Maximal partition logic: towards a logical characterization of copyless cost register automata

Filip Mazowiecki[1] and Cristian Riveros[2]

1 **University of Warsaw**
2 **Pontificia Universidad Católica de Chile**

Abstract

It is highly desirable for a computational model to have a logic characterization like in the seminal work of Büchi that connects MSO with finite automata. For example, weighted automata are the quantitative extension of finite automata for computing functions over words and they can be naturally characterized by a subfragment of weighted logic introduced by Droste and Gastin. Recently, cost register automata (CRA) were introduced by Alur et al. as an alternative model for weighted automata. In hope of finding decidable subclasses of weighted automata, they proposed to restrict their model with the so-called copyless restriction. Unfortunately, copyless CRA do not enjoy good closure properties and, therefore, a logical characterization of this class seems to be unlikely.

In this paper, we introduce a new logic called maximal partition logic (MP) for studying the expressiveness of copyless CRA. In contrast to the previous approaches (i.e. weighted logics), MP is based on a new set of "regular" quantifiers that partition a word into maximal subwords, compute the output of a subformula over each subword separately, and then aggregate these outputs with a semiring operation. We study the expressiveness of MP and compare it with weighted logics. Furthermore, we show that MP is equally expressive to a natural subclass of copyless CRA. This shows the first logical characterization of copyless CRA and it gives a better understanding of the copyless restriction in weighted automata.

1998 ACM Subject Classification F.4.1. Computational logic

Keywords and phrases MSO, Finite Automata, Cost Register Automata, Weighted Automata, Weighted Logics, Semirings.

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

Weighted automata are an extension of finite state automata to compute functions over strings [8]. They have been extensively studied since Schützenberger [21], and its decidability problems [15, 2], extensions [7], and applications [18, 6] have been deeply investigated. From the logic-side, Weighted MSO logic (WMSO) has been introduced and investigated in [7, 14]. This logic is a quantitative extension of MSO to define functions over strings and its natural fragment gives a logic-based characterization of weighted automata.

Recently, Alur et al. [3] introduced the computational model of cost register automata (CRA), an alternative model to weighted automata for computing functions. The main idea of this model is to enhance deterministic finite automata with registers that can be combined with semiring operations, but the registers cannot be used for taking decisions during a computation. Alur et al. show in [3] that a fragment of CRA is equally expressive to weighted automata, but the general model is strictly more expressive.

The main advantage of introducing a new model is that it allows to study natural subclasses of functions that do not arise naturally in the classical framework. This is the case for the class of copyless CRA that where proposed in [3]. The idea of the so-called copyless restriction is to use each register at most once in every transition. Intuitively, the automaton model is register-deterministic in the sense that it cannot copy the content of each register, similar to a deterministic finite automaton that cannot make a copy of its current state. Copyless CRA is also an excellent candidate for having good decidability properties. It was stated in [3] that the existing proofs of undecidability in weighted automata rely on the unrestricted non-deterministic nature of the model and, thus, it might be possible that copyless CRA can have good decidability properties [3]. Despite that this is a natural and interesting model for computing functions, research on this line has not been pursued further and not much is known about copyless CRA.

In this paper, we introduce a new logic called *Maximal Partition Logic* (MP) to define functions over strings. In contrast to the previous approaches (WMSO), MP is based on a different set of quantifiers and it does not need to distinguish between a boolean or quantitative level of evaluation (see [7, 14]). MP is based on regular quantifiers that partition a string into maximal substrings, compute a subformula over each substring separately and then aggregate these outputs with respect to a semiring operation. Recently in [5] a logic with a similar flavor has been proposed but in a different context, namely for data words. The authors define a syntactically restricted fragment of MSO formulas with two free variables called rigid MSO-formulas. Each assignment of the free variables can be seen as choosing the substrings between the assigned positions. The rigid formulas put restrictions in the chosen set of substrings that coincides with our restriction of choosing maximal substrings.

WMSO has the drawbacks of its automata counterpart (weighted automata) – the lack of good decidability properties [2, 7, 14, 15]. We show that MP is less expressive than WMSO and even less expressive than weighted automata. Interestingly, MP can still define natural functions and it is strictly more expressive than finitely ambiguous weighted automata, a subclass of weighted automata, which has good decidability properties. In this paper we study the expressiveness of MP and compare its expressiveness with WMSO and fragments of WMSO. By this comparison, MP might be a good candidate for a logic with good decidability properties.

The main result of this paper is that MP is equally expressive to a natural fragment of copyless CRA, called *bounded alternation copyless* CRA (BAC). This fragment of copyless CRA has good closure properties and, at the same time, it does not lose much in terms of expressibility. Most examples in [3] and this paper are definable by BAC automata. This result could also be the first step in proving the decidability of MP. For example a positive answer to a decidability problem for copyless CRA will imply a positive answer for the same decidability problem for MP.

Organization. In Section 2 we introduce CRA and some basic definitions. In Section 3 we introduce MP and compare it with other formalisms. In particular we discuss the connection between this logic and rigid formulas. In Section 4 we define BAC automata and prove that that this class of automata is equally expressive to MP. In Section 5 we compare the expressiveness of MP with WMSO. We conclude in Section 6 with possible directions for future research. Due to the page limit some proofs are moved to the appendix, available online.

2 Preliminaries

In this section, we summarize the notation and definitions used for finite automata, regular expressions, MSO logic and cost register automata.

Finite automata over strings. Let Σ be a finite set of symbols. We denote by Σ^* the set of all finite strings over Σ and by ϵ the empty string in Σ^*. The length of a string $w \in \Sigma^*$ is denoted by $|w|$. Furthermore, for any $a \in \Sigma$ the number of a-symbols in w is denoted by $|w|_a$.

A finite automaton [11] over Σ^* is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of states, $\delta \subseteq Q \times \Sigma \times Q$ is a finite transition relation, q_0 is the initial state and F is the set of final states. A run ρ of $\mathcal{A}$ is a sequence of transitions of the form: $p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} p_n$ where $(p_i, a_{i+1}, p_{i+1}) \in \delta$ for every $i < n$. We say that ρ (like above) is a run of $\mathcal{A}$ over $w = a_1 \dots a_n$ if $p_0 = q_0$. Furthermore, we say that ρ is an accepting run if $p_n \in F$. A string w is accepted by $\mathcal{A}$ if there exists an accepting run of $\mathcal{A}$ over w. We denote by $\mathcal{L}(\mathcal{A})$ the language of all strings accepted by $\mathcal{A}$. A finite automaton $\mathcal{A}$ is called deterministic if δ is a function of the form $\delta : Q \times \Sigma \to Q$.

Regular expressions. Let Σ be an alphabet. The syntax of regular expressions [11] over Σ is given by:

$$R \coloneqq \varnothing \mid \epsilon \mid a \mid R \cdot R \mid R + R \mid R^*$$

where $a \in \Sigma$. The semantics of regular expressions over strings is defined as usual [11]. We write $\mathcal{L}(R)$ to denote the set of all strings that satisfy the regular expression R.

MSO. Let Σ be an alphabet. The syntax of an MSO-formula over Σ-strings is given by:

$$\varphi \coloneqq P_a(x) \mid x \leq y \mid x \in X \mid (\varphi \vee \varphi) \mid \neg\varphi \mid \exists x.\, \varphi \mid \exists X.\, \varphi$$

where $a \in \Sigma$, x and y are first-order variables and X is a set of variables. Let $w = a_1 \dots a_n \in \Sigma^*$ be a string. We represent the string w as a structure $(\{1, \dots, n\}, \leq, (P_a)_{a \in \Sigma})$, where $P_a = \{i \mid a_i = a\}$. Further, we denote by $\mathrm{dom}(w) = \{1, \dots, n\}$ the domain of w as a structure. Given a finite set $\bar{x}$ of first-order and second-order variables, an $(\bar{x}, w)$-assignment σ is a function that maps every first order variable in $\bar{x}$ to $\mathrm{dom}(w)$ and every second order variable in $\bar{x}$ to $2^{\mathrm{dom}(w)}$. Furthermore, we denote by $\sigma[x \to i]$ the extension of the $(\bar{x}, w)$-assignment σ such that $\sigma[x \to i](x) = i$ and $\sigma[x \to i](y) = \sigma(y)$ for all variables $y \neq x$. Consider an MSO-formula $\varphi(\bar{x})$ and a $(\bar{x}, w)$-assignment σ. We write $w \vDash \varphi(\sigma)$ if (w, σ) satisfies $\varphi(\bar{x})$ using the standard MSO-semantics.

Semirings and functions. A semiring is a structure $\mathbb{S} = (S, \oplus, \odot, \mathbb{0}, \mathbb{1})$ where $(S, \oplus, \mathbb{0})$ is a commutative monoid, $(S - \mathbb{0}, \odot, \mathbb{1})$ is a monoid, multiplication distributes over addition, and $\mathbb{0} \odot s = s \odot \mathbb{0} = \mathbb{0}$ for each $s \in S$. If the multiplication is commutative, we say that $\mathbb{S}$ is commutative. In this paper, we always assume that $\mathbb{S}$ is commutative. For the sake of simplicity, we usually denote the set of elements S by the name of the semiring $\mathbb{S}$. As standard examples of semirings we will consider the *semiring of natural numbers* $\mathbb{N}(+, \cdot) = (\mathbb{N}, +, \cdot, 0, 1)$, the *min-plus semiring* $\mathbb{N}_\infty(\min, +) = (\mathbb{N}_\infty, \min, +, \infty, 0)$ and the *max-plus semiring* $\mathbb{N}_{-\infty}(\max, +) = (\mathbb{N}_{-\infty}, \max, +, -\infty, 0)$ which are standard semirings in the field of weighted automata [8].

In this paper, we study the specification of functions from strings to values, namely, from Σ^* to $\mathbb{S}$. We say that a function $f : \Sigma^* \to \mathbb{S}$ is definable by a computational system $\mathcal{A}$ (e.g. weighted automaton, or CRA) if $f(w) = [\![\mathcal{A}]\!](w)$ for any $w \in \Sigma^*$ where $[\![\mathcal{A}]\!]$ is the semantics of $\mathcal{A}$ over strings. For any string w, we denote by w^r the reverse string. We say that a class of functions F is *closed under reverse* [3] if for every $f \in F$ there exists a function $f^r \in F$ such that $f^r(w) = f(w^r)$ for all $w \in \Sigma^*$.

Variables, expressions, and substitutions. Fix a semiring $\mathbb{S} = (S, \oplus, \odot, \mathbb{0}, \mathbb{1})$ and a set of variables $\mathcal{X}$ disjoint from S. We denote by $\mathrm{Expr}(\mathcal{X})$ the set of all syntactical *expressions* that can be defined from $\mathcal{X}$, constants in S, and the syntactical signature of $\mathbb{S}$. For any expression $e \in \mathrm{Expr}(\mathcal{X})$ we denote by $\mathrm{Var}(e)$ the set of variables in e. We call an expression $e \in \mathrm{Expr}(\mathcal{X})$ without variables (i.e. $\mathrm{Var}(e) = \varnothing$) a *ground* expression. For any ground expression we define $[\![e]\!] \in \mathbb{S}$ to be the evaluation of e with respect to $\mathbb{S}$.

A *substitution* over $\mathcal{X}$ is defined as a mapping $\sigma : \mathcal{X} \to \mathrm{Expr}(\mathcal{X})$. We denote the set of all substitutions over $\mathcal{X}$ by $\mathrm{Subs}(\mathcal{X})$. A *ground substitution* σ is a substitution where each expression $\sigma(x)$ is ground for each $x \in \mathcal{X}$. Any substitution σ can be extended to a mapping $\hat{\sigma} : \mathrm{Expr}(\mathcal{X}) \to \mathrm{Expr}(\mathcal{X})$ such that, for every $e \in \mathrm{Expr}(\mathcal{X})$, $\hat{\sigma}(e)$ is the resulting expression $e[\sigma]$ of substituting each $x \in \mathrm{Var}(e)$ by the expression $\sigma(x)$. For example, if $\sigma(x) = 2x$ and $\sigma(y) = 3y$, and $e = x + y$, then $\hat{\sigma}(e) = 2x + 3y$. By using the extension $\hat{\sigma}$, we can define the composition substitution $\sigma_1 \circ \sigma_2$ of two substitutions σ_1 and σ_2 such that $\sigma_1 \circ \sigma_2(x) = \hat{\sigma}_1(\sigma_2(x))$ for each $x \in \mathcal{X}$.

A valuation is defined as a substitution of the form $\nu : \mathcal{X} \to \mathbb{S}$. We denote the set of all valuations over $\mathcal{X}$ by $\mathrm{Val}(\mathcal{X})$. Clearly, any valuation ν composed with a substitution σ defines an expression without variables that can be evaluated as $[\![\nu \circ \sigma(x)]\!]$ for any $x \in \mathcal{X}$.

In this paper, we say that two expressions e_1 and e_2 are equal (denoted by $e_1 = e_2$) if they are equal up to evaluation equivalence, that is, $[\![\hat{\nu}(e_1)]\!] = [\![\hat{\nu}(e_2)]\!]$ for every valuation $\nu \in \mathrm{Val}(\mathcal{X})$. Similarly, we say that two substitutions σ_1 and σ_2 are equal (denoted by $\sigma_1 = \sigma_2$) if $\sigma_1(x) = \sigma_2(x)$ for every $x \in \mathcal{X}$.

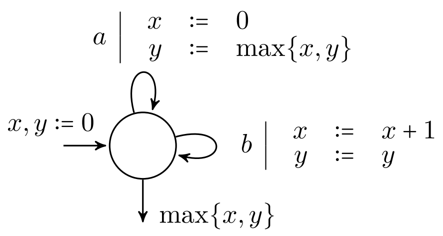
Cost register automata. A cost register automaton (CRA) over a semiring $\mathbb{S}$ [3] is a tuple $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \delta, q_0, \nu_0, \mu)$ where Q is a set of states, Σ is the input alphabet, $\mathcal{X}$ is a set of variables (we also call them registers), $\delta : Q \times \Sigma \to Q \times \mathrm{Subs}(\mathcal{X})$ is the transition function, q_0 is the initial state, $\nu_0 : \mathcal{X} \to \mathbb{S}$ is the initial valuation, and $\mu : Q \to \mathrm{Expr}(\mathcal{X})$ is the final output function. A configuration of $\mathcal{A}$ is a tuple (q, ν) where $q \in Q$ and $\nu \in \mathrm{Val}(\mathcal{X})$ represents the current values in the variables of $\mathcal{A}$. Given a string $w = a_1 \dots a_n \in \Sigma^*$, the run of $\mathcal{A}$ over w is a sequence of configurations: $(q_0, \nu_0) \xrightarrow{a_1} (q_1, \nu_1) \xrightarrow{a_2} \dots \xrightarrow{a_n} (q_n, \nu_n)$ such that, for every $1 \le i \le n$, $\delta(q_{i-1}, a_i) = (q_i, \sigma_i)$ and $\nu_i(x) = [\![\nu_{i-1} \circ \sigma_i(x)]\!]$ for each $x \in \mathcal{X}$. The output of $\mathcal{A}$ over w, denoted by $[\![\mathcal{A}]\!](w)$, is $[\![\hat{\nu}_n(\mu(q_n))]\!]$.

The run of $\mathcal{A}$ over w can be equally defined in terms of ground expressions rather than values. A ground configuration of $\mathcal{A}$ is a tuple (q, ς) where $q \in Q$ and $\varsigma \in \mathrm{Subs}(\mathcal{X})$ is a ground substitution. Given a string $w = a_1 \dots a_n \in \Sigma^*$, the ground run of $\mathcal{A}$ over w is a sequence of ground configurations: $(q_0, \varsigma_0) \xrightarrow{a_1} \dots \xrightarrow{a_n} (q_n, \varsigma_n)$ such that for $1 \le i \le n$, $\delta(q_{i-1}, a_i) = (q_i, \sigma_i)$, $\varsigma_0 = \nu_0$ and $\varsigma_i(x) = \hat{\varsigma}_{i-1}\big(\sigma_i(x)\big)$ for each $x \in \mathcal{X}$. We denote the output ground expression of $\mathcal{A}$ over a string w by $|\mathcal{A}|(w) = \hat{\varsigma}_n(\mu(q_n))$. Notice that, in contrast to ordinary runs, ground runs keep ground expressions as partial values of the run. It is easy to see that $[\![\mathcal{A}]\!](w) = [\![|\mathcal{A}|(w)]\!]$.

Copyless restriction and copyless CRA. We say that an expression $e \in \mathrm{Expr}(\mathcal{X})$ is *copyless* if e uses every variable from $\mathcal{X}$ at most once. For example, $x \cdot (y + z)$ is copyless but $x \cdot y + x \cdot z$ is not copyless (because x is mentioned twice). Notice that the copyless restriction is a syntactical constraint over expressions. Furthermore, we say that a substitution σ is *copyless* if for every $x \in \mathcal{X}$ the expression $\sigma(x)$ is copyless and $\mathrm{Var}(\sigma(x)) \cap \mathrm{Var}(\sigma(y)) = \varnothing$ for every pair of different registers $x, y \in \mathcal{X}$. Copyless substitutions, similar to copyless expressions, are restricted in such a way that each variable is used at most once in the whole substitution.

A CRA $\mathcal{A}$ is called *copyless* if for every transition $\delta(q_1, a) = (q_2, \sigma)$ the substitution σ is copyless; and for every state $q \in Q$ the expression $\mu(q)$ is copyless, where μ is the output function of $\mathcal{A}$. In other words, every time that registers from $\mathcal{A}$ are operated, they can be used just once. In the following, we give some examples of copyless CRA.

