of the construction given in the main part of the paper, that we call the *one-counter construction*. The construction with multiple counters is a nested variant of this one-counter construction.

The transformation of a K-counter hierarchical B-automaton is done by induction on K. If $K = 0$, a simple power-set construction will do the job.

For a K-counter hierarchical B-automaton, However, each node of the structure, instead of being labeled with a set of states, is labeled with a state of the construction for $K - 1$ counters. (In fact, the sets of states labeling nodes in the one-counter construction correspond to the standard power-set construction, which happens to be the 0-counter case).

Hence, a major difference concerns the rule U1: the update step is different. It has to process all transitions that leave counter K-invariant. For this, the new rule U1 performs one step of execution of the construction for $K - 1$ counters. The only subtlety is that some states may die during this step. The new rule U1 has to catch those dying states, and reinsert them in the domination predecessor of the node (if possible, otherwise the states really die). Of course, this also makes the correctness proof more complex. We proceed we a precise description of this construction.

Structure of the construction In this section, we describe, given a B-automaton, how to construct an equivalent history-deterministic S-automaton. The remaining of the section is devoted to the proof of correction of this construction, and to the optimization of the use of counters.

Essentially, our construction is described as follows.

Input A finite set of states Q and a non-negative integer K (the number of counters).

Output A *rich S-automaton for* Q, K denoted $\mathcal{D}^{BS}_{Q,K}$. A rich S-automaton for Q, K is a complete S-automaton which has no initial states nor final states, and is enriched with the following mappings $\texttt{content}, \texttt{init}$, and $\texttt{insert}$:

- a mapping $\texttt{content}$ from $Q_{\mathcal{D}^{BS}_{Q,K}}$ to subsets of Q: $\texttt{content}(S)$ informally represents the set of states that S is keeping track of. To ease reading, we just say that p *appears in* S instead of $p \in \texttt{content}(S)$.
  In general, the idea is that a state of $\mathcal{D}^{BS}_{Q,K}$ consists of its content, organized in a structure that maintains extra information about the possible runs that have reached this state. One can think about the construction of Safra: the set of states appearing in a Safra-tree is exactly the set of reachable states, but further organized in a tree-structure maintaining extra information concerning the Büchi condition.
  In our case, the content may not contain all reachable states. Indeed, some states may be 'removed' from the content because the construction has detected that every run that reaches such states need have seen a lot of increments in a row. Hence, the states in the content are the ones which are reachable, and such that the construction has not detected yet that a big value of some counter has been checked. Conversely, the states which do not belong to the content are either unreachable for $\mathcal{A}$, or reachable but via runs of provably too high cost.
- a mapping $\texttt{init}$ from $\mathcal{P}(Q)$ to $Q_{\mathcal{D}^{BS}_{Q,K}}$: the state $\texttt{init}(P)$ represents the initial state of $\mathcal{D}^{BS}_{Q,K}$ to be used if one wants to keep track of the runs of initial states in P.
- a mapping $\texttt{insert}$ from $Q_{\mathcal{D}^{BS}_{Q,K}} \times \mathcal{P}(Q)$ to $Q_{\mathcal{D}^{BS}_{Q,K}}$: $\texttt{insert}(S, P)$ is defined for $P \cap \texttt{content}(S) = \emptyset$, and it represents a state which has the same behavior as S, but furthermore keeps track of the states in P.

The intention is that a rich automaton is a standard S-automaton, that the mapping $\texttt{init}$ allows to construct 'parametric' sets of initial states, $\texttt{content}$ allows to define the final states, and $\texttt{insert}$ is a new capability given to the the automaton: the ability to handle 'on-the-fly new initial states'.

The reason for this presentation is that the automaton $\mathcal{D}^{BS}_{Q,K}$ is constructed using an induction on the number of counters. When adding a counter, one uses the construction of the previous level to create 'sub-processes' in the construction (typically for keeping track of pieces of run in which the new counter is not touched). The construction requires that we are able to initialize those sup-processes (mapping $\texttt{init}$), we are able to inspect their content (mapping $\texttt{content}$), and we are able to insert on-the-fly new states that need to be kept track of (mapping $\texttt{insert}$).

Given a B-automaton with states Q, initial states I and final states F (we will use from now the automaton $\mathcal{F}^B_{Q,K,I,F}$), one constructs the S-automaton $\mathcal{D}^{BS}_{Q,K,I,F}$ that has the following characteristics:

- the structure of the automaton is $\mathcal{D}^{BS}_{Q,K}$,
- the sole initial state is $\texttt{init}(I)$,
- the final states are the states $S \in Q_{\mathcal{D}^{BS}_{Q,K}}$ such that $F \cap \texttt{content}(S) = \emptyset$.

Our goal is to prove that this automaton is history-deterministic, and equivalent to $\mathcal{F}^B_{Q,K,I,F}$.

One can informally understand this construction as follows. The automaton is initialized in order to keep track of every run of $\mathcal{F}^B_{Q,K,I,F}$ which starts with some state from I, and a run is accepting if it contains no final states: i.e., the construction has detected that every run starting in an initial state and ending in an final state need have a high cost.

The validity of the construction is to establish the correctness of the above. This is an involved work which requires the use of subtle properties to be preserved. In this section, we just describe the construction with some simple intuitions. The exact properties needed for the proof will be presented upon need in the subsequent sections. This construction is a refinement of the one presented in the main part of the paper.

From now, we fix a finite set of states Q. We construct, by induction on a non-negative integer K, a rich S-automaton $\mathcal{D}^{BS}_{Q,K}$ for Q, K.

Construction: the base case. The base case consists in transforming a no-counter B-automaton into an equivalent history-deterministic S-automaton. Recall that the B-automata with no counters accept exactly the characteristic function of the underlying non-deterministic language automaton. For S-automata, the language is the characteristic function of the complement of the language accepted by the underlying language automaton. What it means is that transforming a 0-counter B-automaton into an equivalent 0-counter S-automaton amounts to perform a complement of the underlying non-deterministic language automaton. That is why our construction performs a subset construction as for non-deterministic language automata. Since the subset construction determinizes the automaton, we also immediately get the history-determinism.

We construct the S-automaton $\mathcal{D}^{BS}_{Q,0}$ with the following characteristics:

- The set of states is $\mathcal{P}(Q)$,
- There are no counters,
- The (deterministic) transition relation is

$$\{(P, A, \varepsilon, \{q \ : \ p \in P, \ (p, \varepsilon, q) \in A\})\} \ .$$

- For S a state of $\mathcal{D}^{BS}_{Q,0}$, $\texttt{content}(S) = S$,
- For $P \subseteq Q$, $\texttt{init}(P) = P$,
- For S, P such that $\texttt{content}(S) \cap P = \emptyset$, $\texttt{insert}(S, P) = S \cup P$.

We prove below that this construction satisfies all the properties required. One can still already see that this is exactly the standard power-set construction. This also shows more concrete examples of the mappings `content`, `init`, and `insert`. One sees how this is the standard determinization construction, enriched with a mapping `content` allowing to inspect the states that are kept track of by the construction, enriched of a mapping `init` that allows initialization, and enriched of a mapping `insert` allowing on-the-fly insertion of new states.

Construction: the inductive case. We describe now the construction for K counters, for $K \geq 1$. By the induction hypothesis, we assume the existence of an automaton $\mathcal{D}^{BS}_{Q,K-1}$, as well as of the mappings $\texttt{content}^{Q,K-1}$, $\texttt{init}^{Q,K-1}$ and $\texttt{insert}^{Q,K-1}$.

A *pre-structure* S (over the states Q) is a forest together with a mapping from its nodes to the states of $\mathcal{D}^{BS}_{Q,K-1}$. One writes S as a mapping from nodes to states of $\mathcal{D}^{BS}_{Q,K-1}$. The *content* of the pre-structure $\texttt{content}(S)$ is the union of the content of its nodes, i.e., a state *appears* in S if it appears in one of the $S(x)$ for x a node. One also writes $p \in S(x)$ for simplicity. A pre-structure is a *structure* if furthermore every state appears at most in one node of the pre-structure, and no node has an empty content.

The essential idea of the construction is that a (pre-)structure maintains the states in Q that could have been reached without seeing a big value of the counter. Those states are organized in a tree-like structure which attempts to relate the positions of a state p in this tree (and the counters) to the best possible value of the counter reachable at this state. The key principle is that, the smallest is the position at which a state occurs for the order $\leq_{\text{rlex}}$, the higher the corresponding counter value it has. Unfortunately, this relationship has to be stated informally since working modulo $\approx$ means that exact values are not relevant. The much more involved proof will make this relationship more formal. Despite this, it is good keep in mind the following principle:

The less is the position for $\leq_{\text{rlex}}$, the higher is the value of the counter.

Note that there are only finitely many structures. A *real position* is a position which corresponds to a node in some structure. A *relevant position* is a position which occurs in some pre-structure which may occur during the description of the construction. There are infinitely many positions, but there are only finitely many real positions and relevant positions. By definition, every real position is relevant.

We construct the *S-automaton* $\mathcal{D}^{BS}_{Q,K}$ which has the following characteristics:

- The states are structures.
- The counters are either of the form x for x a possible node of some structure, or of the form (x, γ) for x node of some comparison structure and γ a counter of $\mathcal{D}^{BS}_{Q,K-1}$.
  The counters of the form x are the counters used for maintaining sufficient information concerning the counter numbered K of $\mathcal{A}$ (as in the construction described in the main part of the paper).

Counters of the form (x, γ) are used for the induction hypothesis. Each node x of the structure can be seen as running a sub-process for one counter less, and the counters of the form (x, γ) are used as the internal counters used by the sub-process runned at node x.
Given a valuation V and a node x, $V|_x$ denotes the valuation which to γ associates $V((x, \gamma))$, i.e., a valuation of the counters of $\mathcal{D}^{BS}_{Q,K-1}$. The intention is that $(S(x), V|_x)$ will be a meaningful configuration of the automaton $\mathcal{D}^{BS}_{Q,K-1}$.

- The transition relation–the difficult part–is described below.
- $\texttt{content}(S)$ is $\bigcup_{x \text{ node of } S} \texttt{content}^{Q,K-1}(S(x))$,
- $\texttt{init}(P)$ is the empty forest if $P = \emptyset$, otherwise, it is the forest consisting of a single node labeled with $\texttt{init}^{Q,K-1}(P)$,
- $\texttt{insert}(S, P)$ for P such that $\texttt{content}(S) \cap P = \emptyset$ is the forest obtained by adding a rightmost root to S, and initializing it to $\texttt{init}^{Q,K-1}(P)$.

Remark that counters are attached to nodes of the structure. This makes that there are finitely many of them. In fact, at any moment in the construction, only the counters that are attached to a real node (i.e., a node present in the structure) are relevant. That is why, whenever one *destroys a node* during the construction, it implicitly means that the corresponding counters are reset. This ensures that whenever a new node is (re-)created later, its associated counter has the default value 0.

One needs now to define the *transition relation* of our automaton. We describe the result of reading a letter A when in a structure S. The possible transitions are described as the composition of three successive steps. Each step contains a certain quantity of non-determinism. During those steps, actions on the counters are performed. Those actions are supposed to be collected during the construction without further explanations.

Beforehand, we need to define what it means to execute a letter on a node. Indeed, one principle of this construction is to use each node of the structure as the state of a sub-process executing $\mathcal{D}^{BS}_{Q,K-1}$ and keeping track of a reduced number of runs. The global construction is in charge of dispatching the runs between those sub-processes. This is why, a basic operation required in our description is to be able to execute one step of $\mathcal{D}^{BS}_{Q,K-1}$ inside a node. This is the purpose of the following definition. Let x be a node of a pre-structure, and A be a letter of the full alphabet for $K-1$ counters, *executing A on node x* consists in:

EX One non-deterministically chooses a transition $(S(x), A, t, S')$ of $\mathcal{D}^{BS}_{Q,K-1}$ (there is such a transition since $\mathcal{D}^{BS}_{Q,K-1}$ is assumed to be complete). Then, one sets the label of node x to be S'. Furthermore, for all actions appearing in t on a counter γ, one performs the same action on the counter (x, γ).
A state q is said to *die* during the operation if there is some $p \in \texttt{content}(S(x))$ such that $(p, t, q) \in A$ but $q \notin \texttt{content}(S')$. This situation occurs when the construction has locally detected that all runs ending in state q have reached a high value of some counter (of course, this concerns only the runs that this node is taking care of).

Let D be the set of states which died, then one create a new rightmost child y of x, labeled with $\texttt{init}^{Q,K-1}(D)$.

Variant: We sometime apply this construction with the letter

$$Rem_p = \{(q, \varepsilon, q) \ : \ q \neq p\}$$

for a given state $p \in Q$. This should be thought as destroying the state p from $S(x)$. Performing this action is called *removing state p from $S(x)$*

Remark 1: The new rightmost child y of x has a meaning. This is a node which is inserted just before x for the order $\leq_{\text{rlex}}$. Indeed, $y <_{\text{rlex}} x$, and for all $z <_{\text{rlex}} x$ occurring in S, $z <_{\text{rlex}} y$.

Remark 2: You can remark that no states ever dies if $\mathcal{D}^{BS}_{Q,K-1}$ correspond to the base case, i.e., if $K = 1$. Hence, this part of the construction is not needed for the case of distance automata which corresponds to $K = 1$, and there is nothing similar in the main part of this paper. For $K = 1$, this operation also happens to be deterministic.

We can now turn to the definition of transitions of our automaton. It consists on the successive application of three steps: the update step, which updates the structure according to the letter read by the automaton, yielding a pre-structure, followed by the normalization step, which turns this pre-structure back into a structure, and followed finally by the reduction step, in which the automaton inspects the values of counters (using the check action) and can take advantage of it for removing some states from the content (this is an advantage since removing states will decrease the `content` of the state, and this will make the run of the resulting S-automaton more likely to be accepting).

The first step is the update step. It consists in starting from a structure S, reading a letter A, and outputting a new pre-structure S'. Since we are working with the full alphabet over states Q, K, the letter A can be uniquely decomposed as the union $B \cup A_K$, in which B is a letter of the full alphabet for $K-1$ counters, and A_K is a subset of $Q \times \{\texttt{IC}_K, \texttt{R}_K\} \times Q$. The *update step by letter A* consists in performing the following operations:

U1 For all nodes x of S, one executes B on the node x (see above rule EX).

U2 For all tuples $(p, \texttt{IC}_K, q)$ in A_K such that p appears in $S(x)$, one creates a new rightmost child y of x labeled with $\texttt{init}^{Q,K-1}(q)$.

U3 For all tuple $(p, \texttt{R}_K, q)$ in A_K such that p appears in some $S(x)$, one creates a new rightmost root y in S labeled with $\texttt{init}^{Q,K-1}(q)$.

We reach at this point a pre-comparison structure. Call it again S. The second step, the *normalization step* aims at making it a valid structure. It consists in the application of a sequence of atomic normalization operations. The three possible operations are:

N1 If p appears in both $S(x)$ and $S(y)$ for some nodes $x <_{\text{rlex}} y$, then one removes p from $S(x)$ (see the special case of rule EX). This operation is called a *preference simplification step.*

N2 If the node x as well as all nodes below have an empty content, one performs an *horizontal collapse at node x*:
- All nodes at x, below x, and dominated by x are destroyed (thus, by convention, all corresponding counters are reset). Let $P \subseteq Q$ be the set of states appearing in these nodes before being destroyed.
- If P is nonempty, a new node is created at position x and $S(x)$ is set to $\texttt{init}^{Q,K-1}(P)$.

N3 If there are no states appearing at x, but there are states appearing below x, one performs an *increment collapse at node x*. It consists in:
- Incrementing counter x.
- Destroying all nodes below x (and resetting the corresponding counters). Let P be the set of states contained in these nodes before destruction.
- The label of node x is set to $\texttt{init}^{Q,K-1}(P)$.
- If x has a right sibling y, then one performs an horizontal collapse at y.

After sufficiently many normalization steps, the pre-comparison structure eventually becomes a comparison structure, and no more normalization steps are possible.

During the third step, called the *reduction step*, the automaton can choose non-deterministically to reduce some nodes. The first operation when reducing a node is to check the corresponding counter. This is the reason why the automaton should wait that the value of the counter is sufficiently high before choosing to reduce a node. This non-deterministic choice is the sole real cause of non-determinism of the construction (either directly, or indirectly when the non-determinism is inherited by the induction hypothesis using the rule EX).

The *reduction of a node x* involves the successive application of the following operations:

R1 The counter x is checked. Since one goal of an S-automaton is to avoid checking small values of counters, this amount to require that the counter x has a sufficiently high value.

R2 Let P be the states contained at x or below. One makes the node at x and below empty.

R3 If x has a domination predecessor y, one inserts all states in P at node y, i.e., one replaces $S(y)$ by $\texttt{insert}^{Q,K-1}(S(y), P)$. We call this an *advance of the states in P (from x to y)*.

If x has no domination predecessor, we say that this is the *death* of the states in P, and the states are not reinserted in the structure. This means that the `content` does not contain any states from P anymore. For this reason, the states from P will be considered as dying if the rule EX is used (during the next step of the construction). To this respect, the terminology is consistent.

R4 One performs an horizontal collapse at x.

One can check that the reduction of a node transforms a structure into a structure.

The validity of the construction is the subject of the following Lemma 1.

Lemma 1. *For all finite set Q, all $I, F \subseteq Q$, and all non-negative integer K, the S-automaton $\mathcal{D}^{BS}_{Q,K,I,F}$ is history-deterministic and $[\![\mathcal{F}^{B}_{Q,K,I,F}]\!]_B \approx [\![\mathcal{D}^{BS}_{Q,K,I,F}]\!]_S$.*

In the two following section, Sections B.1 and B.2, we establish successively each of the domination relations involved in Lemma 1, as well as the history determinism property. Indeed, Lemma 1 directly follows from Corollaries 1 and 2.

B.1 Proof of Lemma 1, first direction

In this section, we prove half of Lemma 1. This proof is involved, reflecting the complexity of the construction. From very far, this direction of the proof establishes:

> *If Σ is a n-run of $\mathcal{D}^{BS}_{Q,K,I,F}$ ending in S, V, and p does not appear in S, then there are no $(n-1)$-runs σ of $\mathcal{F}^{B}_{Q,K,I,F}$ ending in state p.*

where runs are assumed to start in an initial state.

The main objects in the proof will be configurations. Each pair (p, v) denotes a configuration of the full automaton $\mathcal{F}^{B}_{Q,K}$. Namely, it consists of a state $p \in Q$, and of a valuation v mapping each counter among $1, \ldots, K$ to its current value. Each pair (S, V) denotes a "pre-configuration" of the automaton $\mathcal{D}^{BS}_{Q,K}$. It consists of a pre-structure S, together with a valuation V of the counters involved in S, i.e., mapping each counter from S to a non-negative integer. However, as we mentioned earlier, by construction all counters corresponding to positions not in S can be considered as existing, of default value 0. Remark that in general the pairs (S, V) are not configurations of the automaton $\mathcal{D}^{BS}_{Q,K}$ since S may not be a structure. However, it is important in our proof to refer to pre-structures, and hence to pre-cofigurations, and for steps of the construction (such as update step/normalization/reduction).

When we say below that *one can go from (S, V) to (S', V') reading letter A*, we mean that there is a transition of $\mathcal{D}^{BS}_{Q,K}$, reading letter A, which goes from structure S to structure S', does not check a counter of value less than n, and such that the application of the actions in the transition transforms the valuation of the counters V into the valuation V'. We also use this notion for pre-structures.

We prove below.

Lemma 2. *If there exists an accepting n-run of $\mathcal{D}^{BS}_{Q,K,I,F}$ over a word u, then there exists no accepting $(n-1)$-run of $\mathcal{F}^{B}_{Q,K,I,F}$ over u.*

From which we get:

Corollary 1. $[\![\mathcal{D}^{BS}_{Q,K,I,F}]\!]_S \leq [\![\mathcal{F}^{B}_{Q,K,I,F}]\!]_B$.

Proof. Indeed, the left part of the implication of Lemma 2 states that $[\![\mathcal{D}^{BS}_{Q,K,I,F}]\!]_S(u) \geq n$, while the right part states that $[\![\mathcal{F}^{B}_{Q,K,I,F}]\!]_B(u) \geq n$. In particular we get the corollary for $n = [\![\mathcal{D}^{BS}_{Q,K,I,F}]\!]_S(u)$. □

The remaining of the section is devoted to the proof of Lemma 2. From now, *we fix a non-negative integer* n, the one involved in Lemma 2.

The statement involves runs of both $\mathcal{A}$ and $\mathcal{D}^{BS}_{Q,K}$. The main difficulty in the proof is to find the correct property which relates the configurations of the two runs: this is the goal of property $\diamondsuit_n$. We define below what it is for a configuration (p, v) and a pre-configuration (S, V) to satisfy property $\diamondsuit_n$. However, since the definitnion of $\diamondsuit_n$ is obtained by induction on the number of counters, we formalize beforehand the inductive hypothesis, i.e., the behavioral properties we will prove for $\diamondsuit_n$:

P1 For all $p \in P$, $(p, 0), (\texttt{init}(P), 0)$ satisfy property $\diamondsuit_n$,

P2 If there is a transition from (p, v) to (p', v') in A such that no counter reaches value n, and there is a transition of $\mathcal{D}^{BS}_{Q,K}$ from (S, V) to (S', V') reading letter A, then

$$(p, v), (S, V) \text{ satisfy } \diamondsuit_n \text{ implies } (p', v'), (S', V') \text{ satisfy } \diamondsuit_n.$$

P3 For $P \cap \texttt{content}(S) = \emptyset$,

$$(p, v), (S, V) \text{ satisfy } \diamondsuit_n \text{ imples } (p, v), (\texttt{insert}(S, P), V) \text{ satisfy } \diamondsuit_n.$$

P4 If $P \cap \texttt{content}(S) = \emptyset$ and $p \in P$ then $(p, m), (\texttt{insert}(S, P), V)$ satisfy $\diamondsuit_n$ for all m.

P5 If $(p, v), (S, V)$ satisfy $\diamondsuit_n$ then p appears in S.

Assuming P1,. . . ,P5, we can prove Lemma 2.

Proof (of Lemma 2). Consider an $(n-1)$-run σ of $\mathcal{F}^{B}_{Q,K}$ over some word u,

$$\sigma = (p_0, v_0), \ldots, (p_m, v_m) \ ,$$

and an n-run Σ of $\mathcal{D}^{BS}_{Q,K}$ over the same word u,

$$\Sigma = (S_0, V_0), \ldots, (S_m, V_m) \ .$$

By P1, $(p_0, v_0), (S_0, V_0)$ satisfy $\diamondsuit_n$. Then, by inductive application of P2 we get that $(p_i, v_i), (S_i, V_i)$ satisfy $\diamondsuit_n$ for all $i = 0 \ldots m$. It follows by P5 that p_m appears in S_m, i.e., $p_m \in \texttt{content}(S_m)$. Assume now both p_m and S_m would be accepting, then $p_m \in \texttt{content}(S_m) \cap F = \emptyset$ (definition of the final states of $\mathcal{D}^{BS}_{Q,K,I,F}$). This is a contradiction. □

One can remark that P3 and P4 were never used in the argument. This is because P3 and P4 are used solely for the induction to go through: the construction for K-counters involve the on-the-fly insertion of states in the construction of level $K-1$, i.e; the use of the '`insert`' capability.

The base case: no counters. For the base case, there are no counters involved. Two configurations $(p, 0), (S, 0)$ are said to satisfy *property* $\diamondsuit_n$ if $p \in S$ (recall that in this case, a state of $\mathcal{D}^{BS}_{Q,K}$ is simply a set of states of $\mathcal{A}$). We have:

Lemma 3. *Properties P1,P2,P3,P4 and P5 hold for the base construction.*

Proof. Straightforward. □

Property $\diamondsuit_n$, the inductive case. Given a configuration (p, v) of $\mathcal{F}^B_{Q,K}$, and a pre-configuration (S, V) of $\mathcal{D}^{BS}_{Q,K}$ one says that $(p, v), (S, V)$ satisfy *property* $\diamondsuit_n$ if there exist $x \leq_{\text{rlex}} y$ nodes of S such that:

$\diamondsuit_n.0$ [4] $v(l) < n$ for all $l = 1 \dots K$;
$\diamondsuit_n.1$ p appears in $S(y)$;
$\diamondsuit_n.2$ $v(K) \geq V[x] \stackrel{\text{def}}{=} \sum_{\varepsilon \neq z \sqsubseteq x} (V(z) + 1) - 1 \quad \left(= \sum_{\varepsilon \neq z \sqsubseteq x} V(z) + |x| - 1 \right)$;
$\diamondsuit_n.3$ if $x = y$, $(p, v|_{<K}), (S(y), V|_y)$ satisfy property $\diamondsuit_n$ (for $K-1$ counters) in which $v|_{<K}$ represents the valuation v restricted to the counters $1, \dots, K-1$ (this part is irrelevant for the one-counter case).

The positions x, y making the above properties true are called the *witnesses* of property $\diamondsuit_n$ for $(p, v), (S, V)$. One says that $(p, v), x, (S, V)$ satisfy property $\diamondsuit_n$, or even better that $(p, v), x, y, (S, V)$ satisfy property $\diamondsuit_n$ if we want to make explicit the witnesses for property $\diamondsuit_n$.

Lemma 4 (P1). *Let $P \subseteq S$, and $p \in P$, then $(p, 0), (\texttt{init}(P), 0)$ have property $\diamondsuit_n$.*

Proof. We take $x = y = 0$ as witnesses of $\diamondsuit_n$.

$\diamondsuit_n.1$ Since $p \in P$, we have $p \in \texttt{init}^{Q,K-1}(P) = \texttt{init}(P)(y)$.
$\diamondsuit_n.2$ $v(K) = 0 \geq 0 = V[x]$,
$\diamondsuit_n.3$ We indeed have $x = y$. One applies the induction hypothesis, and get that $(p, 0), (\texttt{init}(P), 0)$ have property $\diamondsuit_n$, which means that $(p, 0), (S(y), 0)$ satisfy property $\diamondsuit_n$.

□

In what follows we see the different operations (update, preference simplification, horizontal collapse, increment collapse, and reduction) as transformations of configurations. I.e., each such operation can be seen as a relation mapping (S, V) to (S', V') in which S, S' are pre-structures, and V, V' are valuations of the counters of $\mathcal{D}^{BS}_{Q,K}$. We prove that each such operation preserves the property $\diamondsuit_n$. The gathering of those properties yields property P2. All the proofs are by induction on the number of counters. So, from now, we assume that properties P1,...,P5 hold for $K-1$ counters. Since we intend to establish property $\diamondsuit_n$, we assume all values of the counters occurring in configurations of $\mathcal{F}^B_{Q,K}$ to be at most $n-1$.