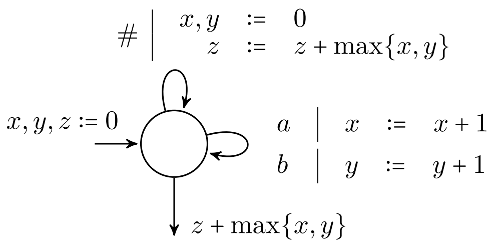
▸ **Example 1.** Let $\mathbb{S}$ be the max-plus semiring $\mathbb{N}_{-\infty}(\max,+)$ and $\Sigma = \{a,b\}$. Consider the function f_1 that for a given string $w \in \Sigma^*$ computes the longest substring of b's. This can be easily defined by the following CRA $\mathcal{A}_1$ with two registers x and y.


$a \mid x := 0$, $y := \max\{x,y\}$
$x, y := 0$
$b \mid x := x+1$, $y := y$
$\max\{x,y\}$

$\mathcal{A}_1$ stores in the x-register the length of the last suffix of b's and in the y-register the length of the longest substring of b's seen so far. After reading a b-symbol $\mathcal{A}_1$ adds one to x (the b-infix has increased by one) and it keeps y unchanged. Furthermore, after reading an a-symbol it resets x to zero and updates y by comparing the substring of b's that has just finished (i.e. the previous x-content) with the length of the longest substring of b's (i.e. the previous y-content) that has been seen so far. Finally, it outputs the maximum between x and y.

One can easily check that the previous CRA satisfies the copyless restriction and, therefore, it is a copyless CRA. Indeed, each substitution is copyless and the final output expression $\max\{x,y\}$ is copyless as well.

▸ **Example 2.** Again, let $\mathbb{S}$ be the max-plus semiring $\mathbb{N}_{-\infty}(\max,+)$ and $\Sigma = \{a,b,\#\}$. Consider the function f_2 such that, for any $w \in \Sigma^*$ of the form $w_0\#w_1\#\ldots\#w_n$ with $w_i \in \{a,b\}^*$, it computes the maximum number of a's or b's for each substring w_i (i.e. $\max\{|w_i|_a, |w_i|_b\}$) and then it sums these values over all substrings w_i, that is, $f_2(w) = \sum_{i=0}^{n} \max\{|w_i|_a, |w_i|_b\}$. One can check that the copyless CRA $\mathcal{A}_2$ defined below computes f_2:



In the above diagram of $\mathcal{A}_2$, we omit an assignment if a register is not updated (i.e. it keeps its previous value). For example, for the a-transition we omit the assignments $y := y$ and $z := z$ for the sake of presentation of the CRA. Similarly, we also omit the assignment $x := x$ and $z := z$ for the b-transition. One should keep in mind these assignments because of the copyless restriction.

The copyless CRA $\mathcal{A}_2$ follows similar ideas to $\mathcal{A}_1$: the registers x and y count the number of a's and b's, respectively, in the longest suffix without # and the register z stores the partial output without considering the last suffix of a's and b's. When the last substring w_i over $\{a,b\}$ is finished (i.e. there comes a #-symbol or the input ends), then $\mathcal{A}_2$ adds the maximum number of a's or b's in w_i to z (i.e. $z := z + \max\{x,y\}$).

Trim assumption. For technical reasons, in this paper we assume that our finite automata and cost register automata are always *trim*, namely, all their states are reachable from some initial states (i.e., they are accessible) and they can reach some final states (i.e., they are co-accessible). It is worth noticing that verifying if a state is accessible or co-accessible is reduced to a reachability test in the transition graph [19]; and this can be done in NLogSpace. Thus, we can assume without lost of generality that all our automata are trimmed.

3 A quantitative logic based on partitions

3.1 Regular selectors

In this subsection we extend regular expressions for selecting intervals from a string. Our approach is similar to the one in [9, 12], but we restrict the selection to just a set of intervals (i.e. spans in [9]) instead of relations of intervals.

Fix a string $w \in \Sigma^*$. An interval of w is a pair (i, j) such that $1 \leq i \leq j \leq |w|$. We write $\mathrm{Int}(w)$ for the set of all intervals of w. For an interval (i, j), we denote by $w[i, j]$ the substring between positions i and j, by $w[\cdot, j]$ the prefix of w until position j and by $w[i, \cdot]$ the suffix of w starting from position i. For the sake of simplification, we define $w[\cdot, i]$ and $w[i, \cdot]$ equal to ϵ whenever $i \notin \{1, \ldots, |w|\}$.

A *regular selector* (RS) over Σ (or just selector or triple) is a triple (R, S, T) where R, S, and T are regular expressions over Σ. The set of all selectors over Σ is denoted by RS_Σ. We usually write $R\langle S\rangle T$ instead of (R, S, T). The main motivation of a selector (R, S, T) is to select intervals (i, j) from a string w by dividing w into $w = xyz$ such that x, y, and z match R, S, and T, respectively, and $w[i, j] = y$. Specifically, we say that an interval (i, j) of a string w is selected by a triple $R\langle S\rangle T$ if, and only if, $w[\cdot, i-1] \in \mathcal{L}(R)$, $w[i, j] \in \mathcal{L}(S)$, and $w[j+1, \cdot] \in \mathcal{L}(T)$. The set of all intervals of w selected by $R\langle S\rangle T$ is defined as:

$$\mathrm{Sel}(w, R\langle S\rangle T) \;=\; \left\{\, (i, j) \in \mathrm{Int}(w) \;\middle|\; w[\cdot, i-1] \in \mathcal{L}(R) \wedge w[i, j] \in \mathcal{L}(S) \wedge w[j+1, \cdot] \in \mathcal{L}(T) \,\right\}$$

▸ **Example 3.** Let $\Sigma = \{a, b\}$. Suppose that we want to define all maximal intervals that define substrings of b-symbols in a string. This can be defined by the following regular selector:

$$((a+b)^* a + \epsilon)\ \langle b^+\rangle\ (a(a+b)^* + \epsilon)$$

The purpose of a selector $R\langle S\rangle T$ is to extract all intervals that satisfy the regular expression S under the context defined by R and T. In our logic, we restrict the semantics of selectors to consider just intervals that are maximal in terms of containment. More precisely, we say that an interval (i_1, j_1) is contained in an interval (i_2, j_2) (denoted by $(i_1, j_1) \sqsubseteq (i_2, j_2)$) if, and only if, $i_2 \leq i_1$ and $j_1 \leq j_2$. The $\sqsubseteq$-relation basically defines a partial order between intervals and we can talk about the $\sqsubseteq$-maximal intervals of a set. We write $\mathrm{Max}_\sqsubseteq(I)$ to denote the set of all maximal intervals in I with respect to the partial order $\sqsubseteq$ for any set I of intervals. Given a selector $\mathtt{R} = R\langle S\rangle T$ and a string w, we define the set of intervals selected by $\mathtt{R}$ over w under maximal semantics by:

$$\mathrm{Max}(w, R\langle S\rangle T) \;=\; \mathrm{Max}_\sqsubseteq(\mathrm{Sel}(w, R\langle S\rangle T))$$

That is, under the maximal semantics we select just intervals that are maximal with respect to the partial order $\sqsubseteq$. This new semantics simplifies selectors from Example 3.

▸ **Example 4.** With the maximal semantics, we can easily define the the set of maximal intervals that define substrings of b-symbols like in Example 3. By using the maximal semantics we can define this set of intervals easily as follows:

$$(a+b)^*\ \langle b^+\rangle\ (a+b)^*$$

We usually do not need the context R and T when we are using the maximal semantics. For instance, in the previous example R and S were equal to $(a+b)^*$ and could be omitted. For the sake of simplification, we usually omit R, T and the angular brackets whenever R and T are both equivalent to Σ^*. We can simplify the above selector and just write b^+ to select the maximal intervals of b's.

3.2 Maximal partition logic

For a fixed semiring $\mathbb{S} = (S, \oplus, \odot, \mathbb{0}, \mathbb{1})$ and an alphabet Σ we define the *maximal partition logic* (MP). This is a logic for computing functions similar to weighted logics [7] but with a different set of quantifiers that are parametrized by regular selectors. Formally, the formulas of MP over a semiring $\mathbb{S} = (S, \oplus, \odot, \mathbb{0}, \mathbb{1})$ and an alphabet Σ are defined by the following grammar:

$$\varphi \quad := \quad s \mid (\varphi \oplus \varphi) \mid (\varphi \odot \varphi) \mid \bigoplus \mathtt{R}.\ \varphi \mid \bigodot \mathtt{R}.\ \varphi$$

where $s \in \mathbb{S}$ and $\mathtt{R} \in \mathrm{RS}_\Sigma$ is a regular selector. Similar as in [7], our formulas use constants $s \in \mathbb{S}$ and moreover constants are the only atomic formulas in MP. Our logic also includes the binary sum $\oplus$ and product $\odot$ like it is common in weighted or quantitative logics [7, 14]. Of course, the signature of these operators depends on the semiring that is chosen, for example $\max\{\varphi_1, \varphi_2\}$ or $\varphi_1 + \varphi_2$ are MP-formulas for the max-plus semiring $\mathbb{N}_{-\infty}(\max, +)$. The new quantifiers here are the formulas of the form $\bigoplus \mathtt{R}.\ \varphi$ or $\bigodot \mathtt{R}.\ \varphi$. We say that $\bigoplus \mathtt{R}.$ and $\bigodot \mathtt{R}.$ are partition quantifiers. We stress again that the signature of these quantifiers depends on the signature of the semiring. The idea here is that, over any input $w \in \Sigma^*$, $\mathtt{R}$ will select the set of maximal intervals I of w and then φ will be computed over each substring $w[i,j]$ for $(i,j) \in I$. The outputs of φ over $w[i,j]$ will be aggregated under the $\oplus$ or $\odot$ operation. It is important to remark that φ will be computed over a substructure of w and not over the whole string. This differs from the classical logic semantics where an element, set or relation is chosen and the subformulas are evaluated over the whole structure plus an assignment over the variables. Here we have taken a different direction and we consider just the substructure induced by the interval provided by the regular selector.

Formally, each MP-formula φ defines a function $[\![\varphi]\!]$ from Σ^* to $\mathbb{S}$. The semantics of MP-formulas is defined recursively over any string $w \in \Sigma^*$ as follows:

$$\begin{aligned}
[\![s]\!](w) &:= s \\
[\![\varphi_1 \oplus \varphi_2]\!](w) &:= [\![\varphi_1]\!](w) \oplus [\![\varphi_2]\!](w) \\
[\![\varphi_1 \odot \varphi_2]\!](w) &:= [\![\varphi_1]\!](w) \odot [\![\varphi_2]\!](w) \\
[\![\bigoplus \mathtt{R}.\ \varphi]\!](w) &:= \bigoplus_{(i,j) \in \mathrm{Max}(w,\mathtt{R})} [\![\varphi]\!](w[i,j]) \\
[\![\bigodot \mathtt{R}.\ \varphi]\!](w) &:= \bigodot_{(i,j) \in \mathrm{Max}(w,\mathtt{R})} [\![\varphi]\!](w[i,j])
\end{aligned}$$

for any MP-formulas φ, φ_1, and φ_2; and for any regular selector $\mathtt{R}$ over Σ. For the special case when $\mathrm{Max}(w, \mathtt{R}) = \varnothing$, we define $[\![\bigoplus \mathtt{R}.\ \varphi]\!](w) = \mathbb{0}$ and $[\![\bigodot \mathtt{R}.\ \varphi]\!](w) = \mathbb{1}$.

In the sequel we give some examples in order to understand the syntax and semantics of the logic.

▸ **Example 5.** Suppose that we want to compute the number of b-symbols in a string and we want to specify this function with MP-formulas over the max-plus semiring $\mathbb{N}_{-\infty}(\max, +)$. Here, we use $\max\{\cdot,\cdot\}$ and $+$ for the binary operators, and $\mathrm{Max}\ \mathtt{R}.\ \varphi$, $\sum \mathtt{R}.\ \varphi$ for the partition quantifiers. Then the number of b-symbols in a string can be computed easily with the following formula:

$$\varphi_1 \;:=\; \sum b.\ 1$$

To understand φ_1, we need to first understand the regular selector given by the simple expression b. Recall that this is a shorthand for $(a+b)^*\langle b\rangle(a+b)^*$. Thus, the regular selector b is choosing all the maximal intervals with just one b-symbol, that is, all substrings of the form b. Then for each b-symbol in the input the formula is outputting 1 and, by aggregating them all, it is calculating the number of b-symbols in a string.

By definition for any fixed string u, a formula of the form $\sum u.\ 1$ counts how many times the u-string appears in the input. It is interesting to compare how simple and readable is this formula in comparison to any equivalent formula in other logics (e.g. weighted logics [7]) or other formalism (e.g. weighted expressions [20]) for computing function over strings.

MP also has the ability of defining regular properties in a simple way. For example, let R be a regular expression and suppose one wants to output $\mathbb{1}$ if the input is definable by R and $\mathbb{0}$ otherwise. This is defined by the expression $\bigoplus \epsilon\langle R\rangle\epsilon.\ \mathbb{1}$. Here, the prefix and suffix of the selected interval are ϵ, thus the regular selector chooses the whole string depending if it belongs to R. If the string belongs to R the formula outputs $\mathbb{1}$; otherwise it outputs $\mathbb{0}$. Therefore, MP has a native use of regular expressions embedded in the language.

▸ **Example 6.** Suppose that one wants to compute the length of the maximum substring of b-symbols. The following formula shows how to define this function in MP logic over the semiring $\mathbb{N}_{-\infty}(\max,+)$:

$$\varphi_2 \;:=\; \text{Max } b^+.\ \sum b.\ 1$$

In the previous formula, the partition quantifier Max b^+ is breaking the input into maximal substrings of b-symbols and passing each substring to the subformula $\sum b.\ 1$ that counts the number of b-symbols in the substring. Finally we maximize over all maximal substrings of b-symbols.

We want to highlight again how declarative is φ_2 in comparison to other logics. Here the words are partitioned into maximal substrings of b-symbols and the length of each substring is counted. In the end it is maximized over all lengths.

The next example defines a more complicated function.

▸ **Example 7.** Let $\Sigma = \{a, b, \#\}$ and suppose that we want to compute the same function as in Example 4, that is, for each subinterval between #-letters, we want the maximum between its number of a- or b-symbols, and then sum these values over all intervals. This complicated function can be easily defined by the following MP formula over the max-plus semiring $\mathbb{N}_{-\infty}(\max,+)$:

$$\varphi_3 \;:=\; \sum (a+b)^+.\ \max\Big\{\ \sum a.\ 1\ ,\ \sum b.\ 1\ \Big\}$$

One can easily understand the function from the definition of the MP-formula φ_3. The first quantifier $\sum (a+b)^+$ is dividing the word into maximal substrings of a- and b-symbols or, in other words, substrings that are between #-symbols (or the prefix and the suffix). Then for each of these substrings the subformula $\max\Big\{\ \sum a.\ 1\ ,\ \sum b.\ 1\ \Big\}$ is taking the maximum between the number of a-symbols or b-symbols. In the end these values are summed over all maximal substrings of a- and b-symbols.

3.3 Design decisions behind MP

MP uses regular selectors for choosing intervals from the input and computing a subformula over the selected substrings. Here we are taking two design decisions about this new logic: (1) we decided to use regular expressions for selecting intervals and (2) we consider only the maximal intervals. In the following we give evidence of how these decisions are related with previous work.

Regular expressions have been used from the beginning for extracting intervals from strings [1, 10]. For example, regular expressions are used in practice for matching substrings from files or documents [10]. Similar to regular selectors, a RegExp-engine (like egrep) parses a regular expression R and an input document D, and extracts all words from D that match with R. RegExp-engines even use parentheses "$(\cdot)$" for declaring that the subword that matches the subexpressions between parentheses must be output. Furthermore, in RegExp-engines the parentheses semantics is greedy, namely, they select the larger subword that matches the subexpression inside parentheses. This semantics is similar to the maximal semantics of regular selectors with the exception that the greedy-semantics is even more restrictive since the selected interval depends on how the input is parsed from left-to-right [10]. Despite this fact, it is interesting that even a more restricted flavor of the maximal semantics is already presented in practice which supports the decision of including it for MP.

Recently, regular expressions for substring selection have been considered in the context of information extraction [9, 12]. In [9], the authors propose a regular expression language enhanced with variables, called regex, to extract relations of substrings from an unstructured document. Regular selectors can be seen as a restrictive subfragment of regex, where only one variable is used. We note that we could have used regex language or any other formalism with the maximal semantics for selecting intervals from a string. However, we believe that regular selectors are very simple, flexible and concise, and they include the best features of previous works without loosing expressibility [9].

Finally, we could have also chosen MSO logic with two free variables for selecting intervals instead of regular expressions (i.e. with respect to the normal semantics), namely, for any MSO-formula $\varphi(x,y)$ to extract the set $\mathrm{Sel}(w,\varphi(x,y))$ of all intervals (i,j) over a string w such that: $w \models \varphi(i,j)$. Of course, both formalism for selecting intervals are equivalent. Namely, it is easy to show that for every MSO-formula $\varphi(x,y)$ there exists a finite set of regular selectors $\mathtt{R}_1,\ldots,\mathtt{R}_n$ such that $\bigcup_{i=1}^{n} \mathrm{Sel}(w,\mathtt{R}_i) = \mathrm{Sel}(w,\varphi(x,y))$ and vice versa. Notice that with this definition of selecting intervals by MSO formulas we can assume that formulas additionally satisfy $x \le y$.