Lemma 5. *If one goes from (S, V) to (S', V') by an update step of letter A, $(p, \zeta, q) \in A$, $(p, v), (S, V)$ satisfy property $\diamondsuit_n$, and all counters in $\zeta(v)$ have value at most $n-1$, then $(q, \zeta(v)), (S', V')$ satisfy $\diamondsuit_n$.*

[4] This is reasonable since Lemma 2 is concerned with $n-1$-runs of $\mathcal{F}^B$, and hence no counter of $\mathcal{F}^B$ will ever exceed value $n-1$ in this direction of the proof. We do not mention this property in the remaining of the proof.

Proof. Let x, y be witnesses of property $\diamondsuit_n$ for $(p, v), (S, V)$. One distinguishes three cases depending on the nature of ζ. We choose x minimal.

If $\zeta = \mathtt{IC}_K$, one sets y' to be the newly created rightmost child of y. According to the definition of the update step (U2), and since $p \in S(y)$ (from the $\diamondsuit_n$-hypothesis), we have $q \in S'(y')$ ($\diamondsuit_n.1$). Two sub-cases can happen. Le us treat first the case of x below or at y. I this case, we have $\diamondsuit_n.2$ witnessed by y', y' thanks to:

$$\mathtt{IC}_K(v)(K) = v(K) + 1 \geq V[x] + 1 \geq V[y] + 1 = V'[y] + 1 = V'[y'] \ .$$
(using the induction hypothesis, and $V'(y') = 0$ since y' is newly created)

Furthermore, since $y' = y'$, $\diamondsuit_n.3$ need be verified. It boils down to checking property $\diamondsuit_n$ for $(q, 0), (\mathtt{init}^{Q,K-1}, 0)$, which holds by property P1 for $K-1$ counters. Hence $(q, \zeta(v)), (S', V')$ satisfy $\diamondsuit_n$. Otherwise, $x \leq_{\text{rlex}} y$ but x is not below or at y. In this case, we prove that x, y' witness property $\diamondsuit_n.2$ for $(q, \zeta(v)), (S', V')$, indeed,

$$\mathtt{IC}_K(v)(K) = v(K) + 1 \geq v(K) \geq V[x] = V'[x] \ .$$
(since the counters involved in $V[x]$ are left unchanged)

Item $\diamondsuit_n.3$ holds since $x \neq y'$. Overall $\diamondsuit_n$ holds for $(q, \zeta(v)), (S', V')$.

If $\zeta = \mathtt{R}_K$, we set y' to be the new rightmost root created during the update step, and prove that y', y' witness $\diamondsuit_n$ for $(q, \zeta(v)), (S', V')$. According to the definition of the update step, and since $p \in S(x)$, we have $q \in S'(y')$. Furthermore, since $\mathtt{R}_K(v)(K) = 0 = V'[y']$ (using the fact that y' is a newly created node, which implies that $V'(y') = 0$), we get that y', y' witness $\diamondsuit_n.2$. Property $\diamondsuit_n.3$ boils down once more to property $\diamondsuit_n$ for $(q, 0), (\mathtt{init}^{Q,K-1}, 0)$, which holds by property P1 for $K-1$ counters. Hence $\diamondsuit_n$ holds for $(q, \zeta(v)), (S', V')$.

Finally, if ζ does not touch the counter K (i.e., $\zeta(v)(K) = v(K)$), then $(p, \zeta, q) \in A_{Q,K-1}$ (as defined in U1). We claim that x, y also witness $\diamondsuit_n$ for $(q, \zeta(v)), (S', V')$. According to the definition of the update step, since all counters have value at most $n-1$ for $\zeta(v)$, since $p \in S(x)$, using proposition P2 for $K-1$ counters, we have $q \in S'(x)$ (the fact that counters do not exceed $n-1$ gives that p cannot die during the execution of $A_{Q,K-1}$ on x). Clearly, $\diamondsuit_n.2$ also holds since no counters involved in the computation of $V[x]$ are touched. Finally, if $x = y$, then $(p, v|_{<K}), (S(y), V|_y)$ since $(p, v), x, y, (S, V)$ satisfy $\diamondsuit_n$. Hence, by definition of the uptate step, and by property P2 for $K-1$ counters, $(q, \zeta(v)|_{<K}), (S'(y), V'|_y)$. Hence $\diamondsuit_n$ holds for $(q, \zeta(v)), (S', V')$. □

Lemma 6. *If one goes from (S, V) to (S', V) by a preference simplification step (rule N1), and $(p, v), (S, V)$ satisfy property $\diamondsuit_n$, then $(p, v), (S', V')$ satisfy $\diamondsuit_n$.*

Proof. Let x, y be witnesses of property $\diamondsuit_n$ for $(p, v), (S, V)$. Let z be the node which has been simplified during the preference simplification step. If $z \neq y$, then, one claims that x, y also witness $\diamondsuit_n$ for $(p, v), (S, V)$. Indeed, $\diamondsuit_n.1$ is preserved since node y is left unchanged (i.e., $S(y) = S'(y)$). Property $\diamondsuit_n.2$ is also preserved since $V[x] = V'[x]$. Finally, $\diamondsuit_n.3$ is preserved since $x = y$ implies $x \neq z$

and hence node x is left unchanged. Hence $\diamondsuit_n$ for $(p, v|_{<K}), (S(y), V|_y)$ means that $\diamondsuit_n$ holds for $(p, v|_{<K}), (S'(y), V'|_y)$.

It remains the case $z = y$. We separate two case depending on the state q subject to simplification. If $p = q$, then from rule N1, there exists a node $y' >_{\text{rlex}} y$ such that p appears in $S(y')$. We claim that x, y' witness property $\diamondsuit_n$ for $(p, v), (S', V')$. For $\diamondsuit_n.1$ this is obvious since p appears in $S(y') = S'(y')$. For $\diamondsuit_n.2$ this is also obvious since $V[x] = V'[x]$. Finally, $\diamondsuit_n.3$ holds always since $x \leq_{\text{rlex}} y <_{\text{rlex}} y'$, and hence $x \neq y'$.

Otherwise, if $p \neq q$, we separate two cases. If $x = y$, then from $\diamondsuit_n.3$ for $(p, v), (S, V)$, using P2 and the fact that the counters in v have value at most $n-1$, we deduce that $(p, v|_{<K}), (S'(y), V'|_y)$. This means in particular that p appears at y ($\diamondsuit_n.1$) and $\diamondsuit_n.3$. Property $\diamondsuit_n.2$ is simply inherited from $\diamondsuit_n.2$ for $(p, v), (S, V)$ since $V[x] = V'[x]$. Otherwise, if $x <_{\text{rlex}} y = z$, then let y' be y if p does not die while q is removed from $S(y)$, otherwise, let y' be the rightmost child of y that is newly created while q is removed. In both cases, we have $x <_{\text{rlex}} y'$, and p appears in $S'(y')$. Hence x, y' witness $\diamondsuit_n.1$ for $(p, v), (S', V')$, and property $\diamondsuit_n.2$ $(p, v), (S', V')$ holds simply because $v(K) \geq V[x] = V'[x]$. Finally, since $x \neq y$, property $\diamondsuit_n.3$ is straightforward. □

Lemma 7. *If one goes from (S, V) to (S', V') by an horizontal collapse (rule N2), and $(p, v), (S, V)$ satisfy property $\diamondsuit_n$, then $(p, v), (S', V')$ satisfy property $\diamondsuit_n$.*

Proof. Let c be the node at which the horizontal collapse has been performed. Let x, y be witnesses of property $\diamondsuit_n$ for $(p, v), (S, V)$.

Two situations can occur for x. If x is at c, below c or dominated by c, then one sets $x' = c$, otherwise, set x' to be x. Two situations can occur for y. If y is at c, below c or dominated by c, then one sets $y' = c$, otherwise, set y' to be y. Let us show that x', y' witness $\diamondsuit_n$ for $(p, v), (S', V')$.

Remark first that since $x \leq_{\text{rlex}} y$, this also holds for x', y', i.e., $x' \leq_{\text{rlex}} y'$. $\diamondsuit_n.1$ is obvious: y' has been chosen to be such that p appears in $S'(y')$ according to the definition of the horizontal collapse. $\diamondsuit_n.2$ is also obvious since x' is a prefix of x and hence $v \geq V[x] \geq V[x'] = V'[x']$. For $\diamondsuit_n.3$, assume $x' = y'$. Two cases can happen. If $x' = y' = c$, then $\diamondsuit_n.3$ boils down to property P1 for $K - 1$ counters. Otherwise, $x' = y' = x = y$ is a node not at c, below c, nor dominated by c. This node has not been touched by the horizontal collapse, and hence $\diamondsuit_n.3$ for $(p, v), (S', V')$ is directly inherited from $\diamondsuit_n.3$ for $(p, v), (S, V)$. □

Lemma 8. *If one goes from (S, V) to (S', V') by an increment collapse, and $(p, v), (S, V)$ satisfy property $\diamondsuit_n$, then $(p, v), (S', V')$ satisfy $\diamondsuit_n$.*

Proof. Let c be the node at which the increment collapse has been performed, and x, y be witnessing property $\diamondsuit_n$ for $(p, v), (S, V)$. We assume x to be minimal for the prefix ordering such that x, y is a witness. This means that either $x = y$, or the parent of x is also a strict ancestor of y.

The most subtle case is when $x = c$ (case A). In this case, since no state appear at $S(c)$ (assumption of an increment collapse), $y >_{\text{rlex}} c$. This means, by minimality of x, that x dominates y. During the increment collapse, all nodes

dominated by the collapsing node (in particular y) are subject to the horizontal collapse (last item in rule N3). Hence p will appear in $S'(c')$, where c' is the direct right sibling of c resulting from the horizontal collapse. In this case, we show that c', c' are witnesses of $\diamondsuit_n$ for $(p,v),(S',V')$. Indeed, by result of the horizontal collapse, y appears in $S'(c')$. Furthermore, for z the parent of x, which is an ancestor of y, we have $v \geq V[x] \geq V[z]+1 = V'[z]+1 = V'[c']$ ($\diamondsuit_n$.2). Since $x \neq c'$, $\diamondsuit_n$.3 holds.

In all other cases, one proves the validity after executing the three first items in the definition of the increment collapse. The fourth item being an horizontal collapse preserves property $\diamondsuit_n$ by Lemma 7. Assume case A does not hold. If x is below c, set $x' = c$, otherwise set $x' = x$ (x cannot be at c since this would be case A). In the same way, if y is below c, set $y' = c$ (remark that y cannot be equal to c, since an increment collapse assumes that c be an empty node, and that p appears in $S(y)$), otherwise set $y' = y$. Let us establish that x', y' witness $\diamondsuit_n$ for $(p,v),(S'',V'')$ (where (S'',V'') is the 'configuration' just before the horizontal collapse).

Remark first that since $x \leq_{\text{rlex}} y$ we have also $x' \leq_{\text{rlex}} y'$. It is also by construction that p appears in $S''(y)$. If $x' \neq c$, then $x = x'$, and we have $v \geq V[x] = V''[x]$ ($\diamondsuit_n$.2). Otherwise, if $x' = c$, then c is a strict prefix of x, and we have once more $\diamondsuit_n$.2 using

$$v \geq V[x] \geq V[c]+1 \geq V''[c] = V''[x'] \ .$$

Assume now $x' = y'$. If $x' = y' = c$, $\diamondsuit_n$.3 boils down as usual to condition P1 for $K-1$ counters. Otherwise, $x = x' = y = y'$, and the $\diamondsuit_n$.3 for $(p,v),(S'',V'')$ is directly inherited from $\diamondsuit_n$.3 for $(p,v),(S,V)$. □

Lemma 9. *If one goes from (S,V) to (S',V') by a reduction of a node z such that $V(z) \geq n$, $(p,v),(S,V)$ satisfy property $\diamondsuit_n$, then $(p,v),(S',V')$ satisfy $\diamondsuit_n$.*

Proof. Let x, y be witnessing $\diamondsuit_n$ for $(p,v),(S,V)$. We assume x to be minimal for the prefix ordering such that x, y is a witness. We claim first that x cannot be at z or below. Indeed, this would mean $n > v \geq V[y] \geq V(z) \geq n$, which is obviously a contradiction.

As in the proof of the previous lemma, we establish the property up to before the horizontal collapse. Hence, let (S'',V'') be the configuration after the substeps R1,R2,R3. We prove that $(p,v),(S'',V'')$ satisfy $\diamondsuit_n$. Lemma 7 then teaches us that $\diamondsuit_n$ still holds after R4.

If x is dominated by z, the property $\diamondsuit_n$ is obviously preserved for $(p,v),(S'',V'')$ (nothing concerning x nor y is touched by R1,R2,R3). The same holds if $y \leq$

Otherwise, if y is below z, then necessarily $x <_{\text{rlex}} z$ (since x cannot be at z or below). It follows that z has a domination predecessor. Call it y'. Otherwise, is y is not below or at z, set $y' = y$. Let us show that x, y' witness $\diamondsuit_n$.

Since $x \leq_{\text{rlex}} y$, $x \leq_{\text{rlex}} y'$ (if $y = y'$, it is obvious, otherwise, the domination predecessor y is the immediate predecessor of y for $<_{\text{rlex}}$). It is also clear that p appears in $S''(y')$ (y' is, by construction the place where p ends up), i.e., $\diamondsuit_n$.1. Since x is not at z, below z, nor dominated by z, one has $v \geq V[x] = V''[x]$,

i.e., $\diamondsuit_n.2$. For $\diamondsuit_n.3$, the only interesting case is if $x = y'$ is the domination predecessor of z. Two sub-cases can happen. Either $x = y = y'$, and property $\diamondsuit_n.3$ for $(p, v|_{<K}), (S(y), V|_y)$ is preserved by property P3. Or y is below z, and in this case p has been inserted in $S(y')$, and $\diamondsuit_n.3$ holds this time from P4. □

Putting all the above lemmas together, one directly gets:

Lemma 10 (P2). *Assume the automaton $\mathcal{D}^{BS}_{Q,K}$ can go from (S, V) to (S', V') by a transition of $\mathcal{D}^{BS}_{Q,K}$ reading a letter A in which all checked counters assume value at least n. Then for all transitions $(p, \zeta, q) \in A$ of S and all valuations $v < n$,*

$$\text{if } (p, v), (S, V) \text{ satisfy property } \diamondsuit_n \text{ then } (q, \zeta(v)), (S', V') \text{ satisfy } \diamondsuit_n.$$

Proof. A transition of $\mathcal{D}^{BS}_{Q,K}$ is the sequence of an update step followed by simplification steps, horizontal collapses, increment collapses, and reduction steps. By Lemmas 5, 6, 7, 8, and9, each of those atomic steps preserve property $\diamondsuit_n$. □

Lemma 11 (P3,P4,P5). *Properties P3,P4, and P5 hold.*

Proof. It follows from the definitions. □

B.2 Proof of Lemma 1, second direction

Let us now turn to the second direction in the proof of Theorem 3, namely Corollary 2. From very far, this direction of the proof establishes:

> "It is possible to find a translation strategy such that, every state appearing in the state of $\mathcal{D}^{BS}_{Q,K,I,F}$ after reading some word while following this strategy, has a reason to be kept, witnessed by a run of $\mathcal{F}^{B}_{Q,K,I,F}$ over the same word."

The formal statement is the following:

Lemma 12. *There exists $\alpha(n)$ non-decreasing polynomial in n and for all n a translation strategy δ_n for $\mathcal{D}^{BS}_{Q,K,I,F}$, such that for all words u, if $\delta_n(u)$ is not accepting, there exists an accepting $\alpha(n)$-run of $\mathcal{F}^{B}_{Q,K,I,F}$ over u.*

This lemma gives us the second inequality. Indeed, assume Lemma 12 holds. Let n be the maximal n such that there is no $\alpha(n)$-run of $\mathcal{F}^{B}_{Q,K,I,F}$ over u. By maximality, this means that $[\![\mathcal{F}^{B}_{Q,K,I,F}]\!]_B(u) \leq \alpha(n+1)$. Using the above lemma, this means also that $\delta_n(u)$ is accepting, i.e., that $n \leq [\![\mathcal{D}^{BS}_{Q,K,I,F}]\!]^{\delta}_S$. Using the monotonicity of α, we get $[\![\mathcal{F}^{B}_{Q,K,I,F}]\!]_B(u) \leq \alpha([\![\mathcal{D}^{BS}_{Q,K,I,F}]\!]^{\delta}_S(u) + 1)$. We obtain:

Corollary 2. $[\![\mathcal{F}^{B}_{Q,K,I,F}]\!]_B \preccurlyeq_{\alpha(n+1)} [\![\mathcal{D}^{BS}_{Q,K,I,F}]\!]^{\delta}_S$.

Hence, we need to establish Lemma 12. As for the first direction of the proof, the proof relies on a carefully chosen property putting in relation the configurations of $\mathcal{F}^B$ and the (pre-)configurations of $\mathcal{D}^{BS}_{Q,K}$. It states the existence of a translation strategy δ_n, a non-negative integer $\alpha(n)$, and a property $\triangle_n$ of pairs of configurations $(p, v), (S, V)$ which respect the following properties:

Q1 For all $p \in P$, $(p, m), (\texttt{init}(P), 0)$ satisfy property $\triangle_n$ for all $m \leq 0$ [5].

Q2 If $\delta_n(S, V, A) = (S', V')$ and q appears in S', then there exists $(p, \zeta, q) \in A$ such that

$$(p, v), (S, V) \text{ satisfy } \triangle_n \text{ implies } (q, \zeta(v)), (S', V') \text{ satisfy property } \triangle_n.$$

Q3 If $P \cap \texttt{content}(S) = \emptyset$, and $(p, v), (S, V)$ satisfy $\triangle_n$ then property $\triangle_n$ holds for $(p, v), (\texttt{insert}(S, P), V)$.

Q4 If $P \cap \texttt{content}(S) = \emptyset$, and $p \in P$ then $(p, 0), (\texttt{insert}(S, P), V)$ satisfy property $\triangle_n$.

Q5 If $(p, v), (S, V)$ satisfy $\triangle_n$ and p appears in S, then $v(l) \leq n$ for all counters $l = 1, \ldots, K$.

The most interesting case is Q2. It clearly states that if a state q appears in the state S' of $\mathcal{D}^{BS}_{Q,K}$ after following the translation strategy, this means that there exists a reason to have kept the state q, namely a transition (p, ζ, q) such that p was already present before the transition. Furthermore here, this transition preserves the property $\triangle_n$, which will mean that the values of the counters are related in the correct way.

We can now conclude the proof of Lemma 12, assuming properties Q1,...,Q5.

Proof (of Lemma 12). Consider an n-run Σ of $\mathcal{D}^{BS}_{Q,K}$ over a word $u = A_1 \ldots A_m$ driven by δ_n:

$$\Sigma = (S_0, V_0), (S_1, V_1), \ldots, (S_m, V_m) \ .$$

Assume that this run does not end in an accepting state of $\mathcal{D}^{BS}_{Q,K,I,F}$. This means that there exists some $p_m \in S_m(x_m)$ which is accepting, i.e., belongs to I. Then, by a straightforward (downward) inductive use of Q2, we construct the sequence of $p_0, \ldots, p_m$, such that for all $i = 0, \ldots, m$, $p_i \in S_i$ and for all $i = 1, \ldots, m$, $(p_{i-1}, \zeta_i, p_i) \in A_i$ is the witness given by property Q2. If we now fix v_0 to be 0, the execution of the successive transitions (p_{i-1}, ζ_i, p_i) induces a run of $\mathcal{F}^{B}_{Q,K,I,F}$ over u of successive configurations:

$$\sigma = (p_0, v_0), \ldots, (p_m, v_m) \ .$$

Since $p_0 \in S_0$ and S_0 is the initial state of $\mathcal{D}^{BS}_{Q,K,I,F}$, this means $p_0 \in \texttt{init}(I)$, and hence, p_0 is initial. It follows that σ is an accepting run of $\mathcal{F}^{B}_{Q,K,I,F}$ over $A_1 \ldots A_m$. Furthermore, by Q1, $(p_0, v_0), (S_0, V_0)$ satisfy property $\triangle_n$. Then, by an inductive use of the consequences of P2, we get that $(p_i, v_i), (S_i, V_i)$ satisfy $\triangle_n$ for all $i = 0 \ldots m$. It follows by Q5 that no counter ever has a value beyond $\alpha(n)$ along the run σ. Hence σ is an $\alpha(n)$-accepting run of $\mathcal{F}^{B}_{Q,K,I,F}$ over $A_1 \ldots A_m$. This concludes the proof of Lemma 12. □

As for the first direction, properties Q3 and Q4 are not used in this the proof of Lemma 12. Those properties are used for the induction hypothesis to go through. The remaining of the proof is devoted to the definition, inductively on the number of counters, of the strategy δ_n, the property $\triangle_n$, and the polynomial α, and to the establishment of the invariants Q1,...,Q5.

[5] Allowing negative value simplifies in some proofs. It is used by subtracting constants to the configuration.

The base case: no counters.

In this case, the automaton $\mathcal{D}^{BS}_{Q,K}$ is deterministic, hence there is no choice for δ_n. One chooses N to be 0, and one defines $\triangle_n$ to hold for $(p, v), (S, V)$ (where v and V have empty domain since there are no counters involved) if $p \in S$. It is clear that properties Q1,...,Q5 hold for this definition of $\triangle_n$.

The general case: K counters.

We assume that we have a translation strategy $\delta_n^{Q,K-1}$ for the construction up to $K-1$ counters, as well as an upper bound $\alpha^{Q,K-1}(n)$ and a property $\triangle_n$ for $K-1$ counters which satisfy the invariants Q1,...,Q5. We need now to construct δ_n, $\alpha(n)$, and $\triangle_n$ such that Q1,...,Q5 hold.

We first define the translation strategy δ_n. The translation strategy δ_n does only depend on the current configuration of the automaton. Let us list the possible source of non-determinism which can happen in this construction.

A first source is when there is a choice in the construction, for instance there are several way to normalize a pre-configuration depending on the choice of simplifications, horizontal collapse and increment collapse. Call it the *presentation ambiguity*.

A second source of non-determinism is when the automaton can choose to reduce some nodes, which are chosen non-deterministically (call this the *reduction non-determinism*).

The last source of non-determinism is the one inherited from the induction hypothesis. Namely when the rule EX is used (this happens during an update step, and during a preference simplification step when the state is removed). Call it the *induction non-determinism*.

Our *translation strategy* δ_n resolves the different forms of non-determinism as follows.

Presentation ambiguity. It is resolved in an arbitrary way. Any choice is equivalent.

Reduction non-determinism. When choosing the nodes to reduce, the translation strategy decides to reduce exactly the nodes x such that the counter x has value at least n, i.e., has value exactly n since no counter has ever the occasion to go beyond value n using this strategy.

Induction non-determinism. When executing a letter $A_{Q,K-1}$ on a node x (rule EX), the strategy $\delta_n^{Q,K-1}$ is applied based on the configuration $(S(x), V|_x)$, i.e., the non-determinism is resolved using the induction hypothesis. This happens during the update step, and also during the preference simplification step when the simplified state is removed.

Seen from further, the reduction non-determinism is in fact the sole form of non-determinism. Indeed, the presentation ambiguity does only exist because it makes the construction easier to describe, and the induction non-determinism in fact originates from the reduction non-determinism of counters below K, via the induction.

For the definition to be precise, we need the notion of a *relevant position*. A relevant position is a position of a node which appear in some of the pre-structure occurring in the construction (as opposed to the real positions, some relevant positions may never be used in any of the states of $\mathcal{D}^{BS}_{Q,K}$). Since there are only finitely many structures, and finitely many possible transitions, there are only finitely many pre-structures that are considered. Each of them has only finitely many nodes. Hence, there are only finitely many relevant positions. This finiteness property is used in the definitions below.

We fix from now a positive integer n. As for the first implication, the difficulty of the proof is to have the correct property $\triangle_n$. Given a state p, a valuation v of the counter of $\mathcal{F}^B$, a pre-comparison structure S and a valuation V of the counters of $\mathcal{D}^{BS}_{Q,K}$, one says that $(p, v), (S, V)$ *satisfy property* $\triangle_n$ if the state p appears in $S(x)$ for some node x and:

$\triangle_n.1$ $(p, v|_{<K} - B_y(n)), (S(x), V|_x)$ has property $\triangle_n$,
where[6] $v|_{<K} - B_y(n)$ maps every counter γ to $v((x, \gamma)) - B_y(n)$,
and $B_y(n)$ is a carefully chosen non-decreasing polynomial in n. It is chosen such that $B_y(n) \geq B_z(n) + \alpha^{Q,K-1}(n) + 1$ for all relevant $y <_{\text{rlex}} z$.
Remark that in the case $K = 1$, this just boils down to $p \in S$ since there are no counters below.

$\triangle_n.2$ $v(K) \leq V\langle x \rangle \stackrel{\text{def}}{=} \sum_{\varepsilon \neq y \sqsubseteq x} C_y(n)(V(y) + 1)$,
in which each $C_y(n)$ is a carefully chosen positive integer which depends on $\mathcal{A}$ and n: Each C_y must be chosen such that it is at least $(n + 1)$ times bigger than the sum of C_z for z ranging over relevant nodes dominated by y (since the domination relation is an order, this constraint is not cyclic). In particular, one can construct each $C_y(n)$ to be a non-decreasing polynomial in n, such that the proof goes trough.

The node x is called the *witness* of property $\triangle_n$ for $(p, v), (S, V)$. One defines the value $\alpha(n)$ to be $(n + 1)C_0(n)$.

Property $\triangle_n$ holds immediately at the beginning of a run:

Lemma 13 (Q1). *For all $p \in I$, $(p, 0), (\texttt{init}(I), 0)$ satisfy property $\triangle_n$.*

The difficult point is to establish P2, i.e., establish the following Lemma 14.

Lemma 14 (Q2). *Assume following δ_n one can go from (S, V) to (S', V') for some letter A and q appears in S', then there is a transition $(p, \zeta, q) \in A$ such that for all v,*

$$\text{if } (p, v), (S, V) \text{ satisfy } \triangle_n, \text{ then } (q, \zeta(v)), (S', V') \text{ satisfy } \triangle_n.$$

The proof proceeds by establishing several lemmas stating that the operations of update, preference simplification, horizontal collapse, increment collapse, and reduction behave well with respect to property $\triangle_n$. Each result is obtained by a careful case analysis. We begin our study with the update step.

[6] Substracting $B_y(n)$ is a way to avoid to have an extra parameter for $\triangle_n$. It works by shifting the origin of $\triangle_n$ for the induction hypothesis. Technically, it means that sometimes negative values are used, but has no other impact.