Regarding the maximal semantics of regular selectors, it is important to note that a similar semantics was studied before. In [5] the authors define a subset of MSO formulas with two free variables called rigid MSO-formulas. Formally, an MSO-formula $\varphi(x,y)$ over strings is called rigid if for all strings $w \in \Sigma^*$ and all positions $i \in \mathrm{dom}(w)$ there is at most one position $j \in \mathrm{dom}(w)$ such that $w \models \varphi(i,j)$, and at most one $j' \in \mathrm{dom}(w)$ such that $w \models \varphi(j',i)$; in other words, $\varphi(x,y)$ defines two partial injective functions on $\mathrm{dom}(w)$. One can easily check that intervals defined by a regular selector with the maximal semantics are also definable by a rigid MSO-formula. Indeed, for any regular selector $\mathtt{R}$ suppose that $\varphi_{\mathtt{R}}(x,y)$ is an equivalent MSO formula that defines the same set of intervals (i.e. with the normal semantics). Then $\mathrm{Max}(w,\mathtt{R}) = \mathrm{Sel}(w,\varphi^*_{\mathtt{R}}(x,y))$, where:

$$\varphi^*_{\mathtt{R}}(x,y) \;:=\; \varphi_{\mathtt{R}}(x,y) \;\wedge\; \forall x'.\forall y'.\; (\varphi_{\mathtt{R}}(x',y') \;\wedge\; x' \le x \;\wedge\; y \le y') \to (x' = x \;\wedge\; y' = y)$$

The formula $\varphi^*_{\mathtt{R}}(x,y)$ is restricting the intervals that satisfy $\varphi_{\mathtt{R}}(x,y)$ to be maximal. In particular, one can easily check that $\varphi^*_{\mathtt{R}}(x, y)$ is indeed a rigid formula. This implies that the maximal semantics can be expressed by rigid formulas. The next proposition shows that rigid formulas can also be defined by sets of regular selectors with the maximal semantics.

▸ **Proposition 8.** *For every regular selector* $\mathtt{R}$ *there exists a rigid formula* $\varphi_{\mathtt{R}}(x, y)$ *such that* $\mathrm{Max}(w, \mathtt{R}) = \mathrm{Sel}(w, \varphi_{\mathtt{R}}(x, y))$ *for every* $w \in \Sigma^*$*. Furthermore, for every rigid formula* $\varphi(x, y)$ *there exists a set of regular selectors* $\mathtt{R}_1, \dots, \mathtt{R}_n$ *such that* $\mathrm{Sel}(w, \varphi(x, y)) = \bigcup_{i=1}^n \mathrm{Max}(w, \mathtt{R}_i)$ *for every* $w \in \Sigma^*$*.*

4 Automata-based characterization of MP

In [17] (see Corollary 1) it was shown that the class of functions defined by copyless CRA is not closed under reverse, that is, the run of copyless CRA is asymmetric with respect to the input. Intuitively, this fact is contrary to the spirit of a logical characterization for a computational model: a logic should express properties over the whole string and its expressiveness should not depend on the orientation of the input. This implies that a characterization of copyless CRA in terms of a logic is far to be possible. To solve this, we introduce the subclass of *bounded alternation copyless* CRA (in short BAC) which is a restricted variant of copyless CRA. We show that BAC have good closure properties and, moreover, this is the right model to capture the expressiveness of maximal partition logic.

The *alternation* of an expression $e \in \mathrm{Expr}(\mathcal{X})$ is defined as the maximum number of switches between $\oplus$ and $\odot$ operations over all branches of the parse-tree of e. Formally, let $\otimes \in \{\oplus, \odot\}$ and $\bar{\otimes}$ be the dual operation of $\otimes$ in $\mathbb{S}$. We define the set of expressions $\mathrm{Expr}^{\otimes}_0(\mathcal{X})$ with 0-alternation by $\mathrm{Expr}^{\otimes}_0 = \mathcal{X} \cup \mathbb{S}$. For any $N \geq 1$, we define the set of expressions $\mathrm{Expr}^{\otimes}_N(\mathcal{X})$ as the $\otimes$-closure of $\mathrm{Expr}^{\bar{\otimes}}_{N-1}(\mathcal{X})$, namely, $\mathrm{Expr}^{\otimes}_N(\mathcal{X})$ is the minimal set of expressions that contains $\mathrm{Expr}^{\bar{\otimes}}_{N-1}(\mathcal{X})$ and satisfies $e_1 \otimes e_2 \in \mathrm{Expr}^{\otimes}_N(\mathcal{X})$ for all $e_1, e_2 \in \mathrm{Expr}^{\otimes}_N(\mathcal{X})$. We denote by $\mathrm{Expr}_N(\mathcal{X}) = \mathrm{Expr}^{\oplus}_N(\mathcal{X}) \cup \mathrm{Expr}^{\odot}_N(\mathcal{X})$ the set of all expressions with alternation bounded by N.

We say that a copyless CRA $\mathcal{A}$ has *bounded alternation* if there exists $N \in \mathbb{N}$ such that for every $w \in \Sigma^*$ it holds that $|\mathcal{A}|(w) \in \mathrm{Expr}_N(\mathcal{X})$, that is, the number of alternations of all ground expressions output by $\mathcal{A}$ is uniformly bounded by a constant. A copyless CRA $\mathcal{A}$ is called a bounded alternation copyless CRA (in short BAC) if $\mathcal{A}$ has bounded alternation. All the examples of copyless CRA presented in this paper have bounded alternation. For example, functions in Examples 1 and 2 are part of the BAC-class.

Bounding the alternation of expressions or formulas is a standard assumption in logic [16] and here we used it to syntactically restrict the expression constructed by a copyless CRA. One can easily check that this syntactical property can be verified in NLogSpace in the size of the copyless CRA. Indeed, a copyless CRA has unbounded alternation iff there exists a loop that alternates between $\odot$ and $\oplus$ in its transition graph. Of course, the existence of such loops can be determined by standard reachability tests in NLogSpace [19].

The fact that we can express the BAC automata in Examples 1 and 2 by MP-formulas in Examples 6 and 7 , respectively, is not a coincidence. In the following theorem, we present the main result of the paper.

▸ **Theorem 9.** *Maximal partition logic and bounded alternation copyless CRA are equally expressive, that is:*

- *for every* MP*-formula* φ *there exists a BAC* $\mathcal{A}_\varphi$ *such that* $[\![\varphi]\!] = [\![\mathcal{A}_\varphi]\!]$*;*
- *for every BAC* $\mathcal{A}$ *there exists a* MP*-formula* $\varphi_{\mathcal{A}}$ *such that* $[\![\mathcal{A}]\!] = [\![\varphi_{\mathcal{A}}]\!]$*.*

Proof. We present here sketch proofs of the two directions. Due to the space limit, the full proofs are moved to the appendix (available online).

From logic to automata. Let φ be a MP-formula. We sketch the definition of a BAC $\mathcal{A}$ that specifies the same function as φ. The proof is by induction over the size of φ. The interesting case is when $\varphi = \bigotimes R\langle S\rangle T.\ \psi$ where $R\langle S\rangle T$ is a regular selector and ψ is an MP-formula for which there exists a BAC $\mathcal{B}$ such that $[\![\psi]\!] = [\![\mathcal{B}]\!]$. The main idea behind the definition of $\mathcal{A}$ is to keep many copies of the automaton $\mathcal{B}$ and each copy is responsible for evaluating the formula ψ on intervals defined by $R\langle S\rangle T$. For $\otimes$-aggregating the outputs of the $\mathcal{B}$-copies, $\mathcal{A}$ uses one additional register x^* that, each time an interval is closed, the output of the $\mathcal{B}$-copy is $\otimes$-operated with x^* and then stored in x^*.

Let $\mathcal{A}_R$, $\mathcal{A}_S$, and $\mathcal{A}_T$ be the finite automata recognizing the regular languages R, S, and T, respectively. The first issue we have to deal with is that the number of $\mathcal{B}$-copies cannot depend on the input string w. We prove that, for every k-position in w, the number of maximal intervals defined by $R\langle S\rangle T$ and containing k is uniformly bounded. Moreover this bound is universal for all strings, i.e., it depends only on the size of $\mathcal{A}_S$. To see this, suppose that I is the set of maximal intervals defined by $R\langle S\rangle T$ and containing k. Furthermore, suppose that the size of I is bigger than the number of states in $\mathcal{A}_S$. If we assign to every interval in I the state of $\mathcal{A}_S$ in position k, then there are two intervals i_1 and i_2 with the same state assigned. It is easy to see that we can merge these two intervals into one interval that is selected by $R\langle S\rangle T$ but is bigger than i_1 and i_2. This is clearly a contradiction with the fact that both i_1 and i_2 are maximal.

The second issue is to recognize the maximal intervals selected by $R\langle S\rangle T$ while $\mathcal{A}$ is reading the input. The main observation here is that one can rewrite $R\langle S\rangle T$ into a new regular selector that does not needs maximal semantics, i.e., there exists a regular selector $R'\langle S'\rangle T'$ that defines with the normal-semantics all maximal intervals selected by $R\langle S\rangle T$. Thus, we can assume that $R\langle S\rangle T$ is already in this form and we focus on all selected intervals.

Now that $\mathcal{A}$ does not have to deal with checking whether an interval is maximal or not, it has to decide whether an interval will be selected by $R\langle S\rangle T$. Of course, $\mathcal{A}$ can keep track of runs of $\mathcal{A}_R$, $\mathcal{A}_S$, and $\mathcal{A}_T$ over w to find new potential intervals selected by $R\langle S\rangle T$. The problem is that, in the end, the intervals can turn out to be spurious (e.g. the remaining suffix does not belong to the language defined by T) and we cannot afford to keep all potential intervals since the number of $\mathcal{B}$-copies is bounded. To deal with this issue we use Theorem 2 in [17] which shows that BAC are closed under regular-lookahead, that is, the model can be extended with regular look-ahead and this does not add more expressibility to the model. This extension allows BAC to make decisions based on whether the remaining suffix of the input word belongs to a regular language or not. By using this extension, $\mathcal{A}$ can determine in advance whether an interval is going to be selected by $R\langle S\rangle T$ and solve the problem with the spurious intervals.

The final automaton $\mathcal{A}$ works as follows. Whenever $\mathcal{A}$ finds a new interval selected by $R\langle S\rangle T$, it starts evaluating a $\mathcal{B}$-copy over this interval. With regular look-ahead it also checks if an interval is closing. If that is the case, then the output of the $\mathcal{B}$-copy in charge of this interval is aggregated with the additional register x^* and the registers in this $\mathcal{B}$-copy are reset to the values defined by the initial function of $\mathcal{B}$. Finally, the output function of $\mathcal{A}$ is defined by aggregating x^* with all intervals closed in the last step of $\mathcal{A}$.

From automata to logic. Let $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \delta, q_0, \nu_0, \mu)$ be a bounded alternation copyless CRA. We sketch the definition of the formula $\varphi_{\mathcal{A}}$ that defines the same function as $\mathcal{A}$. The proof is by induction over the alternation bound N of $\mathcal{A}$.

The first step is to understand the ground expressions defined by $\mathcal{A}$. Let g be the ground expression defined by the run of $\mathcal{A}$ on a string w. By applying the associativity and commutativity of $\mathbb{S}$, one can show that g can be rewritten into an expression g^* of the form $\bigotimes_{c \in C} c \otimes \bigotimes_{e \in E} e$ for some operation $\otimes \in \{\oplus, \odot\}$, where $C \subseteq S$ is a multiset of constants and E is a multiset of expressions whose alternation is strictly lower than N. Interestingly, one can define MP-formulas $\varphi_C^{\otimes}$ and $\varphi_E^{\otimes}$ each taking care of $\bigotimes_{c \in C} c$ and $\bigotimes_{e \in E} e$, respectively. To define $\varphi_C^{\otimes}$, we use a set of regular selectors that chooses all 1-letter intervals where each constant in C was generated by a transition of $\mathcal{A}$. Here we define the selectors in such a way that in each position we are able to retrieve the state and substitution used in the run of $\mathcal{A}$. The formula $\varphi_C^{\otimes}$ is then defined by aggregating the right constants (i.e. the ones in C) used by substitutions of the run of $\mathcal{A}$ over w.

The formula $\varphi_E^{\otimes}$ requires more effort. For every expression $e \in E$ we define a BAC $\mathcal{A}_e$, a modified variant of $\mathcal{A}$, such that $\mathcal{A}_e$ outputs e on a substring $w[i_e, j_e]$. We modify only q_0, ν_0, and μ, and the other components $\mathcal{X}$, Q and δ remain the same. Thus, the number of new automata does not depend on the size of E but only on $\mathcal{A}$. Given that the expressions in E have alternation strictly less than N, then by induction we can find a formula $\varphi_{\mathcal{A}_e}$ for every automaton $\mathcal{A}_e$. The main difficulty in the proof is to define regular selectors that find the intervals (i_e, j_e), where $\mathcal{A}_e$ or, more concretely $\varphi_{\mathcal{A}_e}$, must be applied. Indeed, it is easy to define a set of expressions that find these intervals but the problem is the maximal semantic, in particular, the set of intervals $\{(i_e, j_e) \mid e \in E\}$ does not have to be a set of maximal intervals. To solve this problem we define the intervals by rigid formulas instead of using the maximal semantics. By Proposition 8, one can turn a rigid formula into a sum of selectors that define the same set of intervals on every string.

Summing up, having the formulas $\varphi_C^{\otimes}$ and $\varphi_E^{\otimes}$ defined, it is easy to define the final formula $\varphi_{\mathcal{A}}$. Notice that for the base cases of the induction (i.e. when $N = 0, 1$) we do not need the formula $\varphi_E^{\otimes}$ and, therefore, $\varphi_C^{\otimes}$ includes the base case. Of course, there are some exceptional cases not discussed in this proof-sketch because of space restrictions. The full proof includes all these cases. ◂

Theorem 9 gives a logic-based characterization of bounded alternation copyless CRA. This is useful to show new results in the automata model that are implications from the logic counterpart. For example, one can easily show that MP is invariant under the orientation of a word.

▸ **Proposition 10.** *For every formula φ in* MP *there exists a formula φ^r such that for all words $[\![\varphi]\!](w) = [\![\varphi^r]\!](w^r)$, where w^r is the reverse word of w.*

Interestingly, Proposition 10 and Theorem 9 implies that the BAC-class is closed under reverse. Note that this result is unexpected if we try to prove it directly from the automata model.

▸ **Corollary 11.** *For every BAC $\mathcal{A}$ there exists a BAC $\mathcal{A}^r$ that computes the reverse function, that is, $[\![\mathcal{A}]\!](w) = [\![\mathcal{A}^r]\!](w^r)$ for every $w \in \Sigma^*$.*

The logic-based characterization of BAC and its good closure properties suggest that these automata are a robust class in the world of weighted automata. In the next section, we compare its expressibility with respect to weighted MSO and weighted automata.

5 Weighted MSO vs MP

In this section we compare MP with Weighted MSO, a quantitative logic that was proposed as the logic counterpart of weighted automata. Recall that formulas of Weighted MSO [7]

(WMSO) over a semiring $\mathbb{S} = (S, \oplus, \odot, \mathbb{0}, \mathbb{1})$ and an alphabet Σ are defined by the following grammar (note that we use the modern syntax from [4, 14]):

$$\theta ::= \varphi \mid s \mid (\theta \oplus \theta) \mid (\theta \odot \theta) \mid \bigoplus x.\, \theta(x) \mid \bigodot x.\, \theta(x) \mid \bigoplus X.\, \theta(X)$$

where φ is an MSO-formula over Σ, $s \in \mathbb{S}$, x is a first-order variable, and X is a set of variables. The syntax of WMSO is given by boolean formulas (for the MSO fragment) and quantitative formulas (for the rest of the syntax). Let $w = w_1 \dots w_n$ be a string over Σ and σ a $(\bar{x}, w)$-assignment. The semantics $[\![\varphi]\!](w,\sigma)$ of a boolean formula φ over w and σ is equal to $\mathbb{1}$ if $w \models \varphi(\sigma)$ and $\mathbb{0}$ otherwise. The semantics of a quantitative formula θ over w and σ is defined as follows.

$$\begin{aligned}
[\![s]\!](w,\sigma) &:= s \\
[\![(\theta_1 \otimes \theta_2)]\!](w,\sigma) &:= [\![\theta_1]\!](w,\sigma) \otimes [\![\theta_2]\!](w,\sigma) && \text{for } \otimes \in \{\oplus, \odot\} \\
[\![\bigotimes x.\, \theta(x)]\!](w,\sigma) &:= \bigotimes_{i=1}^{n} [\![\theta(x)]\!](w, \sigma[x \to i]) && \text{for } \otimes \in \{\oplus, \odot\} \\
[\![\bigoplus X.\, \theta(X)]\!](w,\sigma) &:= \bigoplus_{I \subseteq [1,n]} [\![\theta(X)]\!](w, \sigma[X \to I])
\end{aligned}$$

▸ **Example 12.** One can compare WMSO with MP by defining WMSO formulas for the functions in Examples 5 and 6. We start with the WMSO-formula for counting the number of b-symbols in a string:

$$\sum x.\, \max\{P_b(x) + 1, 0\} \tag{1}$$

To understand formula (1), recall that in the semiring $\mathbb{N}_{-\infty}(\max, +)$ the operations and constants are defined as follows: $\mathbb{0} = -\infty$, $\mathbb{1} = 0$, $\oplus = \max$ and $\odot = +$. For any position i and assignment $x \to i$, if i is labeled with b then $P_b(x)$ evaluates to 0; otherwise $P_b(x)$ evaluates to $-\infty$. Now it is easy to understand formula (1): we are summing 1 over all positions with a b-symbol and 0 over all other positions.