Lemma 15. *If, following δ_n, one can go from (S,V) to (S',V) by an update step of letter A and q appears in $S'(x')$, there exists a node x of S and $p \in S(x)$ such that $(p,\zeta,p') \in A$, and for all v,*

$$\text{if } (p,v),x,(S,V) \text{ satisfy } \triangle_n \text{ then } (q,\zeta(v)),x',(S,V) \text{ also satisfy } \triangle_n.$$

Proof. By definition of the update step, if $q \in S'(x')$, there exists $p \in S(x)$ such that there is a possible transition (p,ζ,q) that resulted in $q \in S'(x')$ (this is clear from the definition, possibly using the inductive property P2).

There are three different cases depending on this transition.

If $\zeta = \mathtt{IC}_K$, then x' is the newly created rightmost child of x. Assume that $(p,v),x,(S,V)$ satisfy property $\triangle_n$. In particular, $(p, v|_{[K-1]} - C_x(n)),(S(x),V|_x)$ satisfy property $\triangle_n$. This means by P5 that $v(l) \leq C_x(n) + \alpha^{Q,K-1}(n) \leq C_{x'}(n)$ by choice of $C_{x'}$. Hence, by P1, $(p, v|_{<K} - C_{x'}(n)),(S',V')$ satisfies $\triangle_n$. Thus $\triangle_n.1$ holds for $(q,\zeta(v)),x',(S,V)$. Furthermore $\triangle_n.2$ holds by $\mathtt{IC}_K(v)(K) = v(K)+1 \leq V(K)\langle x\rangle + 1 \leq V\langle x\rangle + b_{x'}(n) \leq V\langle x'\rangle$. This establishes property $\triangle_n$ for $(p,\zeta(v)),x',(S',V')$.

If $\zeta = \mathtt{R}_K$, then x' is a new rightmost root added to the structure. We have $\mathtt{R}_K(v)(l) = 0$ for $l = 1,\ldots,K$. It follows by P1 that $\triangle_n.1$ holds. Property $\triangle_n.2$ holds by $\mathtt{R}_K(v) = 0 \leq V\langle x\rangle$, i.e., $(p,\zeta(v)),x',(S',V')$ satisfy property $\triangle_n$.

Otherwise ζ preserves the value of counter K. There are two sub-cases. Since an execution step was performed on node x, either $x = x'$, or q died during the execution, and is reinserted in a rightmost child x' of x. If $x = x'$, property $\triangle_n.1$ for $(p,\zeta(v)),x,(S',V')$ is directly obtained from $\triangle_n.1$ for $(p,v),x,(S,V)$ by application of P2 (for $K-1$ counters), Property $\triangle_n$ also clearly holds for $(p,\zeta(v)),x',(S',V')$ since $\zeta(v)(K) = v(K)$ and $V\langle x\rangle = V'\langle x\rangle$. If x' is a child of x, then a case similar to $\mathtt{IC}_K$ occurs. □

The case of simplification is straightforward.

Lemma 16. *If, following δ_n, one goes from (S,V) to (S',V') by a preference simplification and $p \in S'(x')$, then $p \in S(x)$ for some x, and for all v,*

$$\text{if } (p,v),x,(S,V) \text{ satisfy } \triangle_n, \text{ then } (p,v),x',(S',V') \text{ also does.}$$

Proof. This is the same proof as third case in the proof of Lemma 15. □

Lemma 17. *If, following δ_n, one goes from (S,V) to (S',V') by an horizontal collapse of node c (rule N2), c and all nodes below are empty in S, and $p \in S'(x')$, then there exists a node x of S such that $p \in S(x)$ and for all v,*

$$\text{if } (p,v),x,(S,V) \text{ satisfy property } \triangle_n, (p,v),x',(S',V') \text{ also satisfy } \triangle_n.$$

Proof. Two cases occur. Either x' is different from c, and clearly $p \in S(x)$ for $x = x'$. Furthermore, since the node x is not touched during the collapse, whenever $\triangle_n$ holds for $(p,v),x,(S,V)$, the same holds after the collapse.

Otherwise, if $x' = c$, then by definition of rule N2, there is a node x dominated by x' such that $p \in S(x)$. Item $\triangle_n.1$ simply comes from the fact that

since $(p, v|_{<K} - B_x), (S(x), V|_x)$ satisfy $\triangle_n.1$, and hence for all $l = 1, \ldots, K$, $v(l) \leq B_x + \alpha^{Q,K-1}(n) \leq B_{x'}$. Hence $(p, v|_{<K} - B_{x'}), (S(x'), V|_{x'})$ satisfy $\triangle_n.1$ by Q2. For $\triangle_n.2$, assume $(p, v), x, (S, V)$ satisfy property $\triangle_n$, and for z the parent of x (z may be ε, but this is not a problem), we have:

$$v(K) \leq V\langle x\rangle = V\langle z\rangle + \sum_{z \sqsubset t \sqsubseteq x} C_t(n)(V(t)+1) \overset{(1)}{\leq} V\langle z\rangle + C_y(n) \leq V\langle x'\rangle \ ,$$

in which (1) is by choice of $C_y(n)$, and using the fact that $V(t) \leq n$ for all t. Hence $(p, v), x', (S', V')$ satisfy property $\triangle_n$. □

Lemma 18. *If, following δ_n, one goes from (S, V) to (S', V') by an increment collapse at node y, and $p \in S'(x')$, then there exists x such that $p \in S(x)$ and for all v,*

if $(p, v), x, (S, V)$ satisfy property $\triangle_n$ then $(p, v), x', (S', V')$ also satisfy $\triangle_n$.

Proof. By definition of the increment collapse, x' cannot be below y. If $x' = y$, then there is a node x below y such that $p \in S(x)$. Assume that $(p, v), x, (S, V)$ satisfy property $\triangle_n$. Properties $\triangle_n.1$ is obtained as in the previous lemma. We also have

$$v \leq V\langle x\rangle = V\langle y\rangle + \sum_{y \sqsubset t \sqsubseteq x} C_t(n) \overset{(1)}{\leq} V\langle y\rangle + C_y(n) = V'\langle y\rangle \ ,$$

in which (1) is by construction of $C_y(n)$, and using the fact that $V(t) \leq n$ for all t. Hence $(p, v), x', (S', V')$ also satisfy property $\triangle_n.2$.

Otherwise, if x' is not at y, one sets $x' = x$. We have $p \in S'(x)$, and p, v, x', S', V' satisfies immediately property $\triangle_n$ since $V\langle x\rangle = V'\langle x\rangle$. □

Lemma 19. *If, following δ_n, one goes from (S, V) to (S', V') by a reduction at node y, and $p \in S'(x')$, then there exists x such that $p \in S(x)$ and for all v,*

$(p, v), x, (S, V)$ satisfy property $\triangle_n$, $(p, v), x', (S', V')$ also satisfy $\triangle_n$.

Proof. Recall that the reduction terminates with an horizontal collapse at node y. We prove the lemma up to that point, i.e., just before R4, the remaining being obtain using Lemma 17. Two cases can happen.

Either x' is not the domination predecessor of y or p does not belong to the state inserted by rule R3. Then $x = x'$ one sets $x = x'$. We have $S(x) = S'(x)$, $V|_x = V'|_c$, $p \in S(x)$ and $V\langle x\rangle = V'\langle x'\rangle$. The conclusion follows immediately.

Otherwise x' is the domination predecessor of y, and there exists x at y or below y such that $p \in S(x)$. Assume that $(p, v), x, (S, V)$ satisfy property $\triangle_n$. Property $\triangle_n.1$ is abtained as in the two previous lemmas. Furthermore, we have:

$$v \leq V\langle x\rangle = V\langle y\rangle + \sum_{y \sqsubset t \sqsubseteq x} C_t(n) \overset{(1)}{\leq} V\langle y\rangle + C_y(n) \leq V\langle y\rangle \ ,$$

in which (1) is by construction of $b_y(n)$, and using the fact that $V(t) \leq n$ for all t. Hence $(p, v), x', (S', V')$ also satisfy property $\triangle_n.2$. □

Gathering Lemmas 15 to 19 yields Lemma 14, i.e., property Q2. The proofs of properties Q3, Q4 and Q5 are direct. This concludes the proof of Lemma 12.

C Optimizing the output counters

The automata that we obtain from the above construction are not hierarchical by default, and uses many counters ($k2^n - 1$), in which n is the number of states of the original automaton and k the number of counters.

In its original construction, Safra faced the same problem (he was concerned with Rabin-pairs and not counters). That is why Safra's construction contains a system for reusing Rabin-pairs among a small pool. This technique has then been improved by Piterman, combining it with a latest appearance record, yielding a better complexity, a parity condition, and fewer priorities [14]. Our approach here is simply to mimic this Safra-Piterman's approach in the context of cost automata. We describe the construction, but we do not enter the proof of their validity (which is essentially a conversion of Piterman's argument to the framework of cost functions).

The principle is to attach to each counter appearing in the pre-structure counter number $\gamma(x)$, yielding the *enriched pre-structure* (S, γ). The mapping γ is required to be

- injective, and such that
- whenever $x \preceq y$ then $\gamma(x) \geq \gamma(y)$, where $\preceq$ is the transitive closure of the relation

  x is the ancestor of y or x dominates y or y is of the form (x, γ).

An *enriched structure* is an enriched pre-structure (S, γ) such that S is a structure, and $\gamma(S)$ (the range of γ) is an interval of the form $\{1, \ldots, l\}$. This is a form of normalization of the use of counters.

The only property that is important for the order $\preceq$ is that at each step of the construction, each time the counter x is modified (incremented or reset), then all counters $y \succ x$ are reset. This property tells that the construction already uses partially hierarchical counters.

The construction described here is nothing more than a standard latest appearance record technique, which is further optimized, taking advantage from the fact that the construction already uses counters which are 'partially' hierarchical. The essential principle of the construction is that, when the original construction performs an action on the counter of node x, this operation will be performed on the counter $\gamma(x)$. Furthermore, the labeling γ will be updated in consequence.

Remark that the counters in an enriched pre-structure may exceed the range of $\{1, \ldots, |Q|\}$. However, this is not the case for enriched structures.

The extra labeling γ is updated during the construction, this is done by adapting the counter actions of the original construction:

Creation when a new node x is created, it is given a new maximal counter number $\gamma(x)$.

Increments when the original construction increments the counter of node x, then the new construction increments the hierarchical counter $\gamma(x)$, and resets all counters $1, \ldots, \gamma(x) - 1$ (hierarchical action $\mathtt{I}_{\gamma(x)}$).

Resets when the original construction resets the counter of node x (all the nodes below are destroyed), then the new construction resets the counter $1, \ldots, \gamma(x)$ (hierarchical action $\mathtt{R}_{\gamma(x)}$) and sets $\gamma(x)$ is set to be a new maximal counter number.

Checks when the original construction checks the counter of node x, the new construction checks the counter $\gamma(x)$.

Just applying those rules, one reaches an enriched pre-structure. The application of the last rule makes it an enriched structure:

Normalization when γ does not range over an interval of the form $\{1, \ldots, l\}$, let $m = \min\{i \ : \ i \geq 1, i \notin \gamma(S)\}$, the counters m and below are reset and the counter numbers $\gamma(x)$ are shifted in order to reach a structure, while preserving their relative order.

D The general construction from S to B

In this section, we provide the complete construction from hierarchical S-automata to B-automaton. The construction is very resemblant to the above one, but still need to be completely restated. In Section D.1 we describe in an informal way how the construction from B to S and the construction from S to B are related. It can help the understanding to read the construction as well as Section D.1 simultaneously.

A first difference is that in the B-to-S-construction there is only need to keep track of a single representative of each state of the original automaton of the construction (as soon as a state occurs in two distinct nodes of the structure, a preference simplifications step is applied).

This is not the case anymore for S-automata. This comes from the fact that one needs to further remember at least one bit per counter. Indeed, imagine some S-automaton with two counters γ_1, and γ_2. Then, it may happen that over some word, a possible run gets to have a big value for γ_1, but a small for γ_2, while another run gets to reach the same state, and has a big value of γ_2 and a small value of γ_1. In this case, it is impossible to 'prefer' one run over the other. Indeed, if the next action is to check counter γ_1, then it would have been a catastrophe to have preferred the second run. Conversely, if the following action if to check counter γ_1, then preferring the first run would be as bad. Hence, a (history-)deterministic construction which would like to maintain all the information about this word, need to remember for each state, whether is is reachable, but even further, it needs to remember/detect what are the counters which carry big values at that moment. In the following, we refer to this vector of bit telling whether each counter has a big value or not, as a *type*. A derived object used a lot in the construction is the one of typed state: a *typed state* is a pair (p, t) consisting of a state of the S-automaton together with a type.

Another difference is that the technique of advancing a state when it dies, which is a trick inspired from the proof of positional determinacy of hierarchical B-games [6] is not usable anymore. So this optimization trick is now deactivated (which is a simplification for the proofs).

Structure of the construction In this section, we describe, given an S-automaton, how to construct an equivalent history-deterministic B-automaton. We aim at transforming a full automaton $\mathcal{F}^S_{Q,K,I,F}$ into an equivalent history-deterministic B-automaton $\mathcal{D}^{SB}_{Q,K,I,F}$.

Essentially, our construction is described as follows.

Input A finite set of states Q and a non-negative integer K.

Output A *rich B-automaton* $\mathcal{D}^{SB}_{Q,K}$. A rich B-automaton for Q, K is a complete B-automaton which has no initial states nor final states, and is enriched with the following mappings `content`, `init`, and `insert`:

- a mapping `content` from $Q_{\mathcal{D}^{SB}_{Q,K}}$ to subsets of $Q \times 2^K$. The set `content`(S) informally represents the set of typed states the state S is keeping track

of. To ease reading, we just say that p *appears in* S *with type* t instead of $(p,t) \in \texttt{content}(S)$.

In particular, if one checks some counter l, then all pairs (p,t) such that $t(l) = 0$ should be discarded from the construction. Indeed, this corresponds to checking a counter for which it is not yet established that it has reached a big value of the counter.

- a mapping `init` from $\mathcal{P}\left(Q \times 2^K\right)$ (i.e., a set of typed states) to $Q_{\mathcal{D}^{SB}_{Q,K}}$: the state $\texttt{init}(P)$ represents the initial state of $\mathcal{D}^{SB}_{Q,K}$ to be used if one wants to keep track of the runs of initial typed states in P.
- a mapping `insert` from $Q_{\mathcal{D}^{SB}_{Q,K}} \times \mathcal{P}\left(Q \times 2^K\right)$ to $Q_{\mathcal{D}^{SB}_{Q,K}}$: $\texttt{insert}(S,P)$ is defined for $P \cap \texttt{content}(S) = \emptyset$, and it represents a state which has the same behavior as S, but furthermore keeps track of the typed states in P.

The intention is that a rich automaton is a standard B-automaton, that the mapping `init` allows to construct 'parametric' sets of initial states, `content` allows to define the final states, and `insert` is a new capability given to the the automaton: the ability to handle 'on-the-fly new initial states'.

Given a s-automaton with states Q, initial states I and final states F (we will use from now the automaton $\mathcal{F}^S_{Q,K,I,F}$), one constructs the S-automaton $\mathcal{D}^{SB}_{Q,K,I,F}$ that has the following characteristics:

- the structure of the automaton is $\mathcal{D}^{SB}_{Q,K}$,
- the sole initial state is $\texttt{init}(I)$,
- the final states are the states $S \in Q_{\mathcal{D}^{SB}_{Q,K}}$ such that $F \cap \texttt{content}(S) = \emptyset$.

Our goal is to prove that this automaton is history-deterministic, and equivalent to $\mathcal{F}^S_{Q,K,I,F}$.

From now, we fix a finite set of states Q. We construct, by induction on a non-negative integer K, a rich B-automaton $\mathcal{D}^{SB}_{Q,K}$ for Q, K.

Construction: the base case. The base case consists in transforming a no-counter S-automaton into an equivalent history-deterministic B-automaton. Recall that the S-automata with no counters accept exactly the characteristic function of the compelment of the language accepted by the underlying automaton. For B-automata, this is the converse. What it means is that transforming a 0-counter S-automaton into an equivalent 0-counter B-automaton amounts to perform a complement of the underlying non-deterministic language automaton. That is why our construction performs a subset construction as for non-deterministic language automata. Since the subset construction determinizes the automaton, we also immediately get the history-determinism.

We construct the B-automaton $\mathcal{D}^{SB}_{Q,K}$ with the following characteristics:

- The set of states is $\mathcal{P}(Q)$,
- There are no counters,

– The (deterministic) transition relation is:

$$\{(P, A, \varepsilon, \{q \ : \ p \in P, \ (p, \varepsilon, q) \in A\})\} \ .$$

– For S a state of $\mathcal{D}^{SB}_{Q,K}$, $\texttt{content}(S) = S \times \{\langle\rangle\}$ (the sole type over 0 counters is denoted $\langle\rangle$).
– For $P \subseteq Q$, $\texttt{init}(P \times \{\langle\rangle\}) = P$,
– For S, P such that $\texttt{content}(S) \cap P = \emptyset$, $\texttt{insert}(S, P \times \{\langle\rangle\}) = S \cup P$.

Construction: the inductive case. We describe now the construction for K counters, for $K \geq 1$. By the induction hypothesis, we assume the existence of an automaton $\mathcal{D}^{SB}_{Q,K-1}$, as well as of the mappings $\texttt{content}^{Q,K-1}$, $\texttt{init}^{Q,K-1}$ and $\texttt{insert}^{Q,K-1}$.

A *pre-structure* (H, S) (over the states Q) is the ordered pair consisting of

– a forest S together with a mapping from its nodes to the states of $\mathcal{D}^{SB}_{Q,K-1}$. One writes S as a mapping from nodes to states of $\mathcal{D}^{SB}_{Q,K-1}$.
– a state H of $\mathcal{D}^{SB}_{Q,K-1}$. This component maintains the information concerning the states for which it is proven that a big value of counter K has been reached, i.e., such that the type maps K to 1.

From now, one identifies the structure H with $S(\oplus)$, as if $\oplus$ was a normal node of the pre-structure. This special node is isolated (no children, no parent), and assumed to be smaller than all other nodes for the order $\leq_{\text{rlex}}$. We will write from now simply S instead of (H, S)

At several places in the construction we need to make use of *typed states*, i.e, pairs of a state and a type $(p, t) \in Q \times 2^K$, as well as set of typed states. Given a type $t \in 2^{K-1}$ (over $K-1$ counters), we denote by $t[K \mapsto i]$ for $i = 0, 1$ the type t (now over K-counters) extended with $t(K) = i$. Furthermore, given a set of pairs $P \subseteq Q \times 2^K$, one naturally uniquely decomposes it as $P_0 \times \{0\} \cup P_1 \times \{1\}$. This means that $P_0, P_1 \subseteq Q \times 2^{K-1}$, and $P = \{(p, t[K \mapsto i]) \ : \ (p, t) \in P_i, \ i = 0, 1\}$.

The *content* of the pre-structure S is:

$$\begin{aligned}\texttt{content}(S) = \{&(p, t[K \mapsto 1]) \ : \ (p, t) \in \texttt{content}(S(\oplus))\} \\ &\cup \{(p, t[K \mapsto 0]) \ : \ (p, t) \in S(x) \text{ for a node } x \text{ different from } \oplus\}\end{aligned}$$

i.e., p *appears* in S with type t if either p appears in $S(\oplus)$ with type $t|_{<K}$ and $t(K) = 1$ or if $t(K) = 0$ and p appears in some node of S with type $t|_{<K}$. A *structure* is a pre-structure such that no node $S(x)$ is empty (however possibly $S(\oplus)$ is allowed to be empty).

The essential idea of the construction that the node $\oplus$ of the pre-structure maintains the information concerning the runs ending with a big value of counter K, while the other nodes of S maintains the run for which such a big value of the counter has not yet been witnessed. This tree-structure maintains the information relative to the counter values of different runs is a way very similar to the case B-to-S. In particular, the following informal principle remains:

The less is the position for $\leq_{\text{rlex}}$, the higher is the value of counter K.

(remark that the choice of $\oplus$ being minimal for $\leq_{\text{rlex}}$ is consistent with this principle).

As in the B-to-S case, there are infinitely many pre-structures but finitely many structures. A *real position* is a position which corresponds to a node in some structure. A *relevant position* is a position which occurs in some pre-structure which may occur during the description of the construction. There are infinitely many positions, but there are only finitely many real positions and relevant positions. By definition, every real position is relevant.

We construct the *B-automaton* $\mathcal{D}^{SB}_{Q,K}$ which has the following characteristics:

- The states are structures.
- The counters are either of the form x for x a possible node of some structure different from $\oplus$, or of the form (x, γ) for x node of some comparison structure (possibly $\oplus$) and γ a counter of $\mathcal{D}^{SB}_{Q,K-1}$. In particular, there are counters of the form $(\oplus, \gamma)$. Given a valuation V and a node x, $V|_x$ denotes the valuation which to γ associates $V((x, \gamma))$, i.e., a valuation of $S(x)$. In particular, $V|_\oplus$ denotes the mapping which to γ associates $V((\oplus, \gamma))$.
- The transition relation–the complex part–is described below.
- $\texttt{content}(S)$ is the mapping defined above for pre-structures,
- $\texttt{init}(P)$ where $P = P_0 \times \{0\} \cup P_1 \times \{1\}$ is the structure in which $\oplus$ is labeled with $\texttt{init}^{Q,K-1}(P_1)$, and if $P_0 \neq \emptyset$, S has a unique other node labeled with $\texttt{init}^{Q,K-1}(P_0)$.
- $\texttt{insert}(S, P)$ where $P = P_0 \times \{0\} \cup P_1 \times \{1\}$ is the structure obtained from S by replacing $S(\oplus)$ by $\texttt{insert}^{IH}(S(\oplus), P_1)$, and if $P_0 \neq \emptyset$, then one adds a new rightmost root to S, labeled with $\texttt{init}^{Q,K-1}(P_0)$.

Remark that counters are attached to nodes of the structure. This makes that there are finitely many of them. In fact, at any moment in the construction, only the counters that are attached to a node present in the structure are meaningful. That is why, whenever one *destroys a node* during the construction, it implicitly means that the corresponding counters are reset. This ensures that whenever a new node is (re-)created later, its associated counter has the default value 0.

One needs now to define the *transition relation* of our automaton. We describe the result of reading a letter A when in a structure S. The possible transitions are described as the composition of three successive steps. Each step contains a certain quantity of non-determinism. During those steps, actions on the counters are performed. Those actions are supposed to be collected during the construction without further explanations.

Beforehand, we need to define what it means to execute a letter on a node. Indeed, one principle of this construction is to use each node of the structure as the state of a sub-process executing $\mathcal{D}^{SB}_{Q,K-1}$ and keeping track of a reduced number of runs. The global construction is in charge of dispatching the runs between those sub-processes. This is why, a basic operation required in our description is to be able to execute one step of $\mathcal{D}^{SB}_{Q,K-1}$ inside a node. This is the purpose of the following definition. Let x be a node of a pre-structure, and A be a letter of

the full alphabet over $Q, K-1$, *executing* A *on node* x (possibly $x = \oplus$) consists in:

EX One non-deterministically chooses a transition $(S(x), A, t, S')$ of $\mathcal{D}^{SB}_{Q,K-1}$ (there is such a transition since $\mathcal{D}^{SB}_{Q,K-1}$ is assumed to be complete). Then, one sets the label of node x to be S'. Furthermore, for all action appearing in t on a counter γ, one performs the same action on the counter (x, γ).
Here the situation differs. No special treatment is performed for dying states. One dos not even collect them.
Variant: We sometime apply this construction on the letter

$$Rem_{p,t} = \{(q, \varepsilon, q) \ : \ q \neq p\} \cup \{(p, \mathtt{c}_l, p) \ : \ t(l) = 0\}$$

where p is a state, and t a type (over $K-1$ states). This should be thought as destroying the state p of type t or below from $S(x)$. Performing this action is called *removing* (p, t) *from* $S(x)$.

We can now turn ourselves to the definition of transitions of our automaton. It consists on the successive application of three steps: the update step, which updates the structure according to the letter read by the automaton, yielding a pre-structure, followed by the normalization step, which turns this pre-structure back into a structure, and followed finally by the reduction step during which the typed states may change of nodes, possibly reaching $\oplus$. Here, the orientation is different from the proof from the B-to-S case. It is always possible for the automaton to reduce a node. However, the interest of the B-automaton resulting from the construction is that as few typed states as possible manage to reach the node $\oplus$. Hence the B-automaton's interest is to wait as long as possible before reducing a node (this will be the translation strategy).

The first step is the update step. It consists in starting from a structure S, reading a letter A, and outputting a new pre-structure S'. Since we are working with the full alphabet over states Q, K, the letter A can be uniquely decomposed as the union $A' \cup A_K$, in which A' is a letter of the full alphabet over $Q, K-1$, and A_K is a subset of $Q \times \{\mathtt{I}_K, \mathtt{R}_K, \mathtt{c}_K\} \times Q$. (Recall that since the automaton is hierarchical, the actions $\mathtt{I}_K$ and $\mathtt{R}_K$ reset all counters.) The *update step by letter* A consists in performing the following operations:

U1 For all nodes x of S (possibly $\oplus$), one executes A'' on the node x (see above rule EX), where

$$A''_{Q,K-1} = \begin{cases} A'_{Q,K-1} \cup \{(p, \varepsilon, q) \ : \ (p, \mathtt{c}_K, q) \in A_K\} & \text{if } x = \oplus, \\ A'_{Q,K-1} & \text{otherwise.} \end{cases}$$

(The result is that when a transition checks counter K by an action $\mathtt{c}_K$, and the origin state does not occur in $\oplus$, then the corresponding run is simply erased from the structure).

U2 For all tuples $(p, \mathtt{I}_K, q)$ in A_K such that p appears in $S(x)$ with type t,
- if $x = \oplus$, one reinserts $(q, 0)$ at node $\oplus$, i.e., one replaces $S(\oplus)$ by $\mathtt{insert}^{Q,K-1}(S(\oplus), \{(q, 0)\})$,

– otherwise one creates a new rightmost child y of x, and set $S(y)$ to be $\texttt{init}^{Q,K-1}(\{(q,0)\})$.

(type 0 corresponds to the fact that $\mathtt{I}_K$ resets all counters up to $K-1$)

U3 For all tuples $(p, \mathtt{R}_K, q)$ in A_K such that p appears in some $S(x)$ with type t, one creates a new rightmost root y in S labeled with $\texttt{init}^{Q,K-1}(\{(q,0)\})$.

We reach at this point a pre-comparison structure. Call it again S. The second step, the *normalization step* aims at making it a valid structure. It consists in the application of a sequence of atomic normalization operations. The three possible operations are:

N1 If p appears in $S(x)$ with type t and in $S(y)$ with type t' for some nodes $x <_{\text{rlex}} y$ (x possibly $\oplus$) and types $t \geq t'$, then one removes (p, t') from $S(y)$ (see the special case of rule EX). This operation is called a *preference simplification step.* This rule is reversed compared to the construction from B to S. This comes from the reversed semantics of the two models B and S.