To define the length of the maximum substring of b-symbols, as in Example 6, one can write the following WMSO-formula:

$$\operatorname{Max} x. \sum y.\, \max\left\{(x \le y \wedge \forall z.(x \le z \wedge z \le y) \to P_b(z)) + 1, 0\right\} \tag{2}$$

The formula (2) selects all pairs (x,y). The boolean subformula is satisfied if (x,y) is an interval of b's; then such a pair contributes 1, otherwise it contributes 0. For a fixed x the formulas sums over all y that vary through all elements of the interval (x,y). Since we take maximum over all variables x, we get the desired formula.

WMSO was proposed by Droste and Gastin as the logic counterpart of weighted automata but it turns out to be more expressive. In [7] it is shown that by restricting the nesting and alternation of semiring quantifiers $\bigoplus x$, $\bigodot x$, and $\bigoplus X$ one can capture exactly the expressiveness of weighted automata. For more details we refer the reader to the paper [7]. We shall use their notation to define different fragments of WMSO. The fragment of WMSO equally expressive to weighted automata is denoted WMSO$[\bigoplus_X \bigodot_x^1]$. Furthermore, in [14] it was shown that two natural fragments of weighted automata, namely, finitely ambiguous weighted automata and polynomial ambigous weighted automata are equally expressive to the fragments denoted respectively by WMSO$[\bigodot_x^1]$ and WMSO$[\bigoplus_x \bigodot_x^1]$. By results in [13, 14] this shows that in terms of expressiveness, these fragments are strictly contained in each other:

$$\text{WMSO}[\bigodot_x^1] \subsetneq \text{WMSO}[\bigoplus_x \bigodot_x^1] \subsetneq \text{WMSO}[\bigoplus_X \bigodot_x^1]$$

We compare the expressiveness of MP with WMSO by exploiting the relation with copyless CRA (Theorem 9). The first question is whether MP is more expressive than $\mathrm{WMSO}[\bigoplus_X \bigodot_x^1]$. At a first sight, one could believe that this is possible, since the syntax of MP is symmetric with respect to both semiring operations, that is, there is no syntactical restriction on $\oplus$- and $\otimes$-quantifiers. Interestingly, in terms of expressiveness, MP is contained in $\mathrm{WMSO}[\bigoplus_X \bigodot_x^1]$. We prove this by showing that functions definable by copyless CRA are also definable by weighted automata. This result combined with Theorem 9 proves the following proposition.

▸ **Proposition 13.** *For every formula in* MP *there exists a formula in* $\mathrm{WMSO}[\bigoplus_X \bigodot_x^1]$ *defining the same function.*

The previous upper-bound opens the question of what is a good lower-bound for the expressiveness of MP. An answer to this question is given in the next result which shows that MP contains the fragment $\mathrm{WMSO}[\bigodot_x^1]$. We prove this result (see the appendix) by showing that every function definable by a finitely ambiguous weighted automaton is definable by a bounded alternation copyless CRA. This combined with the results in [14] and Theorem 9 proves the next proposition.

▸ **Proposition 14.** *For every formula in* $\mathrm{WMSO}[\bigodot_x^1]$ *there exists a formula in* MP *defining the same function.*

The examples presented in Section 3 tell us a bit more about the expressiveness of MP. For example, it was shown in [13] that the function from Example 4 is not definable by any finitely ambiguous weighted automata. This proves that $\mathrm{WMSO}[\bigodot_x^1]$ is strictly contained in MP. On the other hand, in [17] it is shown that there exists a function that is definable by polynomial ambiguous weighted automata but it is not definable by any copyless CRA. This shows that MP is strictly contained in $\mathrm{WMSO}[\bigoplus_X \bigodot_x^1]$ and, moreover, is does not contain $\mathrm{WMSO}[\bigoplus_x \bigodot_x^1]$. Summing up, we get the following diagram representing the expressiveness of MP in terms of WMSO.

$$\begin{tikzcd}
& \mathrm{MP} \arrow[dr, phantom, sloped, "\subsetneq"] & \\
\mathrm{WMSO}[\bigodot_x^1] \arrow[ur, phantom, sloped, "\subsetneq"] \arrow[dr, phantom, sloped, "\subsetneq"] & & \mathrm{WMSO}[\bigoplus_X \bigodot_x^1] \\
& \mathrm{WMSO}[\bigoplus_x \bigodot_x^1] \arrow[u, phantom, sloped, "\not\subseteq"] \arrow[ur, phantom, sloped, "\subsetneq"] &
\end{tikzcd}$$

We conjecture that MP is not contained in $\mathrm{WMSO}[\bigoplus_x \bigodot_x^1]$, such a result would complete the diagram. We guess that this can be shown by proving that the function from Example 2 is not definable by any polynomial ambiguous weighted automata.

6 Conclusions and future work

In this paper we proposed and investigated maximal partition logic. Our main result shows that MP is a logic characterization of BAC, a natural restriction of copyless CRA. MP has no syntactical restrictions and, in contrast to Weighted MSO, there is no division between the boolean and the quantitative parts of the logic. A mild restriction is put in the semantics of the logic since we allow only maximal intervals. Thanks to this semantic our formulas are usually more readable and easy to write (see Example 3).

For future work we would like to extend MP and copyless CRA beyond semirings. It seems that in our proofs we need the commutativity, associativity and the neutral element of each operator separately, but we do not use the distributivity. For this reason we think that we could extend the semiring with additional operators and the results proved in this work will still hold. The comparison of MP with WMSO shows that this logic is in the edge of decidability. It lays between finite ambiguous weighted automata, a class of functions with good decidability properties, and weighted automata for which most interesting problems are undecidable. For this reason, we believe that for future work it is important to understand the decidability properties of MP and copyless CRA.

Acknowledgments

We thank the anonymous referees for their helpful comments. In particular for the comment simplifying the proof of Theorem 9. The first author was supported by Poland's National Science Center grant 2013/09/N/ST6/01170. The last author was supported by CONICYT + PAI / Concurso Nacional Apoyo al Retorno de Investigadores/as desde el extranjero – Convocatoria 2013 + 821320001 and by the Millenium Nucleus Center for Semantic Web Research under grant NC120004.

References

1 V. Alfred. Algorithms for finding patterns in strings. *Handbook of Theoretical Computer Science: Algorithms and complexity*, 1:255, 1990.

2 S. Almagor, U. Boker, and O. Kupferman. What's decidable about weighted automata? In *ATVA*, pages 482–491, 2011.

3 R. Alur, L. D'Antoni, J. Deshmukh, M. Raghothaman, and Y. Yuan. Regular functions and cost register automata. In *Proceedings of the 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 13–22. IEEE Computer Society, 2013.

4 B. Bollig, P. Gastin, B. Monmege, and M. Zeitoun. Logical characterization of weighted pebble walking automata. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, page 19. ACM, 2014.

5 T. Colcombet, C. Ley, and G. Puppis. On the use of guards for logics with data. In *Mathematical Foundations of Computer Science 2011*, pages 243–255. Springer, 2011.

6 K. Culik II and J. Kari. Image compression using weighted finite automata. In *Mathematical Foundations of Computer Science 1993*, pages 392–402. Springer, 1993.

7 M. Droste and P. Gastin. Weighted automata and weighted logics. *Theor. Comput. Sci.*, 380(1-2):69–86, 2007.

8 M. Droste, W. Kuich, and H. Vogler. *Handbook of Weighted Automata.* Springer, 1st edition, 2009.

9 R. Fagin, B. Kimelfeld, F. Reiss, and S. Vansummeren. Spanners: a formal framework for information extraction. In *Proceedings of the 32nd symposium on Principles of database systems*, pages 37–48. ACM, 2013.

10 J. Friedl. *Mastering regular expressions.* " O'Reilly Media, Inc.", 2006.

11 J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

12 B. Kimelfeld. Database principles in information extraction. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 156–163. ACM, 2014.

13 I. Klimann, S. Lombardy, J. Mairesse, and C. Prieur. Deciding unambiguity and sequentiality from a finitely ambiguous max-plus automaton. *Theor. Comput. Sci.*, 327(3):349–373, 2004.

14 S. Kreutzer and C. Riveros. Quantitative monadic second-order logic. In *Proceedings of the 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 113–122. IEEE Computer Society, 2013.

15 D. Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. In *ICALP*, pages 101–112, 1992.

16 L. Libkin. *Elements of finite model theory.* Springer Science & Business Media, 2004.

17 F. Mazowiecki and C. Riveros. On the expressibility of copyless cost register automata. *CoRR*, abs/1504.01709, 2015.

18 M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.

19 C. H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1993.

20 J. Sakarovitch. *Elements of Automata Theory.* Cambridge University Press, 2009.

21 M. P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.

22 A. Weber. Finite-valued distance automata. *Theor. Comput. Sci.*, 134(1):225–251, 1994.

A Proof of Proposition 8

The construction from a regular selector to a rigid formula is already proven in the body of the paper. For the opposite direction, consider the standard encoding of a word w and an $(\{x,y\},w)$-assignment σ over the alphabet $\Sigma_{\{x,y\}} = \Sigma \times \{0,1\}^2$. More precisely, we say that a word $(w,\sigma) \in \Sigma^*_{\{x,y\}}$ *encodes* an $(\{x,y\},w)$-assignment σ if w is the projection of (w,σ) over Σ and for every variable $z \in \{x,y\}$ we have $\sigma(z) = \{i \in \{1,\dots,|w|\} \mid (w,\sigma)[z]_i = 1\}$, where $(w,\sigma)[z]_i$ denotes the i-letter of the projection of (w,σ) over variable z. For the sake of simplification, we denote by $w(i,j)$ the encoding (w,σ) where $\sigma(x) = i$ and $\sigma(y) = j$ and $i \leq j$.

Now, let $\varphi(x,y)$ be a rigid formula and let $\mathcal{A}_\varphi$ be a deterministic finite automaton that is equivalent to $\varphi(x,y)$. More precisely, $w \models \varphi(i,j)$ if, and only if, there is an accepting run of $\mathcal{A}_\varphi$ on $w(i,j)$. Let n be the number of states in $\mathcal{A}_\varphi$. We say that a sequence of nonempty intervals $(i_1,j_1),\dots,(i_m,j_m)$ is *nested* if $(i_k,j_k) \sqsubseteq (i_{k+1},j_{k+1})$ for every $1 \leq k \leq m-1$, and the inclusions are strict. Let $(i_1,j_1),\dots,(i_m,j_m)$ be a nested sequence of intervals selected by $\varphi(x,y)$. We show that since $\varphi(x,y)$ is a rigid formula, then it must hold that $m \leq n$. Indeed, consider the runs of $\mathcal{A}_\varphi$ on $w(i_k,j_k)$. Every run associates a state to every position. Since the intervals are nested, there is a position l that belongs to every interval. If two different runs would associate the same state to the position l that would violate the assumption that $\varphi(x,y)$ is rigid. Thus, $m \leq n$.

The previous argument shows that for intervals selected by $\varphi(x,y)$ the depth of nesting is at most n. This motivates the following sequence formulas for every $1 \leq i \leq n$:

$$\begin{aligned}\varphi_i \;=\; & \varphi(x,y) \;\wedge\; \neg \bigvee_{1\leq j<i} \varphi_j(x,y) \;\wedge \\ & \forall x'.\forall y'.\ (\varphi(x',y') \;\wedge\; x' \leq x \;\wedge\; y \leq y') \rightarrow (x' = x \;\wedge\; y' = y) \vee \bigvee_{1\leq j<i} \varphi_j(x',y'),\end{aligned}$$

Each formula φ_i defines intervals selected by $\varphi(x,y)$ that have depth of nesting exactly i. Obviously $\varphi(x,y) = \bigvee_{1\leq i\leq n} \varphi_i(x,y)$. By definition each formula $\varphi_i(x,y)$ selects maximal intervals which concludes the proof.

B Proof of Theorem 9: from logic to automata

B.1 Union of regular selectors

A *union of regular selectors* (URS) over Σ is a set T of triples. The intervals selected by a set of triples T over $w \in \Sigma^*$ under normal and maximal semantics is defined similar than for a single triple:

$$\begin{aligned}\mathrm{Sel}(w,T) &= \textstyle\bigcup_{t\in T} \mathrm{Sel}(w,t)\\ \mathrm{Max}(w,T) &= \mathrm{Max}_{\sqsubseteq}\big(\mathrm{Sel}(w,T)\big)\end{aligned}$$

It is easy to see that URS can define a richer set of intervals than a single triple. For example, the set $\{a\langle c\rangle a, b\langle c\rangle b\}$ is not equivalent (in terms of selected intervals) to a single triple. In fact, if a triple selects the interval $(2,2)$ on strings aca and bcb, then it must also select $(2,2)$ on strings acb and bca which are not selected by the initial set. We conclude that URS are more expressive than regular selectors.

B.2 Selector automata

We introduce the automaton counterpart of regular selectors. The approach is similar than for RS; we divide the behavior of the automaton in three disjoint components and the intervals selected by the automaton are the one run in the middle component.

A *selector automaton* (SA) is a tuple $\mathcal{S} = (Q, \Sigma, \delta, \delta_\epsilon, q_0, F)$ such that $Q = Q_1 \uplus Q_2 \uplus Q_3$ is a set of state, i.e., Q is divided into three disjoint set of states (called components), $q_0 \in Q_1$ is the initial state, $F \subseteq Q_3$ is the set of final states, $\delta : Q \times \Sigma \rightharpoonup Q$ is a partial transition function such that $\delta(Q_i, a) \subseteq Q_i$ for every $a \in \Sigma^*$ and $i \in \{1, 2, 3\}$, and $\delta_\epsilon : Q_1 \cup Q_2 \to Q_2 \cup Q_3$ is the transition function between components (like ϵ transitions) such that $\delta_\epsilon(Q_i) \subseteq Q_{i+1}$. Note that each substructure (Q_i, Σ, δ_i) defines a disjoint transition system where δ_i is equal to δ when the domain is restricted to Q_i. For a string $w = a_1 \dots a_n$ and an interval (i, j) of w, we define an (i, j)-run ρ of $\mathcal{S}$ over w as a sequence of transitions:

$$q_0 \xrightarrow{a_1} \dots \xrightarrow{a_{i-1}} q_{i-1} \xrightarrow{\epsilon} q^0_{i-1} \xrightarrow{a_i} \dots \xrightarrow{a_j} q_j \xrightarrow{\epsilon} q^0_j \xrightarrow{a_{j+1}} \dots \xrightarrow{a_n} q_n$$

such that $\delta(q_{k-1}, a_k) = p_k$ for $k \in \{1, \dots, n\} - \{i, j+1\}$, $\delta(q^0_{i-1}, a_i) = q_i$, $\delta(q^0_j, a_{j+1}) = q_{j+1}$, $\delta_\epsilon(q_{i-1}) = q^0_{i-1}$ and $\delta_\epsilon(q_j) = q^0_j$. We say that an interval (i, j) of a string w is selected by $\mathcal{S}$ if there exists an (i, j)-run of $\mathcal{A}$ over w as above and $q_n \in F$. Informally, an interval (i, j) of w is selected by $\mathcal{S}$ whenever w can be divided into $u_1 u_2 u_3$, each u_i can be run over (Q_i, Σ, δ_i) and runs can be connected by the relation δ_ϵ. Similar than for regular selectors, we denote by $\mathrm{Sel}(w, \mathcal{S})$ the set of all intervals of w selected by $\mathcal{S}$ and by $\mathrm{Max}(w, \mathcal{S})$ the set of intervals of w under the maximal semantics (i.e $\mathrm{Max}(w, \mathcal{S}) = \mathrm{Max}_{\sqsubseteq}(\mathrm{Sel}(w, \mathcal{S}))$).

B.3 Equivalence between URS and SA

We say that a function $f : \Sigma^* \to 2^{\mathbb{N}^2}$ over Σ is a *selector function* if for every $w \in \Sigma^*$ and $(i, j) \in f(w)$ it holds that $1 \leq i \leq j \leq |w|$. The following lemma shows that selector automata and union of regular selectors define the same class of selector functions.

▸ **Proposition 15.** *For any selector function f, the following conditions are equivalent:*
1. *f is equivalent to a union of regular selectors T.*
2. *f is equivalent to a selector automaton $\mathcal{S}$.*

Proof. Let T be a union of regular selectors. For each triple $t = (R_1, R_2, R_3) \in T$ one can construct a sequence of finite automata $\mathcal{A}_i = (Q_i, \Sigma, \delta_i, q^i_0, F_i)$ for $i \in \{1, 2, 3\}$ such that $\mathcal{L}(R_i) = \mathcal{L}(\mathcal{A}_i)$. Then it is straightforward to show that the selector automaton $\mathcal{S}_t = (Q_1 \uplus Q_2 \uplus Q_3, \Sigma, \delta_1 \uplus \delta_2 \uplus \delta_3, \delta_\epsilon, q^1_0, F_3)$ with $\delta_\epsilon = F_1 \times \{q^2_0\} \cup F_2 \times \{q^3_0\}$ is equivalent to (R_1, R_2, R_3). Finally, by taking the cross product of $\{\mathcal{S}_t \mid t \in T\}$, one can construct a single selector automaton $\mathcal{S}$ that is equivalent to T.