N2 If a node $x \neq \oplus$ as well as all nodes below have an empty content, one performs an *horizontal collapse at node x*:
- All nodes at x, below x, and dominated by x are destroyed (thus, by convention, all corresponding counters are reset). Let P be the set of typed states which appear at x below x or at a node dominated by x before the nodes are destroyed.
- If P is nonempty, a new node is created at position x and $S(x)$ is set to $\texttt{init}^{Q,K-1}(P)$.

N3 If there are no states appearing at x, but there are states appearing below x, one performs an *increment collapse at node x*. It consists in:
- Incrementing and checking[7] counter x.
- Destroying all nodes below x (and resetting the corresponding counters). Let P be the set of typed states appearing in these nodes before destruction.
- The label of node x is set to $\texttt{init}^{Q,K-1}(P)$.
- If x has a right sibling y, one performs an horizontal collapse at y.

After sufficiently many normalization steps, the pre-comparison structure eventually becomes a comparison structure, and no more normalization steps are possible.

During the third step, called the *reduction step*, the automaton can choose non-deterministically to reduce some nodes. As opposed to the B-to-S case, there are no check guarding a reduction. However, it is the B-automaton's interest to avoid reducing as long as possible. This non-deterministic choice is the sole real cause of non-determinism of the construction (either directly, or indirectly when the non-determinism is inherited by the induction hypothesis using the rule EX).

The *reduction of a node x* (which has to be different from $\oplus$) involves the successive application of the following operations:

[7] The check was not present in the translation from B to S. Here, it forces the automaton to preventably reduce nodes before an increment collapse involving a too high counter value is performed.

R1 The counter x is reset. This rule competes with the increment collapse. Since the increment collapse increments and check the counter, the produced B-automaton has to sometime use the reduction rule before it is too late, i.e., before the counter gets to have a too high value while an increment collapse is performed.

R2 Let P be the typed states contained at x or below. On empties the nodes at x and below.

R3 Necessarily x has a domination predecessor y (possibly $\oplus$) one inserts all typed states in P in node y. i.e., one replaces $S(y)$ by $\texttt{insert}^{Q,K-1}(S(y), P)$. We call this an *advance of the states in P (from x to y).*

R4 One performs an horizontal collapse at x.

One can check that the reduction of a node transforms a structure into a structure.

The validity of the construction is the subject of the following Lemma 20.

Lemma 20. *For all finite sets Q, all $I, F \subseteq Q$, and all non-negative integer K, the B-automaton $\mathcal{D}^{SB}_{Q,K}$ with unique initial state $\texttt{init}(I \times \{0\})$, and set of final states $\{S \ : \ \texttt{content}(S) \cap F = \emptyset\}$, is history-deterministic and is $\approx$-equivalent to the full B-automaton over states Q with K counters, initial states I and final states F.*

Before presenting the essential steps of this proof (which shares a lot of similarities with the B-to-S case), one explains in the next section how the B-to-S and S-to-B cases are related.

D.1 Some principles on the relationship with the B-to-S construction.

As for the B-to-S translation, the proof of validity of the S-to-B construction is quiet involved. Though we give below the main steps of this proof as well as the induction hypotheses that have to be maintained, we provide before in this section a description of the differences between the two constructions, and give some justifications for those differences.

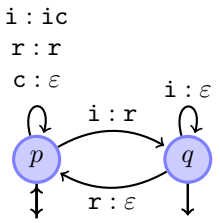
The differences are listed into precisely three categories, which answer the following interrogations: 'why and how do we use types?', 'what is the effect of having an S-automaton instead of a B-automaton as input?' and 'what is the effect of having a B-automaton instead of an S-automaton as output?'

The application of the principles listed under those three categories in fact completely describes (though informally) how the construction from B-to-S and the construction from S-to-B are related.

On the use of types. Types were not present in the B-to-S translation. This comes from the fact that simple hierarchical B-actions have particularly good properties. The B-to-S construction uses those good properties for avoiding the use of types, and correspondingly improve on the number of states.

However, if we had started with a non-simple hierarchical B-automaton, i.e., a B-automaton which would use actions of the form $\mathtt{R}_l, \mathtt{I}_l, \mathtt{c}_l$, then the use of types would again be required in the construction. Furthermore, one can explain them very easily in this context. Let us explain in the one counter case. Assume one aims into translating a B-automaton using actions $\mathtt{r}, \mathtt{i}$, and $\mathtt{c}$ (over the sole counter), into a simple B-automaton, i.e., over the action $\mathtt{r}, \mathtt{ic}$.

One first remarks that the actions $\mathtt{r}, \mathtt{i}, \mathtt{c}$ can be transformed into actions $\mathtt{r}, \mathtt{ic}$ using the following gadget simple B-automaton.



Indeed, this automaton takes as input a sequence of the actions $\mathtt{r}, \mathtt{i}, \mathtt{c}$, and computes (using the B-semantics) its value. The state p can be understood as "A big value has not been reached yet", while state q corresponds to "A big value has been reached" (though if using a bad translation strategy one may be in this state even with a small counter value). This is one bit of information. The type technique consists in maintaining next to each state this bit of information for each counter, i.e., to use typed states.

As a result, if one is interested into translating a non-simple hierarchical B-automaton into a history-deterministic S-automaton, one solution is to compute the product of the automaton with one such gadget for each counter, yielding a simple hierarchical automaton which has typed states of the original automaton as states. And then to apply the B-to-S construction. The resulting construction (call it the *combined construction*) is valid by construction. Furthermore, types naturally appear in this construction, inherited from the gadget B-automaton.

However it is possible to do better, applying two further optimizations to the combined construction:

1. In the above gadget automaton, it is always better to be in state p compared to state q. One can take advantage of this in the combined construction by allowing the preference simplification step to make decisions also based on the order on types. The result is that if one keeps track of several copies of the same state, one has only to consider incomparable types.
2. The second optimization is that, as soon as the bit concerning a specific counter is set to one (the construction has detected that the value of this counter is big), there is no need to maintain any further information concerning the counter values. We implement this optimization by collecting the typed states in a special node of the structure (the node $\oplus$ above).

After those operations done, we have obtained a construction very resemblant to the construction from S to B. We have types, and preference simplification

based on their order. Furthermore, typed states for which the counter is detected to be big are collected in the $\oplus$ states, which is an implementation of the second optimization argument.

One can remark that using this technique, the notion of dying states does not mean anything anymore. Indeed, dying was the notion capturing when the construction detects that runs reach a big value of some counter. Such run would 'die'. However, in the construction using type, the corresponding run just gets to have one bit of its type changing from 0 to 1. This is not dying anymore. This implies in particular that the special treatment used for dying states in the EX rule disappears.

The differences between having B or S-automata as input. Using the above optimized combined construction, the B-condition of the input resembles now much more to an S-condition. In particular it uses the same actions of the form $\mathtt{R}_l, \mathtt{I}_l$, and $\mathtt{c}_l$. We now explain how to transform this combined construction which takes a B-automaton as input into a construction which takes an S-automaton as input.

For this, one uses the two following transformations. The first one corresponds to the fact that between the B and S semantics, min is exchanged for max, and the second to the fact that 'check' means 'at most n' for a B-automaton, while it means 'at least n' for an S-automaton.

1. The preference step is reversed. We have mentioned the informal fact that the leftmost is the position at which a state appear in the tree structure, the 'bigger' is the corresponding counter value.
   That is why it is natural, given a B-automaton as input, that the preference step keeps only the rightmost occurrence of each state. Indeed, a B-automaton aims at minimizing the counter values.
   That is why, conversely, it is natural given an S-automaton as input, that the preference step keeps only the leftmost occurrence of each state. Indeed, an S-automaton aims at maximizing the counter values.
2. When taking a B-automaton as input, the check of counter l is allowed only when the counter has a small value, i.e., when the corresponding type maps l to 0 (this corresponds to the fact that there is no transition labeled $\mathtt{c}$ starting in state q in the above gadget B-automaton). In practice the construction kills every checked typed states when the corresponding counter has type 1: it disappears from the structure.
   If one takes an S-automaton as input, a check is only allowed when the counter value is big. Hence, we change the rule, and kill every typed state when the corresponding type maps l to 0.

Changing the output automaton between B and S. It is a strange phenomenon, that we do not understand yet perfectly, that in most natural constructions, producing a history-deterministic B-automaton or a history-deterministic S-automaton does not make much of a difference in the sense that there are only very minor differences between the two constructions.

In our case, one can see this by listing the actions performed on counters in the two constructions described so far (B-to-S, and S-to-B) when applying the atomic steps:

	output:S	output:B
horizontal collapse	`r`	`r`
increment collapse	`i`	`ic`
reduction step	`cr`	`r`

Using this table, one can modify the translation from B to history-deterministic S automata into a construction from B-automata to history-deterministic B-automata. This is done as follows: One simply has to (a) use action `ic` on the counter when performing an increment collapse (instead of `i`), and (b) action `r` when performing a reduction instead of `cr` and (c) exchange the role final and non-final states. In the same way, one can transform a construction from S-automata to history-deterministic S-automata into a construction from S-automata to history-deterministic B-automata.

One can stress one argument going toward a justification of this modification:

- The translation strategy when producing an S-automaton consists in reducing a node as soon as possible, i.e., as soon as the counter reaches value n.
- The translation strategy when producing a B-automaton consists in not reducing as long as there is no risk to go beyond counter value n. This means to not reduce as long the counter value is smaller than n, and to reduce as soon as the counter reaches value n.

It follows that the translations strategies are in fact *identical*! This implies that if the constructions are both history-deterministic using this translation strategy, then they compute the same cost function.

What one should think following these very informal remarks is that the output automaton can be chosen to be B or S rather independently from the remaining of the construction. Of course this is not a all a proof, but it pinpoints precisely what parts of the construction does depend on whether the output automaton is B or S, namely very few.

To conclude. To conclude, if one applies the modifications presented above, one can transform the B-to-S construction into an S-to-B construction. Furthermore, the resulting construction happens to be isomorphic to the construction we give.

This means that probably, when the models will be better understood, we will be able to describe those constructions differently, in a more modular way, with more modular proofs. We have not reached this situation yet, and one needs to completely redo the proof. This is particularly painful since (almost) nothing can be kept from the proof of the B-to-S construction.

D.2 Proof of Lemma 20, first direction

Principle of the first direction. In this section, we prove half of Lemma 20. This proof is involved, reflecting the complexity of the construction. This direction essentially establishes that:

If Σ is any n-run of $\mathcal{D}^{SB}_{Q,K,I,F}$ ending in S, V, and p does not appear in S,
then there are no $\alpha(n)$-runs σ (over the same word) of $\mathcal{F}^{S}_{Q,K,I,F}$ ending in state p, where α is a polynomial depending on Q and K.

From which one directly get that $[\![\mathcal{F}^{S}_{Q,K,I,F}]\!]_S \leq \alpha \circ [\![\mathcal{D}^{SB}_{Q,K,I,F}]\!]_B$.

The main objects in the proof will be configurations. Each pair (p, v) denotes a configuration of the full S-automaton over states Q with K-counters. Namely, it consists of a state $p \in Q$, and of a valuation v mapping each counter among $1, \ldots, K$ to its current value. One denotes by $\overline{v}^n$ the *type of valuation* v, fixing at n the threshold separating small from big:

$$\text{for all } l = 1 \ldots K, \quad \overline{v}^n(l) = \begin{cases} 0 & \text{if } v(l) < n\,, \\ 1 & \text{otherwise.} \end{cases}$$

In particular, we are interested in n-runs of $\mathcal{F}^{S}_{Q,K}$, and it is acceptable in this context to perform a check of counter l iff $\overline{v}^n l = 1$.

Each pair (S, V) denotes a "pre-configuration" of the automaton $\mathcal{D}^{SB}_{Q,K}$. It consists of a pre-structure S, together with a valuation V of the counters involved in S, i.e., mapping each counter from S to a non-negative integer. However, as we mentioned earlier, by construction all counters corresponding to positions not in S can be considered as existing, of default value 0. Remark that in general the pairs (S, V) are not configurations of the automaton $\mathcal{D}^{SB}_{Q,K}$ since S may not be a structure.

We can now restate an improved description of this direction of the proof, which is now enhanced with a description of the possible types.

If Σ is any n-run of $\mathcal{D}^{SB}_{Q,K,I,F}$ ending in (S, V), and (p, t) does not appear in S,
then there are no $\alpha(n)$-runs σ of $\mathcal{F}^{S}_{Q,K,I,F}$ ending in state (p, v) with $\overline{v}^{\alpha(n)} \geq t$,

in which runs are assumed to start from an initial state.

When we say below that $\mathcal{D}^{SB}_{Q,K}$ *can go from* (S, V) *to* (S', V') *reading letter* A, we mean that there is a transition of $\mathcal{D}^{SB}_{Q,K}$, reading letter A, which goes from structure S to structure S', and such that the application of the action in the transition transforms the valuation of the counters V into the valuation V'. We also use this notion for pre-structures.