For the other direction, given a selector automaton $\mathcal{S} = (Q, \Sigma, \delta, \delta_\epsilon, q_0, F)$ with $Q = Q_1 \uplus Q_2 \uplus Q_3$, one can pick pair of states $(p_1, p_2) \in Q_1 \times Q_2$ and $(q_2, q_3) \in Q_2 \times Q_3$ such that $\delta_\epsilon(p_1) = p_2$ and $\delta_\epsilon(q_2) = q_3$ and define three disjoint finite automata:
1. $\mathcal{A}_{p_1} = (Q_1, \Sigma, \delta_1, q_0, \{p_1\})$,
2. $\mathcal{A}_{p_2,q_2} = (Q_2, \Sigma, \delta_2, p_2, \{q_2\})$, and
3. $\mathcal{A}_{q_3} = (Q_3, \Sigma, \delta_3, q_3, F)$.

where δ_i is the function δ restricted to Q_i for $i \in \{1, 2, 3\}$. Then for each automaton $\mathcal{A}_{p_1}$, $\mathcal{A}_{p_2,q_2}$, and $\mathcal{A}_{q_3}$ one can construct equivalent regular expressions R_{p_1}, R_{p_2,q_2}, and R_{q_3} by Kleene's theorem. Clearly, the triple $R_{p_1}\langle R_{p_2,q_2}\rangle R_{q_3}$ defines all intervals whose runs in $\mathcal{S}$ pass from q_0 to p_1, from p_2 to q_2, and from q_3 to a final state, respectively. That is, $R_{p_1}\langle R_{p_2,q_2}\rangle R_{q_3}$ is equivalent to the selector automaton $(Q, \Sigma, \delta, \{(p_1, p_2), (q_1, q_2)\}, q_0, F)$. By taking the union of all triples $R_{p_1}\langle R_{p_2,q_2}\rangle R_{q_3}$ for every $(p_1, p_2) \in \delta_\epsilon \cap Q_1 \times Q_2$ and $(q_2, q_3) \in \delta_\epsilon \cap Q_2 \times Q_3$, we can define a union of regular selectors T that is equivalent to $\mathcal{S}$. ◂

B.4 Reduction from maximal to normal semantics

The maximal semantics defined over regular selectors is useful for capturing intervals over strings, but it is inconvenient for selector automata: a run must be compared with other run to check if it is selecting a maximal interval or not. The next result shows that one can always construct a selector automata under normal semantics that selects the same set of intervals than a selector automata under maximal semantics.

▸ **Proposition 16.** *For every selector automaton $\mathcal{S}$, there exists a selector automaton $\mathcal{S}'$ such that:*

$$\mathrm{Max}(w,\mathcal{S}) = \mathrm{Sel}(w,\mathcal{S}')$$

for every string $w \in \Sigma^*$.

Proof. Let $\mathcal{S} = (Q,\Sigma,\delta,\delta_\epsilon,q_0,F)$ be a selector automaton where $Q = Q_1 \uplus Q_2 \uplus Q_3$. The main proof idea is to construct a new selector automaton $\mathcal{S}'$ that simulates $\mathcal{S}$ and in paralellel runs all possibles intervals where the interval (to be selected) could be contained. If there is no interval bigger than the selected one, then $\mathcal{S}'$ outputs the interval. For checking the maximality of an interval, $\mathcal{S}'$ will keep a set of possible runs that could interfere with the current interval. Then $\mathcal{S}'$ will reject an interval (i,j) if there does not exist another interval bigger than (i,j) and, thus, $\mathcal{S}'$ will accept only maximal intervals of $\mathcal{S}$.

Let δ_i be the function δ restricted to Q_i for $i \in \{1,2,3\}$. For any relation δ_i, we define the function $\hat{\delta}_i : 2^{Q_i} \times \Sigma \to 2^{Q_i}$ where $\hat{\delta}_i(C,a) = \{q \in Q_i \mid \exists p \in C.\ \delta_i(p,a) = q\}$ for every $C \subseteq Q_i$ and $a \in \Sigma$. Analogously, we define the function $\hat{\delta}_\epsilon : 2^Q \to 2^Q$ where $\hat{\delta}_\epsilon(C) = \{q \in Q \mid \exists p \in C.\ \delta_\epsilon(p) = q\}$ for every $C \subseteq Q$.

Now, we are ready to define the construction of the selector automaton $\mathcal{S}'$. We first give the whole construction to later explain the meaning of each component. The selector automata $\mathcal{S} = (Q',\Sigma,\delta',\delta_{\epsilon'},q_0',F')$ is defined as follows:

- $Q' = Q_1' \uplus Q_2' \uplus Q_3'$ are the set of states where:

$$\begin{aligned} Q_1' &= Q_1 \times 2^{Q_1} \times 2^{Q_2} \\ Q_2' &= Q_2 \times 2^{Q_2} \times 2^{Q_2} \\ Q_3' &= Q_3 \times 2^{Q_2} \times 2^{Q_3} \end{aligned}$$

- $\delta' = \delta_1' \uplus \delta_2' \uplus \delta_3' : Q' \times \Sigma \to Q'$ where:

$$\begin{aligned} \delta_1'((p,C_1,C_2),a) = (q,D_1,D_2) \quad &\text{iff} \quad \delta_1(p,a) = q,\ D_1 = \hat{\delta}_1(C_1,a),\ \text{and} \\ &\qquad D_2 = \hat{\delta}_2(C_2 \cup \hat{\delta}_\epsilon(C_1),a) \\ \delta_2'((p,C_2,C_2'),a) = (q,D_2,D_2') \quad &\text{iff} \quad \delta_2(p,a) = q,\ D_2 = \hat{\delta}_2(C_2,a),\ \text{and} \\ &\qquad D_2' = \hat{\delta}_2(C_2',a) \\ \delta_3'((p,C_2,C_3),a) = (q,D_2,D_3) \quad &\text{iff} \quad \delta_3(p,a) = q,\ D_2 = \hat{\delta}_2(C_2,a),\ \text{and} \\ &\qquad D_3 = \hat{\delta}_3(C_3,a) \cup \hat{\delta}_\epsilon(D_2), \end{aligned}$$

- $\hat{\delta}_\epsilon' \subseteq Q' \times Q'$ such that:

$$\delta'_\epsilon((p, C_1, C_2)) = (q, D_2, D'_2) \quad \text{if} \quad \delta_\epsilon(p) = q, \; D_2 = \hat{\delta}_\epsilon(C_1) \text{ and } D_2 = C_2$$

$$\delta'_\epsilon((p, C_2, C'_2)) = (q, D_2, D_3) \quad \text{if} \quad \delta_\epsilon(p) = q, \; D_2 = C_2 \cup C'_2 \text{ and } D_3 = \hat{\delta}_\epsilon(C'_2)$$

- $q'_0 = (q_0, \{q_0\}, \varnothing)$, and
- $F' = \{(p, C_2, C_3) \in Q'_3 \mid p \in F \;\wedge\; C_3 \cap F = \varnothing\}$.

In the previous construction, each transition system $(Q'_i, \Sigma, \delta'_i)$ simulates in the first component of a state a run of $\mathcal{S}$. We call the state in the first component the current run. The rest of the components of every state keep information of possible runs whose selected interval could contained the interval of the current run. Specifically, $(Q'_1, \Sigma, \delta'_1)$ stores in the second component the last state of runs of (Q_1, Σ, δ) (i.e. subset construction) and in a the third component the last state of runs that already pass to (Q_2, Σ, δ_2), that is, runs whose intervals were open. Then $(Q'_2, \Sigma, \delta'_2)$ keeps in the second component the last state of runs that open an interval at the same time than the current run and in the third component the last state of runs that open their interval before the current run. We must save all runs that open an interval at the same time than the current run given that, if another run captures the same interval than the current run, this do not interfere with respect to the maximal semantics. Finally, $(Q'_3, \Sigma, \delta'_3)$ stores in the second component the last state of runs that have not closed yet their interval and in the third component the last state of runs whose potential selected interval contains the interval of the current run.

For the proof of correctness of $\mathcal{S}'$ (i.e. $\mathrm{Sel}(w, \mathcal{S}') = \mathrm{Max}(w, \mathcal{S})$ for all $w \in \Sigma^*$), follows directly from a standard inductive argument over the run of $\mathcal{S}$ and $\mathcal{S}'$. ◂

B.5 Cost register automata with regular look-ahead

Let REG_Σ be the set of all regular languages over Σ. A CRA-RLA is a tuple $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \delta, q_0, \nu_0, \mu)$ where Q, Σ, $\mathcal{X}$, q_0, ν_0, and μ are defined as before and $\delta : Q \times \mathrm{REG}_\Sigma \rightharpoonup Q \times \mathrm{Subs}(\mathcal{X})$ is a partial transition function with finite domain restricted as follows: for a fixed state q let $\delta(q, L_1) = (q_1, \sigma_1), \delta(q, L_2) = (q_2, \sigma_2), \ldots, \delta(q, L_k) = (q_k, \sigma_k)$ be all transitions with q in the first coordinate and $L_1, \ldots, L_k \in \mathrm{REG}_\Sigma$. Then the languages $L_1, \ldots, L_k$ are pairwise disjoint (i.e. $L_i \cap L_j = \varnothing$). Note that the last requirement forces that on a given string the automaton $\mathcal{A}$ is deterministic. Given a string $w = a_1 \ldots a_n \in \Sigma^*$, the run of $\mathcal{A}$ over w is a sequence of configurations:

$$(q_0, \nu_0) \xrightarrow{L_1} (q_1, \nu_1) \xrightarrow{L_2} \ldots \xrightarrow{p_n} (L_n, \nu_n)$$

such that for $1 \leq i \leq n$, $\delta(q_{i-1}, L_i) = (q_i, \sigma_i)$, $w[i, \cdot] \in L_i$ and $\nu_i(x) = [\![\nu_{i-1} \circ \sigma_i(x)]\!]$ for each $x \in \mathcal{X}$. The output of $\mathcal{A}$ over w is defined as usual, i.e. $[\![\mathcal{A}]\!](w) = [\![\hat{\nu}_n(\mu(q_n))]\!]$. From the restriction on the transition function it is easy to see each string w has exactly one run. The extension of the classes of copyless and bounded alternation CRA-RLA are defined similarly.

B.6 Proof of Theorem 9 (Logic to automata)

For this direction, we show by structural induction over the size of the formula how to construct for each MP-formula φ a BAC $\mathcal{A}_\varphi$ such that $[\![\varphi]\!](w) = [\![\mathcal{A}_\varphi]\!](w)$ for every $w \in \Sigma^*$. Without lost of formality, we will say that φ is equivalent to $\mathcal{A}_\varphi$ or, also, that φ is computable by $\mathcal{A}_\varphi$. For the base case, if $\varphi = c$ where $c \in \mathbb{S}$ the proof is straightforward. For the induction, we suppose that for every MP-subformula ψ of φ there exists a BAC $\mathcal{A}_\psi$ equivalent to ψ and we show that the same holds for φ.

In the next lemma we prove the easy case when φ is the binary operation of two MP-subformulas.

▸ **Lemma 17.** *Let $\otimes$ be any of the two operations $\oplus$ or $\odot$. If $\varphi = \psi_1 \otimes \psi_2$ and each formula ψ_i is computable by a BAC for $i \in \{1,2\}$, then φ is computable by a BAC.*

Proof. Given disjoint BACs $\mathcal{A}_1 = (Q_1, \mathcal{X}_1, \delta_1, q_1^0, \nu_1^0, \mu_1)$ and $\mathcal{A}_2 = (Q_2, \mathcal{X}_2, \delta_2, q_2^0, \nu_2^0, \mu_2)$ that computes ψ_1 and ψ_2, respectively, one can construct a BAC that runs $\mathcal{A}_1$ and $\mathcal{A}_2$ in parallel (cross product of the states) and finally outputs the $\otimes$-operation of both outputs. Formally, define $\mathcal{A} = (Q_1 \times Q_2, \mathcal{X}_1 \uplus \mathcal{X}_2, \delta, (q_1^0, q_2^0), \nu^0, \mu)$. The initial function ν^0 is defined as $\nu^0(x) = \nu_i^0(x)$ whenever $x \in \mathcal{X}_i$ for $i \in \{1,2\}$. Then for every pair of transitions $\delta_1(q_1, a) = (q_1', \sigma_1)$ and $\delta_2(q_2, a) = (q_2', \sigma_2)$ we define $\delta((q_1,q_2),a) = ((q_1',q_2'), \sigma_1 \uplus \sigma_2)$. Finally, the output function μ is defined by $\mu((q_1,q_2)) = \mu_1(q_1) \otimes \mu_2(q_2)$ By construction, it is straightforward to show that $\mathcal{A}$ computes the same function as φ. ◂

The interesting case is when $\varphi = \bigotimes \mathtt{R}.\ \psi$ where $\mathtt{R}$ is a regular selector and ψ is any MP-formula. We suppose that ψ is computable by a BAC $\mathcal{A}_\psi$ and we show how to construct a BAC that computes the same function as φ. For the sake of presentation, we divide this construction in two steps. First, we define what we call an $(\otimes, f)$-aggregation automaton for a function f over strings and show that every $(\otimes, f)$-aggregation automaton can be computed by a BAC (Lemma 18) when f is defined by a BAC. The second step is to show how to convert φ into an $(\otimes, f)$-aggregation automaton (Lemma 19). The purpose of using this "middle-step" model (i.e. $(\otimes, f)$-aggregation automata) is to simplify the construction of the final BAC. This will be clear during the proof.

For the rest of this section, let $\otimes$ be any of the two operations $\oplus$ or $\odot$ and $\mathbb{0}^*$ its respectively neutral element (i.e. $s \otimes \mathbb{0}^* = \mathbb{0}^* \otimes s = s$ for all $s \in S$). The goal of an $(\otimes, f)$-aggregation automaton $\mathcal{G}$ for a function $f : \Sigma^* \to \mathbb{S}$ is to select intervals from the input, computes the function f over each interval and aggregates the outputs with the $\otimes$-operation. For defining the selection mechanism of $\mathcal{G}$, we use patterns (i.e. strings over $\{0,1\}$) to specify a set of intervals over a string. More precisely, given a pattern $\vec{b} = b_1 \dots b_n \in \{0,1\}^+$ we define the set of intervals $\mathrm{I}(\vec{b})$ defined by $\vec{b}$ as follows: $(i,j) \in \mathrm{I}(\vec{b})$ if, and only if, $b_i \dots b_j \in \{1\}^+$ and $b_{i-1} = 0$ ($b_{j+1} = 0$) whenever $1 < i$ ($j < n$, resp.). In other words, $(i,j) \in \mathrm{I}(\vec{b})$ is a maximal subsequence of contiguous 1's in $\vec{b}$. Furthermore, for $k \in \mathbb{N}$ and $\vec{b} = b_1 \dots b_n \in (\{0,1\}^k)^*$ we define the set of intervals $\mathrm{I}(\vec{b}) = \bigcup_{i \le k} \mathrm{I}(\pi_i(\vec{b}))$ where $\pi_i(p) = b_1(i) \dots b_n(i)$ (i.e. the string projected into its i-component).

For a function $f : \Sigma^* \to \mathbb{S}$ an $(\otimes, f)$-aggregation automaton (with regular look-ahead) is a tuple $\mathcal{G} = (Q, \Sigma, N, \delta, q_0)$ such that Q is the set of states, Σ is the finite alphabet, N is a positive number, $\delta : Q \times \mathrm{REG}_\Sigma \rightharpoonup Q \times \{0,1\}^N$ is a partial transition function with finite domain, and q_0 is the initial state. Similar than for CRA with regular look-ahead, we suppose that for a fixed state q, if $\delta(q, L_1) = (q_1, \sigma_1), \delta(q, L_2) = (q_2, \sigma_2), \dots, \delta(q, L_k) = (q_k, \sigma_k)$ are all transitions with q in the first coordinate, then the regular languages $L_1, \dots, L_k$ are pairwise disjoint. Given a string $w = a_1 \dots a_n \in \Sigma^*$, the run of $\mathcal{G}$ over w is a sequence of states and transitions:

$$q_0 \xrightarrow{L_1/b_1} q_1 \xrightarrow{L_2/b_2} \dots \xrightarrow{L_n/b_n} q_n$$

such that $\delta(q_{i-1}, L_i) = (q_i, b_i)$ and $w[i, \cdot] \in L_i$ for every $1 \le i \le n$. Let $\vec{b} = b_1 b_2 \dots b_n$ be the sequence in $(\{0,1\}^k)^*$ generated by the above run. Then the output of $\mathcal{A}$ over $w \in \Sigma^*$ is defined as follows:

$$[\![\mathcal{G}]\!](w) = \bigotimes_{(i,j)\in \mathrm{I}(\vec{b})} f(w[i,j])$$

Note that in the above definition, $f : \Sigma^* \to \mathbb{S}$ could be any function from strings to $\mathbb{S}$. As the following results shows, any $(\otimes, f)$-aggregation automaton can be defined by a BAC whenever the function f is defined by a BAC as well.

▸ **Lemma 18.** *Let $\mathcal{G}$ be an $(\otimes, f)$-aggregation automaton and $f : \Sigma^* \to \mathbb{S}$ is definable by a BAC. Then there exists a BAC with regular look-ahead $\mathcal{A}$ such that $[\![\mathcal{G}]\!](w) = [\![\mathcal{A}]\!](w)$ for any $w \in \Sigma^*$.*

Proof. Suppose that $\mathcal{G} = (Q, \Sigma, N, \delta, q_0)$ is an $(\otimes, f)$-aggregation automaton and $\mathcal{A} = (P, \Sigma, \mathcal{X}, \gamma, p_0, \nu_0, \mu)$ is a BAC such that f is equivalent to $[\![\mathcal{A}]\!]$. The idea of this proof is to construct a BAC $\mathcal{A}'$ that for each input runs $\mathcal{G}$ for determining the valid intervals and, in parallel, runs $\mathcal{A}$ over each interval produced by $\mathcal{G}$. For aggregating the outputs of $\mathcal{A}$, we use a new register x^* that collects the values $[\![\mathcal{A}]\!](w[i,j])$ for all intervals (i,j) selected by $\mathcal{G}$. Finally, we show that the constructed CRA is copyless and with bounded alternation. For the sake of presentation, in this proof we will show the construction for $N = 1$. For $N \geq 2$ the construction is similar but with the overhead of keeping more copies of $\mathcal{A}$ in parallel.

Let $\mathcal{A}' = (Q', \Sigma, \mathcal{X}', \delta', q_0', \nu_0', \mu')$ be a BAC such that each component is defined as follows:

- $Q' = Q \times (P \cup \{\bot\})$ is the set of states where $\bot$ is a fresh element such that $\bot \notin P$.
- $\mathcal{X}' = \mathcal{X} \cup \{x^*\}$ is the set of registers where x^* is a fresh register with $x^* \notin \mathcal{X}$.
- $q_0' = (q_0, \bot)$ is the initial state.
- ν_0' is initialization assignment defined for every $x \in \mathcal{X}'$ as (recall that $\mathbb{0}^*$ is the neutral element of $\otimes$):

$$\nu_0'(x) = \begin{cases} \mathbb{0}^* & \text{if } x = x^* \\ \nu_0(x) & \text{otherwise} \end{cases}$$

- μ' is the output function defined for every $(q,p) \in Q'$ as:

$$\mu'((q,p)) = \begin{cases} x^* & \text{if } p = \bot \\ x^* \otimes \mu(p) & \text{otherwise} \end{cases},$$

- δ' is the transition function defined for every $(q,p) \in Q'$ and $a \in \Sigma$. Suppose that $\delta(q, L) = (q', b)$ where $b \in \{0,1\}$. Depending on p and b, we have to distinguish four different cases:
  - if $p = \bot$ and $b = 0$, then we define $\delta'((q,\bot), L) = ((q', \bot), \sigma')$ such that $\sigma'(x) = x$ for every $x \in \mathcal{X}'$.
  - if $p = \bot$ and $b = 1$, then we define $\delta'((q,\bot), L) = ((q', p'), \sigma')$ where $\gamma(p_0, a) = (p', \sigma)$ and for every $x \in \mathcal{X}'$:

$$\sigma'(x) = \begin{cases} x^* & \text{if } x = x^* \\ \sigma(x) & \text{otherwise} \end{cases}$$

  - if $p \in P$ and $b = 0$, then we define $\delta'((q,p), L) = ((q', \bot), \sigma')$ where σ' is defined for every $x \in \mathcal{X}'$ as follows:

$$\sigma'(x) = \begin{cases} x^* \otimes \mu(r) & \text{if } x = x^* \\ \nu_0(x) & \text{otherwise} \end{cases}$$

  - if $p \in P$ and $b = 1$, then we define $\delta'((q,p), L) = ((q', p'), \sigma')$ where $\gamma(p, a) = (p', \sigma)$ and for every $x \in \mathcal{X}'$:

$$\sigma'(x) = \begin{cases} x^* & \text{if } x = x^* \\ \sigma(x) & \text{otherwise} \end{cases}$$

From the construction, it is straightforward to prove by induction over the size of the string and by case analysis that $\mathcal{A}'$ computes the same function as $\mathcal{G}$. Moreover, one can easily check that $\mathcal{A}'$ is copyless and has bounded alternation. In fact, $\mathcal{A}'$ is copyless from the definition of δ' and μ' and, if $\mathcal{A}$ has bounded alternation, then $\mathcal{A}'$ also has bounded alternation given that $\mathcal{A}'$ always $\otimes$-operates x^* with a register of $\mathcal{A}$ that is of bounded alternation. This concludes the proof. ◂

The previous lemma shows that for every $(\otimes, f)$-aggregation automaton $\mathcal{G}$ where f is computable by a BAC, there exists a BAC with regular lookahead that computes the same function. By using Theorem 2 from [17], we can conclude that there exists a BAC without look-ahead that computes $\mathcal{G}$. Thus, the last part of the proof is to show that any formula $\varphi = \bigotimes \mathtt{R}.\ \psi$ can be computed by an $(\otimes, f)$-aggregation automaton. The next lemma states this formally.

▸ **Lemma 19.** *Let $\varphi = \bigotimes \mathtt{R}.\ \psi$ be a* MP*-formula with $\mathtt{R}$ a regular selector and ψ a* MP*-formula computable by a BAC. Then there exists an $(\otimes, f)$-aggregation automaton $\mathcal{G}$ with f computable by a BAC $\mathcal{A}$ such that $[\![\varphi]\!](w) = [\![\mathcal{G}]\!](w)$ for every $w \in \Sigma^*$.*

Proof. Suppose that $\mathtt{R} = R\langle S\rangle T$ is a BAC that computes the same function as ψ. By Propositions 15 and 16, let $\mathcal{S} = (Q, \Sigma, \delta, \delta_\epsilon, q_0, F)$ where $Q = Q_1 \uplus Q_2 \uplus Q_3$ be the selector automaton equivalent to the regular selection $\mathtt{R}$ such that for every $w \in \Sigma^*$:

$$\text{Max}(w, R\langle S\rangle T) = \text{Sel}(w, \mathcal{S}).$$

The last fact simplifies the construction since we do not need to take care of the maximal semantics and we can focus on just the intervals selected by $\mathcal{S}$.

Let $\mathcal{A}$ be a BAC that computes the function f. The plan is to construct, from $\mathcal{A}$ and $\mathcal{S}$, an $(\otimes, f)$-aggregation automaton $\mathcal{G}'$ that computes the same function as φ. As the reader could already suspect, f will be defined by $\mathcal{A}$ and $\mathcal{G}'$ will be responsible of running $\mathcal{S}$. For this purpose, we need to introduce a set of regular languages that will be used to predict when $\mathcal{S}$ is going to select an interval. Let δ_i be the function δ restricted to Q_i for $i \in \{1, 2, 3\}$. Given a state $p \in Q_1$ and $q \in Q_2$ we define the following NFAs with ϵ-transitions:

$$\begin{array}{rcll} \mathcal{S}_p^{23} & = & (Q, \Sigma, \delta_2 \uplus \delta_3 \uplus \delta_\epsilon, p, F) & \quad\text{and} \\ \mathcal{S}_q^{3} & = & (Q_2 \uplus Q_3, \Sigma, \delta_3 \uplus \delta_\epsilon, q, F) & \end{array}$$

Notice that $\mathcal{S}_p^{23}$ and $\mathcal{S}_q^{3}$ do not contain transitions that reaches the states p and q, respectively. Intuitively, the language recognized by $\mathcal{S}_p^{23}$ are all strings such that there is an interval starting at the first position that will be selected by $\mathcal{S}$ if it is currently at p. The language of $\mathcal{S}_p^{23}$ will be used for verifying (by regular look-ahead) whether an interval will be open or not. Similarly, the intuition behind $\mathcal{S}_q^{3}$ are all strings such that a run of $\mathcal{S}$ that already opened an interval and is in state q will closed it. $\mathcal{S}_p^{23}$ will be used for deciding (by regular look-ahead) whether an interval is closing or not.

Before going into the construction of the $(\otimes, f)$-aggregation automaton, we need to state some facts related to $\mathcal{S}$ and the partial runs that reaches Q_2. Let $w = a_1 \ldots a_n$ be a string in Σ^*. A partial run ρ of $\mathcal{S}$ over w is a sequence $q_0 \xrightarrow{b_1} \ldots \xrightarrow{b_m} q_m$ such that $b_i \in \Sigma \cup \{\epsilon\}$, $w = b_1 \cdot b_2 \cdot \ldots \cdot b_m$ and for every $i \leq m$ it holds that $\delta(q_{i-1}, b_i) = q_i$ or $\delta_\epsilon(q_{i-1}) = q_i$. Notice that the definition of ρ is the standard definition of a run when we see $\mathcal{A}$ as an NFA with ϵ-transition. Furthermore, from the restriction imposed to δ and δ_ϵ one can check that at most two b_i can be equal to ϵ. For a run ρ like above we denote by ρ^f the state at the head of the run (i.e. q_m) in ρ and by $\text{PRun}_{\mathcal{S}}(w)$ the set of all partial runs of $\mathcal{S}$.

The maximality of the intervals selected by $\mathcal{S}$ imposed some restriction on the set of possible partial runs of $\mathcal{S}$ and the states at the head of the runs. For any $w \in \Sigma^*$, let $\mathrm{PRun}_{\mathcal{S}}^{Q_2}(w)$ be the set of all runs in $\mathrm{PRun}_{\mathcal{S}}(w)$ that ends in Q_2, that is, $\mathrm{PRun}_{\mathcal{S}}^{Q_2}(w) = \{\rho \in \mathrm{PRun}_{\mathcal{S}}(w) \mid \rho^f \in Q_2\}$. We make the following two claims about the set $\mathrm{PRun}_{\mathcal{S}}^{Q_2}(w)$:

▸ Claim 20. For every string $w \in \Sigma^*$, it holds that:

1. $|\mathrm{PRun}_{\mathcal{S}}^{Q_2}(w)| \leq |Q_2|$.
2. $\mathcal{L}(\mathcal{S}^3_{\rho^f}) \cap \mathcal{L}(\mathcal{S}^3_{\tau^f}) = \varnothing$ for $\rho, \tau \in \mathrm{PRun}_{\mathcal{S}}^{Q_2}(w)$.

Proof. We first show that set $\mathrm{PRun}_{\mathcal{S}}^{Q_2}(w)$ cannot contain two partial runs ρ and τ that end in the same state, that is, it cannot happen that $\rho^f = \tau^f$. Indeed, if ρ and τ end in the same state, then one can extend w with a string u such that $\mathrm{Sel}(w \cdot u, \mathcal{S})$ will contain two intervals (i_1, j_1) and (i_2, j_2) with $(i_1, j_1) \sqsubseteq (i_2, j_2)$ which contradict the maximality of the intervals selected by $\mathcal{S}$. Then each partial run in $\mathrm{PRun}_{\mathcal{S}}^{Q_2}(w)$ ends in a different state which implies that $|\mathrm{PRun}_{\mathcal{S}}^{Q_2}(w)| \leq |Q_2|$. By the same argument, we can show that if $p = \rho^f$ and $q = \tau^f$ for $\rho, \tau \in \mathrm{PRun}_{\mathcal{S}}^{Q_2}(w)$, then the languages $\mathcal{L}(\mathcal{S}^3_p)$ and $\mathcal{L}(\mathcal{S}^3_q)$ must be disjoint. If not, then take $u \in \mathcal{L}(\mathcal{S}^3_p) \cap \mathcal{L}(\mathcal{S}^3_q)$ and define the string $w \cdot u$. By the definition of $\mathcal{S}^3_p$ and $\mathcal{S}^3_q$, $\mathcal{S}$ will select two intervals of $w \cdot u$ such that one will be contained in the other which is a contradiction again with the maximality of the intervals selected by $\mathcal{S}$. ◂

For the construction of $\mathcal{G}'$, we use partial functions to encode the different threads of the automaton $\mathcal{S}$ that are running in parallel. By the previous claim, we only need to consider at most $|Q_2|$ different runs of $\mathcal{S}$ at each position of the string. Furthermore, the languages $\mathcal{L}(\mathcal{S}^3_p)$ for each head state p are pairwise disjoint. Formally, we define the set $\mathcal{H}$ of all the partial functions $h : \{1, \ldots, |Q_2|\} \to Q_2$ such that $\mathcal{L}(\mathcal{S}^3_{h(i)}) \cap \mathcal{L}(\mathcal{S}^3_{h(j)}) = \varnothing$ for all $i, j \in \mathrm{dom}(h)$. For any $h \in \mathcal{H}$ we define the tuple $\vec{h} \in \{0,1\}^{|Q_2|}$ such that, for every $i \leq |Q_2|$, $\vec{h}(i) = 1$ iff $i \in \mathrm{dom}(h)$. The idea here is that the range of h will track the head states of runs currently in Q_2 and the domain of h (specifically, $\vec{h}$) will be the pattern-symbols generated by $\mathcal{G}'$.

We have all the ingredients to define the $(\otimes, f)$-aggregation automaton $\mathcal{G}'$ that computes the same function as φ. Specifically, for $f = [\![\mathcal{A}]\!]$ let $\mathcal{G}' = (Q', \Sigma, |Q_2|, \delta', q_0')$ be an $(\otimes, f)$-aggregation automaton where each component is defined as follows.

- $Q' = Q_1 \times \mathcal{H}$ is the set of states.
- $q_0' = (\{q_0\}, h_\varnothing)$ where $h_\varnothing$ is the partial function in $\mathcal{H}$ with empty domain (i.e. $\mathrm{dom}(h_\varnothing) = \varnothing$).
- δ' is the transition function defined for every $(q, h) \in Q'$ and $a \in \Sigma$ as follows. Let $i^* = \min\{i \mid i \notin \mathrm{dom}(h)\}$, that is, i^* is an unused entry in h. For each state $(p, h) \in Q'$ and each letter $a \in \Sigma$ we define the following transitions (with regular look-ahead):
  - If $i \in \mathrm{dom}(h)$ and:
  $$L \;=\; \mathcal{L}(a \cdot \Sigma^*) \;\cap\; \mathcal{L}(\mathcal{S}^{23}_p) \;\cap\; \mathcal{L}(\mathcal{S}^3_{h(i)})$$
  (i.e. L is verifying that a is the next letter, there is an interval starting at this position, and the interval marked at the i-position in the pattern should be closed), then $\delta'((p, h), L) = ((p', h'), \vec{h}')$ such that $\delta(p, a) = p'$, $\mathrm{dom}(h') = (\mathrm{dom}(h) \setminus \{i\}) \cup \{i^*\}$ and for every $j \in \mathrm{dom}(h')$:
  $$h'(j) \;=\; \begin{cases} \delta_\epsilon(p) & \text{if } j = i^* \\ \delta(h(j), a) & \text{otherwise} \end{cases}$$

- If $i \in \mathrm{dom}(h)$ and:
$$L \;=\; \mathcal{L}(a\cdot\Sigma^*) \;\cap\; \mathcal{L}(\mathcal{S}_p^{23})^C \;\cap\; \mathcal{L}(\mathcal{S}_{h(i)}^3)$$
(i.e. L is verifying that a is the next letter, there is no interval starting at this position, and the interval marked at the i-position in the pattern should be closed), then $\delta'((p,h),L) = ((p',h'),\vec{h}')$ such that $\delta(p,a) = p'$, $\mathrm{dom}(h') = (\mathrm{dom}(h) \setminus \{i\})$ and $h'(j) = \delta(h(j),a)$ for every $j \in \mathrm{dom}(h')$.
- If:
$$L \;=\; \mathcal{L}(a\cdot\Sigma^*) \;\cap\; \mathcal{L}(\mathcal{S}_p^{23}) \;\cap\; \bigcap_{i\in\mathrm{dom}(h)} \mathcal{L}(\mathcal{S}_{h(i)}^3)^C$$
(i.e. L is verifying that a is the next letter, there is an interval starting at this position but there is no interval closing at this position), then $\delta'((p,h),L) = ((p',h'),\vec{h}')$ such that $\delta(p,a) = p'$, $\mathrm{dom}(h') = \mathrm{dom}(h) \cup \{i^*\}$ and for every $j \in \mathrm{dom}(h')$:
$$h'(j) \;=\; \begin{cases} \delta_\epsilon(p) & \text{if } j = i^* \\ \delta(h(j),a) & \text{otherwise} \end{cases}$$
- If:
$$L \;=\; \mathcal{L}(a\cdot\Sigma^*) \;\cap\; \mathcal{L}(\mathcal{S}_p^{23})^C \;\cap\; \bigcap_{i\in\mathrm{dom}(h)} \mathcal{L}(\mathcal{S}_{h(i)}^3)^C$$
(i.e. L is checking that a is the next letter, there is no interval starting or closing at this position), then $\delta'((p,h),L) = ((p',h'),\vec{h}')$ such that $\delta(p,a) = p'$, $\mathrm{dom}(h') = \mathrm{dom}(h)$ and $h'(j) = \delta(h(j),a)$ for every $j \in \mathrm{dom}(h')$.

First of all, one can easily check that the regular look-ahead at each state is well-defined. Indeed, by the definition of δ' and Claim 20, the regular languages are disjoint for every transition at a state $(p,h) \in Q'$. For showing that the $\mathcal{G}'$ selects all the intervals selected by $\mathcal{S}$, one can easily prove by induction over the length of a string w that each interval in $\mathrm{Sel}(\mathcal{S},w)$ is mentioned in the output pattern produced by $\mathcal{G}'$. ◂

C Proof of Theorem 9: from automata to logic

C.1 Parse trees

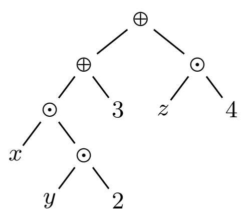
We introduce standard notation for trees that will be useful during this proof. Let Σ be a set of labels. An (unordered) labeled Σ-tree t is a finite function $t : \mathrm{nodes}(t) \to \Sigma$ such that $\mathrm{nodes}(t)$ is a finite prefix-closed subset of $\mathbb{N}^*$, that is, $w \in \mathrm{nodes}(t)$ whenever $w\cdot i \in \mathrm{nodes}(t)$ for some $i \in \mathbb{N}$. Additionally we assume that $w\cdot i$ implies $w\cdot(i-1) \in \mathrm{nodes}(t)$ for $i > 0$. We say that the ϵ-word is the root of t and that $w\cdot i \in \mathrm{nodes}(t)$ is a child of w. Further, we write $\mathrm{labels}(t)$ to denote the set labels of a tree t. For any node $w \in \mathrm{nodes}(t)$, we denote by $t[w]$ the subtree rooted at w, i.e. the labeled tree $t[w] : \mathrm{nodes}(t[w]) \to \Sigma$ such that $t[w](i) = t(w\cdot i)$ for every $i \in \mathrm{nodes}(t[w])$. We usually write $a\{t_1,\ldots,t_k\}$ to denote a tree whose root is labeled by a and $t_1,\ldots,t_k$ are the subtrees hanging from the root (i.e. for every $i \le k$ there exists $j \in \mathbb{N}$ such that $t[j] = t_i$). We say that $w \in \mathrm{nodes}(t)$ is an internal node of t if $w\cdot i \in \mathrm{nodes}(t)$ for some $i \in \mathbb{N}$. Otherwise, w is called a leaf of t. The set of all leaves of t is denoted by $\mathrm{leaves}(t)$. We say that a tree is *complete* if any internal nodes has at least two children. One can easily check that if t is a complete tree, then $|\mathrm{nodes}(t)| \le 2\cdot|\mathrm{leaves}(t)|$. Finally, we denote by $\mathrm{Trees}(\Sigma)$ the set of all Σ-trees.