The invariant $\diamondsuit'_n$, and properties P1,...,P5. The above description involves runs of both $\mathcal{F}^{S}_{Q,K}$ and $\mathcal{D}^{SB}_{Q,K}$. The main difficulty in the proof is to find the correct property which relates the configurations of the two runs: this is the goal of property $\diamondsuit'_n$. We define below what it is for a configuration (p, v) and a pre-configuration (S, V) to satisfy property $\diamondsuit'_n$. However, since the definition of $\diamondsuit'_n$ is obtained by induction on the number of counters, we formalize beforehand the inductive hypothesis, i.e., the behavioral properties we will prove for $\diamondsuit'_n$:

P1 For all $(p,t) \in P$,

$$(p,v),(\texttt{init}(P),0) \text{ satisfy property } \diamondsuit'_n \text{ for all } v \text{ such that } \overline{v}^{\alpha(n)} \leq t.$$

P2 If there is a transition from (p,v) to (p',v') in A such that no counter with value at most n is checked, and there is a transition of $\mathcal{D}^{SB}_{Q,K}$ from (S,V) to (S',V') reading letter A, then

$$(p,v),(S,V) \text{ satisfy } \diamondsuit'_n \text{ implies } (p',v'),(S',V') \text{ satisfy } \diamondsuit'_n.$$

P3 If $P \cap \texttt{content}(S) = \emptyset$ and $(p,v),(S,V)$ satisfy $\diamondsuit'_n$, then

$$(p,v),(\texttt{insert}(S,P),V) \text{ satisfy } \diamondsuit'_n.$$

P4 If $P \cap \texttt{content}(S) = \emptyset$ and $(p,t) \in P$ then,

$$(p,v),(\texttt{insert}(S,P),V) \text{ satisfy } \diamondsuit'_n \text{ for all } v \text{ such that } \overline{v}^{\alpha(n)} \leq t.$$

P5 if $(p,v),(S,V)$ satisfy $\diamondsuit'_n$ then p,t appears in S for some $\overline{v}^{\alpha(n)} \leq t$.

Assuming P1,...,P5, we can conclude the first direction.

Proof. Consider an $\alpha(n)$-run σ of $\mathcal{F}^S_{Q,K,I,F}$ over some word u,

$$\sigma = (p_0,v_0),\ldots,(p_m,v_m)\ ,$$

and an n-run Σ of $\mathcal{D}^{SB}_{Q,K,I,F}$ over the same word u,

$$\Sigma = (S_0,V_0),\ldots,(S_m,V_m)\ .$$

By P1, $(p_0,v_0),(S_0,V_0)$ satisfy $\diamondsuit'_n$. Then, by inductive application of P2 we get that $(p_i,v_i),(S_i,V_i)$ satisfy $\diamondsuit'_n$ for all $i = 0 \ldots m$. It follows by P5 that p_m appears in S_m, i.e., $p_m \in \texttt{content}(S_m)$. Assume now both p_m and S_m would be accepting, then $p_m \in \texttt{content}(S_m) \cap F = \emptyset$ (definition of the final states of $\mathcal{D}^{SB}_{Q,K,I,F}$). This is a contradiction. □

One can remark that P3 and P4 were never used in the argument. This is because P3 and P4 are used solely for the induction to go through: the construction for K-counters involve the on-the-fly insertion of states in the construction of level $K-1$, i.e; the use of the '`insert`' capability. This is also the case for the precise arguments concerning types. Sometimes the inductive construction as to reinsert a state somewhere else, preserving the type. We need be able to prove that such a shift is harmless.

The base case: no counters. For the base case, there are no counters involved. Two configurations $(p,0),(S,0)$ are said to satisfy *property* $\diamondsuit'_n$ if $p \in S$ (recall that in this case, a state of $\mathcal{D}^{SB}_{Q,K}$ is simply a set of states of $\mathcal{F}^S_{Q,K}$). We have:

Lemma 21. *Properties P1,P2,P3,P4 and P5 hold for the base construction.*

Proof. Straightforward. □

Property $\diamondsuit'_n$, the inductive case. Given a configuration (p, v) of $\mathcal{F}^S_{Q,K}$, and a pre-configuration (S, V) of $\mathcal{D}^{SB}_{Q,K}$ one says that $(p, v), (S, V)$ satisfy *property* $\diamondsuit'_n$ if there exist $x \leq_{\text{rlex}} y$ nodes of S and a type t such that:

$\diamondsuit'_n.1$ $\overline{v}^{\alpha(n)} \leq t$, $(p, t|_{<K})$ appears in $S(y)$, and $y = \oplus$ iff $t(K) = 1$.

$\diamondsuit'_n.2$ if $y \neq \oplus$, then $v(K) \leq V\langle y\rangle$ where $V\langle y\rangle$ is defined as in the case of translating B-automata to S-automata. I.e.,

$$V\langle y\rangle \stackrel{\text{def}}{=} \sum_{\varepsilon \neq z \sqsubseteq x} C_z(n)(V(z) + 1)\,,$$

in which each $C_z(n)$ is a carefully chosen positive integer which depends on $\mathcal{F}^S_{Q,K}$ and n: Each C_z must be chosen such that it is at least $(n+1)$ times bigger than the sum of C_w for w ranging over relevant nodes dominated by z (since the domination relation is an order, this constraint is not cyclic). In particular, one can construct each $C_z(n)$ to be a non-decreasing polynomial in n, such that the proof goes trough.

$\diamondsuit'_n.3$ if $x = y$, $(p, v|_{<K}), (S(y), V|_y)$ satisfy property $\diamondsuit'_n$ (for $K-1$ counters) in which $v|_{<K}$ represents the valuation v restricted to the counters $1, \ldots, K-1$.

The positions x, y making the above properties true are called the *witnesses* of property $\diamondsuit'_n$ for $(p, v), (S, V)$. One says that $(p, v), x, (S, V)$ satisfy property $\diamondsuit'_n$, or even better that $(p, v), x, y, (S, V)$ satisfy property $\diamondsuit'_n$ if we want to make explicit the witnesses for property $\diamondsuit'_n$.

D.3 Proof of Lemma 20, second direction

Let us now turn to the second direction in the proof of Theorem **??**, namely Corollary 3. From very far, this direction of the proof establishes:

> "It is possible to find a translation strategy such that, every state appearing in the state of $\mathcal{D}^{SB}_{Q,K,I,F}$ after reading some word while following this strategy, has a reason to be kept, witnessed by a run of $\mathcal{F}^S_{Q,K,I,F}$ over the same word."

The formal statement is the following:

Lemma 22. *There exists for all n a translation strategy δ_n for $\mathcal{D}^{SB}_{Q,K,I,F}$, such that for all words u, if $\delta_n(u)$ is not accepting, there exists an accepting n-run of $\mathcal{F}^S_{Q,K,I,F}$ over u.*

From which we directly get:

Corollary 3. $[\![\mathcal{D}^{SB}_{Q,K,I,F}]\!]^{\delta}_S \leq [\![\mathcal{F}^S_{Q,K,I,F}]\!]_B$.

Hence, we need to establish Lemma 22. As for the first direction of the proof, the proof relies on a carefully chosen property putting in relation the configurations of $\mathcal{F}^S$ and the (pre-)configurations of $\mathcal{D}^{SB}_{Q,K}$. It states the existence of a translation strategy δ_n, a non-negative integer $\alpha(n)$, and a property $\triangle'_n$ of pairs of configurations $(p, v), (S, V)$ which respect the following properties:

Q1 For all $(p,t) \in P$ such that $\overline{v}^n \geq t$, then $(p,v),(\texttt{init}(P),0)$ satisfy property $\triangle'_n$.

Q2 If $\delta_n(S,V,A) = (S',V')$ and q appears in S', then there exists $(p,\zeta,q) \in A$ such that

$$(p,v),(S,V) \text{ satisfy } \triangle'_n \text{ implies } (q,\zeta(v)),(S',V') \text{ satisfy property } \triangle'_n.$$

Q3 If $P \cap \texttt{content}(S) = \emptyset$, and $(p,v),(S,V)$ satisfy $\triangle'_n$ then property $\triangle'_n$ holds for $(p,v),(\texttt{insert}(S,P),V)$.

Q4 If $P \cap \texttt{content}(S) = \emptyset$, and $(p,t) \in P$ for some $\overline{v}^n \geq t$ then $(p,v),(\texttt{insert}(S,P),V)$ satisfy property $\triangle'_n$.

Q5 If $(p,v),(S,V)$ satisfy $\triangle'_n$ and p appears in S with type t such that $t(l) = 1$, then $v(l) \geq n$ for all counters $l = 1,\ldots,K$.

The most interesting case is Q2. It clearly states that if a state q appears in the state S' of $\mathcal{D}^{SB}_{Q,K}$ after following the translation strategy, this means that there exists a reason to have kept the state q, namely a transition (p,ζ,q) such that p was already present before the transition. Furthermore here, this transition preserves the property $\triangle'_n$, which will mean that the values of the counters are related in the correct way. Remark that Q5 implies that if a state is kept by the construction after a check (the type is $t(l)$ is 1), the value constructed is at least n, thus validating the fact the the constructed run is an n-run.

We can now conclude the proof of Lemma 22, assuming properties Q1,...,Q5.

Proof (of Lemma 22). Consider an n-run Σ of $\mathcal{D}^{SB}_{Q,K}$ over a word $u = A_1 \ldots A_m$ driven by δ_n:

$$\Sigma = (S_0,V_0),(S_1,V_1),\ldots,(S_m,V_m)\ .$$

Assume that this run does not end in an accepting state of $\mathcal{D}^{SB}_{Q,K,I,F}$. This means that there exists some $p_m \in S_m(x_m)$ which is accepting, i.e., belongs to I. Then, by a straightforward (downward) inductive use of Q2, we construct the sequence of $p_0,\ldots,p_m$, such that for all $i = 0,\ldots,m$, $p_i \in S_i$ and for all $i = 1,\ldots,m$, $(p_{i-1},\zeta_i,p_i) \in A_i$ is the witness given by property Q2. If we now fix v_0 to be 0, the execution of the successive transitions (p_{i-1},ζ_i,p_i) induces a run of $\mathcal{F}^S_{Q,K,I,F}$ over u of successive configurations:

$$\sigma = (p_0,v_0),\ldots,(p_m,v_m)\ .$$

Since $p_0 \in S_0$ and S_0 is the initial state of $\mathcal{D}^{SB}_{Q,K,I,F}$, this means $p_0 \in \texttt{init}(I)$, and hence, p_0 is initial. It follows that σ is an accepting run of $\mathcal{F}^S_{Q,K,I,F}$ over $A_1 \ldots A_m$. Furthermore, by Q1, $(p_0,v_0),(S_0,V_0)$ satisfy property $\triangle'_n$. Then, by an inductive use of the consequences of Q2, we get that $(p_i,v_i),(S_i,V_i)$ satisfy $\triangle'_n$ for all $i = 0 \ldots m$. It follows by Q5 is ever checked with a value smaller than n along the run σ.

Hence σ is an n-run of $\mathcal{F}^S_{Q,K,I,F}$ over $A_1 \ldots A_m$. This concludes the proof of Lemma 22. □

As for the first direction, properties Q3 and Q4 are not used in this the proof of Lemma 22. Those properties are used for the induction hypothesis to go through. The remaining of the proof is devoted to the definition, inductively on the number of counters, of the strategy δ_n, the property $\triangle'_n$, and the polynomial α, and to the establishment of the invariants Q1,...,Q5.

The base case: no counters.

In this case, the automaton $\mathcal{D}^{SB}_{Q,K}$ is deterministic, hence there is no choice for δ_n. One chooses N to be 0, and one defines $\triangle'_n$ to hold for $(p, v), (S, V)$ (where v and V have empty domain since there are no counters involved) if $p \in S$. It is clear that properties Q1,...,Q5 hold for this definition of $\triangle'_n$.

The general case: K counters.

We assume that we have a translation strategy $\delta_n^{Q,K-1}$ for the construction up to $K-1$ counters and a property $\triangle'_n$ for $K-1$ counters which satisfy the invariants Q1,...,Q5. We need now to construct δ_n and $\triangle'_n$ such that Q1,...,Q5 hold.

We first define the translation strategy δ_n. The translation strategy δ_n does only depend on the current configuration of the automaton. The same source of non-determinism are present as for the B-to-S construction, namely the presentation ambiguity (which is not really non-detetminism); the reduction non-determinism, and the induction non-determinism. In practice, what the translation strategy has to decide is when reducing nodes.

We fix from now a positive integer n. Here the situation is different from the B-to-S case. Indeed, it is always possible to reduce a node. The problem is when the automaton performs an increment collapse. This yield an increment and a check of the corresponding counter. Hence an increment collapse may make the counter be checked while exceeding value n, thus preventing the constructed run to be an n-run. That is why the translation strategy is the following: as long as a counter has a value less than n, one does not reduce it. As soon as a counter reaches value n, since it becomes dangerous, one reduces it.

As for the first implication, the difficulty of the proof is to have the correct property $\triangle'_n$. Given a state p, a valuation v of the counter of $\mathcal{F}^S_{Q,K}$, a pre-comparison structure S and a valuation V of the counters of $\mathcal{D}^{SB}_{Q,K}$, one says that $(p, v), (S, V)$ *satisfy property* $\triangle'_n$ if the state p appears in $S(x)$ for some node x and:

$\triangle'_n.1$ $(p, v|_{<K}), (S(x), V|_x)$ has property $\triangle'_n$.
Remark that in the case $K = 1$, this just boils down to $p \in S$ since there are no counters below.

$\triangle'_n.2$ $v(K) \geq V[x]$ where $V[x]$ is defined as in the first direction in the proof of validity of the B-to-S construction. I.e.,

$$V[x] \stackrel{\text{def}}{=} \sum_{\varepsilon \neq z \sqsubseteq x} (V(z) + 1) - 1 \quad \left(= \sum_{\varepsilon \neq z \sqsubseteq x} V(z) + |x| - 1 \right) .$$

The node x is called the *witness* of property $\triangle'_n$ for $(p, v), (S, V)$.

The proof proceeds by establishing, step by step, property Q1,. . . ,Q5 for $\triangle'_n$.