Let $\mathrm{labs}_{\mathbb{S}} = S \cup \{\oplus,\odot\} \cup \mathcal{X}$ be the a set of labels in $\mathbb{S}$ and variables $\mathcal{X}$. A parse tree is a complete labeled $\mathrm{labs}_{\mathbb{S}}$-tree $\mathbb{p}$ whose leaves are labeled by $S \cup \mathcal{X}$ and internal nodes by $\{\oplus,\odot\}$.

We denote by $\mathrm{Parse}(\mathcal{X})$ the set of all $\mathrm{lab}_{\mathbb{S}}$-parse trees. For any expression $e \in \mathrm{Expr}(\mathcal{X})$, the parse tree of e is a tree $\mathbb{p}_e$ recursively defined as $\mathbb{p}_e = e$ whenever e is equal to a constant or variable, and $\mathbb{p}_e = \otimes\{\mathbb{p}_{e_1}, \mathbb{p}_{e_2}\}$ whenever $e = e_1 \otimes e_2$ where $\otimes \in \{\oplus, \odot\}$.

Conversely, any parse tree $\mathbb{p}$ can be converted into an equivalent expression $\exp(\mathbb{p})$. If $\mathbb{p}$ is a single node then $e = \mathbb{p}$ otherwise if $\mathbb{p} = \otimes\{\mathbb{p}_1, \ldots, \mathbb{p}_k\}$ for $\otimes \in \{\oplus, \odot\}$ then $\exp(\mathbb{p}) = \bigotimes_{1 \le i \le k} \exp(\mathbb{p}_i)$. So far we used $\oplus$ and $\odot$ as binary operations thus the equivalent expressions are not defined uniquely. We allow for this because this definition is unique up to the equivalence of the expressions since $\oplus$ and $\odot$ are commutative and associative. We define the **size of an expression** e as the number of nodes in the parse tree $\mathbb{p}_e$.

▸ **Example 21.** Consider the expression $e = ((x_0 \odot (y \odot 2)) \oplus 3) \oplus (z \odot 4)$ where $x, y, z \in \mathcal{X}$ and $2, 3, 4 \in S$. One can easily check that the parse tree $\mathbb{p}_e$ of e is the following:



Internal nodes of a parse tree can be merged by applying associativity and commutativity of $\oplus$ and $\odot$, respectively. Formally, for any label $\otimes \in \{\oplus, \odot\}$ define the flattening function $\mathrm{flat}_{\otimes}$ such that $\mathrm{flat}_{\otimes}(\circledast\{\mathbb{p}_1, \ldots, \mathbb{p}_k\})$ is equal to $\{\mathbb{p}_1, \ldots, \mathbb{p}_k\}$ whenever $\otimes = \circledast$ and $\circledast\{\mathbb{p}_1, \ldots, \mathbb{p}_k\}$ otherwise, for any label $\circledast \in \{\oplus, \odot\}$ and trees $\mathbb{p}_1, \ldots, \mathbb{p}_k$. Then, given a parse tree $\mathbb{p}$ we denote by $\mathbb{p}^*$ the reduced parse tree constructed recursively as follows: $\mathbb{p}^* = \otimes\{\mathrm{flat}_{\otimes}(\mathbb{p}_1^*), \ldots, \mathrm{flat}_{\otimes}(\mathbb{p}_k^*)\}$ whenever $\mathbb{p} = \otimes\{\mathbb{p}_1, \ldots, \mathbb{p}_k\}$ and $\mathbb{p}^* = \mathbb{p}$ whenever $\mathbb{p}^*$ is equal to a variable or constant. By the construction of $\mathbb{p}^*$, the label of any node of $\mathbb{p}^*$ is different to all root labels of its children, i.e. $\mathbb{p}^*(\epsilon) \neq \mathbb{p}^*(i)$ for any $i \in \mathrm{nodes}(\mathbb{p}^*) \cap \mathbb{N}$. Or equivalently if $\otimes \in \{\oplus, \odot\}$ is a label of a internal node then it does not have a child labeled with $\otimes$.

▸ **Proposition 22.** *Let p be a parse tree. The expressions* $\exp(\mathbb{p})$ *and* $\exp(\mathbb{p}^*)$ *are equivalent and the alternation of these expressions is the same.*

Proof. We prove this by induction on the depth of p. If the depth is 0 then $\mathbb{p} = \mathbb{p}^*$ by definition. Let $\mathbb{p} = \otimes\{\mathbb{p}_1, \ldots, \mathbb{p}_k\}$ and $\mathbb{p}^* = \otimes\{\mathrm{flat}_{\otimes}(\mathbb{p}_1^*), \ldots, \mathrm{flat}_{\otimes}(\mathbb{p}_k^*)\}$. Let $\mathbb{p}' = \otimes\{\mathbb{p}_1^*, \ldots, \mathbb{p}_k^*\}$. By the induction assumption we know that $\exp(\mathbb{p})$ and $\exp(\mathbb{p}')$ are equivalent and the alternation of these expressions is the same. To prove the proposition we need to show that this also holds between $\exp(\mathbb{p}')$ and $\exp(\mathbb{p}^*)$. Let $\mathbb{p}_i^* = \circledast\{\mathbb{p}_{i,1}, \ldots, \mathbb{p}_{i,k_i}\}$. It suffices to prove that for every i the parse trees $\mathbb{p}'$ and $\mathbb{p}^{i,*} = \otimes\{\mathbb{p}_1^*, \ldots, \mathrm{flat}_{\otimes}(\mathbb{p}_i^*), \ldots, \mathbb{p}_k^*\}$ define equivalent expressions with the same alternation. If $\otimes \neq \circledast$ then these parse trees are the same. Otherwise $\mathbb{p}^{i,*} = \otimes\{\mathbb{p}_1^*, \ldots, \mathbb{p}_{i,1}, \ldots, \mathbb{p}_{i,k_i}, \ldots, \mathbb{p}_k^*\}$. Then by definition

$$\exp(\mathbb{p}') = \bigotimes_j \exp(\mathbb{p}_j^*) \approx \bigotimes_{j \neq i} \exp(\mathbb{p}_i^*) \otimes \exp(\mathbb{p}_i^*) = \bigotimes_{j \neq i} \exp(\mathbb{p}_i^*) \otimes \bigotimes_j \exp(\mathbb{p}_{i,j}) \approx \mathbb{p}^{i,*}$$

where $\approx$ is the relation of equivalent expressions. These equalities follow from the commutativity and associativity of $\otimes$.

We show that $\mathbb{p}'$ and $\mathbb{p}^{i,*}$ have the same alternation. Formally the definition of alternation was for binary operators $\oplus$ and $\odot$. It is easy to generalize it for the $\bigotimes$ notation. We define $f_{\oplus}(e) = 1$ if $e = \bigodot_j e_j$, $f_{\odot}(e) = 1$ if $e = \bigoplus_j e_j$ and $f(e) = 0$ otherwise. The alternation

Alt is defined as $\mathrm{Alt}(c) = 0$ for every $c \in S$ and $\mathrm{Alt}(\bigotimes_j e_j) = \max_j\{\mathrm{Alt}(e_j) + f_{\otimes}(e_j)\} + 1$ for $\otimes \in \{\oplus, \odot\}$ and any expressions e_j over $\mathbb{S}$.

$$\mathrm{Alt}(\mathbb{p}') = \max_j\{\mathrm{Alt}(exp(\mathbb{p}_j^*)) + f_{\otimes}(\exp(\mathbb{p}_j^*))\} + 1 =$$
$$\max\{\max_{j\neq i}\{\mathrm{Alt}(exp(\mathbb{p}_j^*)) + f_{\otimes}(\exp(\mathbb{p}_j^*))\}, \mathrm{Alt}(exp(\mathbb{p}_i^*)) + f_{\otimes}(\exp(\mathbb{p}_i^*))\} + 1$$

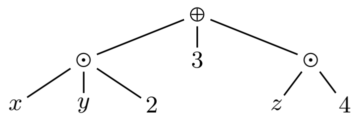
Since $f_{\otimes}(\exp(\mathbb{p}_i^*)) = 0$ then this is equivalent to

$$\max\{\max_{j\neq i}\{\mathrm{Alt}(exp(\mathbb{p}_j^*)) + f_{\otimes}(\exp(\mathbb{p}_j^*))\}, \mathrm{Alt}(exp(\mathbb{p}_i^*))\} + 1 =$$
$$\max\{\max_{j\neq i}\{\mathrm{Alt}(exp(\mathbb{p}_j^*)) + f_{\otimes}(\exp(e_j))\}, \max_j\{\mathrm{Alt}(\mathbb{p}_{i,j}) + f_{\otimes}(\mathbb{p}_{i,j})\}\} + 1 = \mathrm{Alt}(\mathbb{p}^{i,*})$$

◂

By Proposition 22 the alternation of an expression e is equivalent to the depth of its reduced parse tree $\mathbb{p}_e^*$. From now, we will implicitly assume that parse trees are always in their reduced form (i.e. $\mathbb{p} = \mathbb{p}^*$ for every $\mathbb{p} \in \mathrm{Parse}(\hat{\mathcal{X}} \cup \mathcal{Y})$).

▸ **Example 23.** Recall the parse tree $\mathbb{p}_e$ of the expression e. The following tree is the reduced parse tree $\mathbb{p}_e^*$:

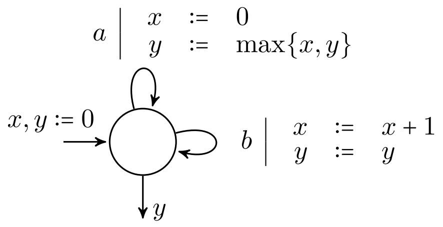


C.2 Main construction

Let $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \delta, q_0, \nu_0, \mu)$ be a BAC automaton and let N be its bound on alternations. Let $(q_0, \varsigma_0) \xrightarrow{a_1} \dots \xrightarrow{a_n} (q_n, \varsigma_n)$ be a sequence of ground configurations, where $\delta(q_{i-1}, a_i) = (q_i, \sigma_i)$, $\varsigma_0 = \nu_0$ and $\varsigma_i(x) = \hat{\varsigma}_{i-1}(\sigma_i(x))$. The ground expression of this run is defined as $g = \hat{\varsigma}_n(\mu(q_n))$. We call the sequence ς_i the *substitution sequence*.

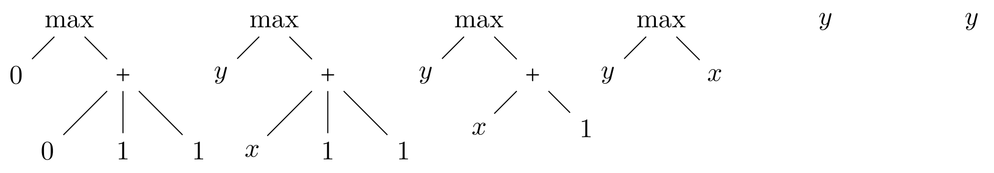
The ground expression is built by composing the substitutions in a “bottom-up” fashion. Next, we present an alternative definition where the substitution are composed top-down by starting from the end. Let $g_{n+1} = \mu(q_n)$. We define $g_i = \hat{\sigma}_i(g_{i+1})$ for $0 \le i \le n$, where $\sigma_0 = \nu_0$. It is clear that $g = g_0$. We call the sequence g_i the **ground sequence**. In this definition the expressions g_i, for $i > 0$, do not have to be ground expressions. Since all expressions $\mu(q_n), \sigma_i$ are copyless then every expression g_i is also copyless. Thus in every expression g_i there are no multiple occurrences of the same registers.

▸ **Example 24.** Consider a BAC automaton $\mathcal{C}$, which is a modified version of the automaton from Example 1.



Comparing to Example 1 the automaton $\mathcal{C}$ has the output function changed to $\mu_{\mathcal{C}}(q) = y$. This way it outputs the longest sequence of b's except for the last sequence which does not precede the letter a. Since there is only one state each substitution is determined by a letter. Let us denote the substitutions by σ_a, σ_b; and by σ_0 the initial valuation. We present the ground sequence of a run of $\mathcal{C}$ on the word $bbab$. Below each g_i we draw its parse tree. Notice that the ground sequence grows from the back by applying the substitutions to the variables.

$$\begin{tikzcd} g_0 & g_1 \arrow[l, "\sigma_0"'] & g_2 \arrow[l, "\sigma_b"'] & g_3 \arrow[l, "\sigma_b"'] & g_4 \arrow[l, "\sigma_a"'] & g_5 \arrow[l, "\sigma_b"'] \end{tikzcd}$$



Let $\otimes \in \{\oplus, \odot\}$. For simplicity we write $\overline{\otimes}$ for the other operation (i.e. $\{\otimes, \overline{\otimes}\} = \{\oplus, \odot\}$) and $id_\otimes$ to denote the identity of $\otimes$ in $\mathbb{S}$ (i.e. $id_\oplus = \mathbb{0}$ and $id_\odot = \mathbb{1}$). Recall that for every expression e we denote by $\mathbb{p}_e$ its reduced parse tree and conversely for every parse tree $\mathbb{p}$ we denote by $\exp(\mathbb{p})$ its corresponding expression. The parse trees of ground expression do not have variable labels in the leaves. We denote by $\mathrm{Root}(e) \in \{\oplus, \odot, \mathit{single}\}$ the label of the root of the parse tree. If the tree has one node and its label is a constant or a variable then we use *single*. By Proposition 22 we can assume that the parse trees are in their reduced form. That is $\mathbb{p}_e$ is either a single node labeled with a variable or a constant; or $\mathbb{p}_e$ has a root labeled with $\otimes \in \{\oplus, \odot\}$ and $\mathbb{p}_e = \otimes\{\mathbb{p}_1, \dots, \mathbb{p}_n, s_1, \dots, s_k, v_1, \dots, v_m\}$, where v_i are variables, s_i are constants and $\mathbb{p}_i$ are subtrees such that $\mathrm{Root}(\mathbb{p}_i) = \overline{\otimes}$. Let us assume that $\mathbb{p}_e$ is of the latter form. We divide the expression e into three parts: $\mathrm{SubExp}(e) = \{\exp(\mathbb{p}_1), \dots, \exp(\mathbb{p}_n)\}$, $\mathrm{SubConst}(e) = \{s_1, \dots, s_k\}$ and $\mathrm{SubVar}(e) = \{v_1, \dots, v_m\}$. For constants and subexpressions we use multisets to capture repetitions. Given a set P of parse trees, we denote by $\otimes P$ the $\otimes$-aggregation of the elements in P. Then we get the following equation:

$$e = (\otimes \,\mathrm{SubExp}(e)) \otimes (\otimes \,\mathrm{SubConst}(e)) \otimes (\otimes \,\mathrm{SubVar}(e)).$$

Since we work with copyless expressions, a set is enough for variables to have this equation.

For simplicity, if $\mathrm{Root}(e) = \otimes$, we write $\mathrm{S}_\otimes\{e\} = \otimes \,\mathrm{SubConst}(e)$ and $\mathrm{P}_\otimes\{e\} = \otimes \,\mathrm{SubExp}(e)$, that is, $\mathrm{S}_\otimes\{e\}$ is the aggregation of all constants in $\mathrm{SubConst}(e)$ and $\mathrm{P}_\otimes\{e\}$ is the aggregation of all subexpressions in $\mathrm{SubExp}(e)$. Notice that if the expression e is ground then:

$$e = \mathrm{S}_\otimes\{e\} \otimes \mathrm{P}_\otimes\{e\}. \tag{3}$$

This partition is crucial for the proof. The idea is to divide the function defined by $\mathcal{A}$ into two parts. To do this we shall analyze the ground expressions of the runs represented as in (3). Consider $\mathcal{C}$ from Example 24. The intuition of partitioning the ground expressions g of a run of $\mathcal{C}$ is that $\mathrm{S}_{\max}\{g\}$ aggregates some 0's that appear during the run below the root; and $\mathrm{P}_{\max}\{g\}$ aggregates the sums of the sequences of b's.

For every expression e we define a subset of its variables $\mathrm{Var}_\otimes(e) \subseteq \mathrm{Var}(e)$ as follows.

$$\mathrm{Var}_\otimes(e) = \begin{cases} \mathrm{Var}(e) & \text{if } \mathrm{Root}(e) = \mathit{single} \\ \mathrm{SubVar}(e) & \text{if } \mathrm{Root}(e) = \otimes \\ \varnothing & \text{if } \mathrm{Root}(e) = \overline{\otimes} \end{cases}$$

Let us look at the sample ground sequence in Example 24. For most i we have $\mathrm{Var}(g_i) = \mathrm{Var}_{\max}(g_i)$. The exceptional cases are for $i = 1,2$, where $\mathrm{Var}(g_i) = \{x,y\}$ and $\mathrm{Var}_{\max}(g_i) = \{y\}$, because the variable x is not directly below the root. The intuition is that in g_1 and g_2 the variable x cannot contribute new constants to the final expression $\mathrm{S}_\otimes\{g_0\}$; and it cannot create new expressions for $\mathrm{P}_\otimes\{g_0\}$ it only extends the old ones. We shall formalize this intuition in Claim 27; but first we need to introduce some definitions.

The fact that ground expressions are built by composing from the back motivates a definition *track automata* that help us keep track of the ground sequences. We define a nondeterministic finite automaton $\mathcal{T}^\otimes$ that keeps track of the run of $\mathcal{A}$. Consider a run of $\mathcal{A}$

$$(p_0,\nu_0),\dots,(p_n,\nu_n)$$

where $\nu_i(x) = [\![\nu_{i-1}\circ\sigma_i(x)]\!]$. Let $g_{n+1},g_n,\dots,g_0$ be the ground sequence of this run. The idea is to have a finite automaton $\mathcal{T}^\otimes$, which has a unique run on the same word of the form:

$$(p_0,\mathrm{Var}_\otimes(g_1),\mathrm{Var}(g_1)),\dots,(p_n,\mathrm{Var}_\otimes(g_{n+1}),\mathrm{Var}(g_{n+1})).$$

Formally, we define $\mathcal{T}^\otimes = (Q\times\mathcal{P}(\mathcal{X})\times\mathcal{P}(\mathcal{X}),\Sigma,\delta^\otimes,Q_0,F)$ as the nondeterministic finite automaton obtained from $\mathcal{A}$. The transition relation is defined as follows:

$$((q,A,B),a,(p,A',B'))\in\delta^\otimes$$

if there exists a substitution σ such that $\delta(q,a) = (p,\sigma)$ and $A = \bigcup_{x\in A'}\mathrm{Var}_\otimes(\sigma(x))$, $B = \bigcup_{x\in B'}\mathrm{Var}(\sigma(x))$. It remains to define the initial and accepting states of $\mathcal{T}^\otimes$. For every state $q\in Q$ let $A_q = \mathrm{Var}_\otimes(\mu(q))$, $B_q = \mathrm{Var}(\mu(q))$. We define the set of final states

$$F = \{(q,A_q,B_q) \mid \text{for all } q\in Q\},$$

and the set of initial states

$$Q_0 = \{(q_0,A,B) \mid \text{for all sets } A,B\subseteq\mathcal{X}\}.$$

We call the automaton $\mathcal{T}^\otimes$ the *track automaton of* $\mathcal{A}$. The states of track automata have three components. We shall refer to them respectively as q-component, A-component and B-component.

Consider a run of $\mathcal{T}^\otimes$ on a word $w = w_1\dots w_n$.

$$(p_0,A_0,B_0),(p_1,A_1,B_1),\dots,(p_n,A_n,B_n) \tag{4}$$

where $((p_{i-1},A_{i-1},B_{i-1}),w_i,(p_i,A_i,B_i))\in\delta^\otimes$. By definition of $\mathcal{T}^\otimes$ for every i there exists a substitution σ_i, such that $\delta(p_{i-1},w_i) = (p_i,\sigma_i)$. By definition of initial states in $\mathcal{T}^\otimes$ we know that $p_0 = q_0$. Thus the run of $\mathcal{A}$ on w is:

$$(p_0,\nu_0),\dots,(p_n,\nu_n) \tag{5}$$

where $\nu_i(x) = [\![\nu_{i-1}\circ\sigma_i(x)]\!]$. Since $\mathcal{A}$ is deterministic, this proves that for all runs of $\mathcal{T}^\otimes$ on a given word w the first component of the states is always deterministic. We show more than that.

▶ Claim 25. The automaton $\mathcal{T}^\otimes$ is unambiguous and accepts Σ^* and thus each word has its unique run in $\mathcal{T}^\otimes$. Moreover, if $g_{n+1},g_n,\dots,g_0$ is the ground sequence defining the ground expression for the run of $\mathcal{A}$ on w, then

$$A_i = \mathrm{Var}_\otimes(g_{i+1}), \tag{6}$$
$$B_i = \mathrm{Var}(g_{i+1}) \tag{7}$$

Proof. Recall that $g_{n+1} = \mu(p_n)$ and $g_i = \sigma_i \circ g_{i+1}$, where $\sigma_0 = \nu_0$. We show (6, 7) by induction down from n to 0, which proves that $\mathcal{T}^{\otimes}$ is unambiguous.

For the final configuration (p_n, A_n, B_n) we have $A_n = A_{p_n} = \mathrm{Var}_{\otimes}(\mu(p_n))$ and $B_n = B_{p_n} = \mathrm{Var}(\mu(p_n))$ by definition. Let $i < n$ and let $\delta(p_i, w_{i+1}) = (p_{i+1}, \sigma_{i+1})$. Then by definition $A_i = \bigcup_{x \in A_{i+1}} \mathrm{Var}_{\otimes}(\sigma_{i+1}(x))$ and $B_i = \bigcup_{x \in B_{i+1}} \mathrm{Var}(\sigma_{i+1}(x))$. On the other side:

$$\begin{array}{lclcll}
\mathrm{Var}_{\otimes}(\sigma_{i+1} \circ g_{i+2}) & = & \displaystyle\bigcup_{x \in \mathrm{Var}_{\otimes}(g_{i+2})} \mathrm{Var}_{\otimes}(\sigma_{i+1}(x)) & = & \displaystyle\bigcup_{x \in A_{i+1}} \mathrm{Var}_{\otimes}(\sigma_{i+1}(x)) & = \; A_i \\
\mathrm{Var}(\sigma_{i+1} \circ g_{i+2}) & = & \displaystyle\bigcup_{x \in \mathrm{Var}(g_{i+2})} \mathrm{Var}(\sigma_{i+1}(x)) & = & \displaystyle\bigcup_{x \in B_{i+1}} \mathrm{Var}(\sigma_{i+1}(x)) & = \; B_i
\end{array}$$

The equalities come from the induction assumption. Since $g_{i+1} = \sigma_{i+1} \circ g_{i+2}$ this proves that $A_i = \mathrm{Var}_{\otimes}(g_{i+1})$ and $B_i = \mathrm{Var}(g_{i+1})$. It remains to prove that the automaton $\mathcal{T}^{\otimes}$ recognizes Σ^*. Consider the run of $\mathcal{A}$ on w:

$$(p_0, \nu_0), \ldots, (p_n, \nu_n)$$

where $\nu_i(x) = [\![\nu_{i-1} \circ \sigma_i(x)]\!]$ and $g_{n+1}, g_n, \ldots, g_0$ is the ground sequence. It is easy to check that the sequence:

$$(p_0, \mathrm{Var}_{\otimes}(g_1), \mathrm{Var}(g_1)), \ldots, (p_n, \mathrm{Var}_{\otimes}(g_{n+1}), \mathrm{Var}(g_{n+1}))$$

is an accepting run of $\mathcal{T}^{\otimes}$ on w. ◀

Let $s, s' \in Q \times \mathcal{P}(\mathcal{X}) \times \mathcal{P}(\mathcal{X})$. By $\mathcal{T}^{\otimes}_{s \to s'}$ we denote the modified automaton $\mathcal{T}^{\otimes}$, where s is the only initial state and s' is the only accepting state. By $R^{\otimes}_{s \to s'}$ we denote the regular expression equivalent to $\mathcal{T}^{\otimes}_{s \to s'}$. Recall that final states in $\mathcal{T}^{\otimes}$ are of the form $s_q = (q, A_q, B_q)$. The automata $\mathcal{T}^{\otimes}_{(q_0, A_0, B_0) \to s_q}$ form a partition of the automaton $\mathcal{T}^{\otimes}$ (up to the automata that recognize the empty language). Since the automaton $\mathcal{T}^{\otimes}$ is unambiguous and recognizes Σ^* then the automata $\mathcal{T}^{\otimes}_{(q_0, A_0, B_0) \to s_q}$ are also unambiguous and the languages $L(\mathcal{T}^{\otimes}_{(q_0, A_0, B_0) \to s_q})$ form a partition of Σ^*. The regular expressions corresponding to these languages shall be crucial for the final formula to extract the information about the run of $\mathcal{A}$.

First, let us come back to the definition of $\mathrm{S}_{\otimes}\{e\}$ and $\mathrm{P}_{\otimes}\{e\}$. Previously we defined these expressions assuming that $\mathrm{Root}(e) = \otimes$. Let us give a general definition for all kinds of expressions.

▸ **Definition 26.** We define two expressions based on e.

$$\mathrm{S}_{\otimes}\{e\} = \begin{cases} \otimes\,\mathrm{SubConst}(e) & \text{if } \mathrm{Root}(e) = \otimes \\ e & \text{if } e \text{ is a constant} \\ id_{\otimes} & \text{otherwise} \end{cases} \qquad \mathrm{P}_{\otimes}\{e\} = \begin{cases} \otimes\,\mathrm{SubExp}(e) & \text{if } \mathrm{Root}(e) = \otimes \\ e & \text{if } \mathrm{Root}(e) = \overline{\otimes} \\ id_{\otimes} & \text{otherwise} \end{cases}$$

By a case analysis of Definition 26 it is easy to see that if e is ground then $e = \mathrm{S}_{\otimes}\{e\} \otimes \mathrm{P}_{\otimes}\{e\}$. If $\mathrm{Root}(e) = \otimes$ then we observed this in (3). Otherwise $\mathrm{S}_{\otimes}\{e\} \otimes \mathrm{P}_{\otimes}\{e\}$ reduces to $e \otimes id_{\otimes}$ or $id_{\otimes} \otimes e$.

▸ Claim 27. Consider the runs on w from (4, 5), where $\delta(p_{i-1}, w_i) = (p_i, \sigma_i)$ and $g_{n+1}, g_n, \ldots, g_0$ is the ground sequence of the run. Let us assume that $\mathrm{Root}(g) = \otimes$. The following equalities hold:

$$\mathrm{S}_{\otimes}\{g_i\} = \mathrm{S}_{\otimes}\{g_{i+1}\} \otimes \bigotimes_{x \in A_i} (\mathrm{S}_{\otimes}\{\sigma_i(x)\}),$$

$$\mathrm{P}_{\otimes}\{g_i\} = \big(\sigma_i \circ \mathrm{P}_{\otimes}\{g_{i+1}\}\big) \otimes \bigotimes_{x \in A_i} (\mathrm{P}_{\otimes}\{\sigma_i(x)\}).$$

Proof. Set i. Recall that $g_i = \sigma_i \circ g_{i+1}$. First let us consider the case when $\mathrm{Root}(g_{i+1}) = \otimes$. Let $\mathbb{p}_{g_{i+1}} = \otimes\{\mathbb{p}_1, \dots, \mathbb{p}_n, s_1, \dots, s_k, v_1, \dots, v_m\}$. The parse tree $\mathbb{p}_{g_i}$ comes from $\mathbb{p}_{g_{i+1}}$, where one substitutes the variables x with the trees $\mathbb{p}_{\sigma(x)}$ and then flattens the tree. The variables are divided into two sets, $\mathrm{Var}_{\otimes}(g_{i+1}) = \{v_1, \dots, v_m\}$; and the remaining variables in the subtrees $\mathbb{p}_i$. Notice that σ_i does not change the leaves s_i. By (6) $A_i = \{v_1, \dots, v_m\}$. The expression $\mathrm{S}_{\otimes}\{g_i\}$ aggregates previous constants $\mathrm{S}_{\otimes}\{g_{i+1}\}$ with the new constants. It remains to show that new constants come from $\mathrm{S}_{\otimes}\{\sigma_i(x)\}$ for every $x \in A_i$. Indeed, $\sigma_i(x)$ adds new constants only if the root of $\sigma_i(x)$ is $\otimes$ or $\sigma_i(x)$ is itself a constant. The new constants are precisely $\mathrm{S}_{\otimes}\{\sigma_i(x)\}$, which proves the first equality.

Similarly the substitution σ_i does not change the internal nodes, thus the expression $\mathrm{P}_{\otimes}\{g_i\}$ comes from extending the previous subtrees $\mathbb{p}_i$ by applying σ_i to the variables in the leaves of $\mathbb{p}_i$, this results in $\sigma_i \circ \mathrm{P}_{\otimes}\{g_{i+1}\}$. They are aggregated with new subtrees, which for every $x \in A_i$ come from $\mathrm{P}_{\otimes}\{\sigma_i(x)\}$. This proves the second equality.

Let us look at the remaining cases. Notice that since $\mathrm{Root}(g) = \otimes$, then it is not possible that $\mathrm{Root}(g_{i+1}) = \overline{\otimes}$ for any i. The remaining case is when $\mathrm{Root}(g_{i+1}) = \mathit{single}$. Then g_{i+1} is a single node. Since we assumed that $\mathrm{Root}(g) = \otimes$ then this single node must be a variable; moreover $|A_i| = 1$ by (6). Then by definition $\mathrm{S}_{\otimes}\{g_{i+1}\} = \mathrm{P}_{\otimes}\{g_{i+1}\} = id_{\otimes}$ and $\mathrm{S}_{\otimes}\{g_i\} = \mathrm{S}_{\otimes}\{\sigma_i(x)\}$, $\mathrm{P}_{\otimes}\{g_i\} = \mathrm{P}_{\otimes}\{\sigma_i(x)\}$, where $A_i = \{x\}$. This concludes the proof of the claim. ◀

From now on, we work with the runs (4,5) and we shall assume that $\mathrm{Root}(g) = \otimes$. Recall that $g = g_0$, $g_{n+1} = \mu(p_n)$ and $\sigma_0 = \nu_0$. Applying iteratively the first equality from Claim 27 to $\mathrm{S}_{\otimes}\{g_i\}$, we get the following.

$$\begin{aligned}
&\mathrm{S}_{\otimes}\{g\} = \mathrm{S}_{\otimes}\{g_0\} = \mathrm{S}_{\otimes}\{g_1\} \otimes \bigotimes_{x \in A_0} (\mathrm{S}_{\otimes}\{\sigma_0(x)\}) = \dots = \\
&\mathrm{S}_{\otimes}\{g_{n+1}\} \otimes \bigotimes_{0 \le i \le n} \Big(\bigotimes_{x \in A_i} (\mathrm{S}_{\otimes}\{\sigma_i(x)\}) \Big) = \\
&\mathrm{S}_{\otimes}\{\mu(p_n)\} \otimes \bigotimes_{0 \le i \le n} \Big(\bigotimes_{x \in A_i} (\mathrm{S}_{\otimes}\{\sigma_i(x)\}) \Big) = \\
&\qquad\qquad \mathrm{S}_{\otimes}\{\mu(p_n)\} \otimes \bigotimes_{1 \le i \le n} \Big(\bigotimes_{x \in A_i} (\mathrm{S}_{\otimes}\{\sigma_i(x)\}) \Big) \otimes \bigotimes_{x \in A_0} \nu_0(x).
\end{aligned} \tag{8}$$

Recall the definition of the substitution sequence $\varsigma_0 = \nu_0 = \sigma_0$ and $\varsigma_i(x) = \varsigma_{i-1} \circ \sigma_i(x)$. The substitution ς_i defines the expressions in the registers after reading i letters. For simplicity we assume that $\varsigma_{-1} = id_{\otimes}$. Applying iteratively the second equality from Claim 27 we get the following.

$$\begin{aligned}
&\mathrm{P}_{\otimes}\{g_0\} = \big(\sigma_0 \circ \mathrm{P}_{\otimes}\{g_1\}\big) \otimes \bigotimes_{x \in A_0} (\mathrm{P}_{\otimes}\{\sigma_0(x)\}) = \\
&\big(\underbrace{\sigma_0 \circ \sigma_1}_{\varsigma_1} \circ \mathrm{P}_{\otimes}\{g_2\}\big) \otimes \bigotimes_{x \in A_1} \big(\underbrace{\sigma_0}_{\varsigma_0} \circ \mathrm{P}_{\otimes}\{\sigma_1(x)\}\big) \otimes \bigotimes_{x \in A_0} (\mathrm{P}_{\otimes}\{\sigma_0(x)\}) = \dots = \\
&\big(\varsigma_n \circ \mathrm{P}_{\otimes}\{g_{n+1}\}\big) \otimes \bigotimes_{0 \le i \le n} \Big(\bigotimes_{x \in A_i} (\varsigma_{i-1} \circ \mathrm{P}_{\otimes}\{\sigma_i(x)\}) \Big) = \\
&\qquad\qquad \big(\varsigma_n \circ \mathrm{P}_{\otimes}\{\mu(p_n)\}\big) \otimes \bigotimes_{1 \le i \le n} \Big(\bigotimes_{x \in A_i} (\varsigma_{i-1} \circ \mathrm{P}_{\otimes}\{\sigma_i(x)\}) \Big)
\end{aligned} \tag{9}$$

Notice that when aggregating the final expression in (9) we drop the case when $i = 0$. This is because $\sigma_0(x) = \nu_0(x)$ is a constant for every x and thus by definition $\mathrm{P}_{\otimes}\{\sigma_0(x)\} = id_{\otimes}$. Recall that by (3) we have $g = \mathrm{S}_{\otimes}\{g\} \otimes \mathrm{P}_{\otimes}\{g\}$. Thus (8,9) give as a new representation of g.

We show how to use this representation of g. We define the formula $\psi_{\mathcal{A}}$ in partition logic that defines the same function on words as the automaton $\mathcal{A}$. The definition is by induction over N, i.e., the number of alternations in $\mathcal{A}$. Recall that by (3) $g = \mathrm{S}_{\otimes}\{g\} \otimes \mathrm{P}_{\otimes}\{g\}$. We define a formulas that separately evaluate to $\mathrm{S}_{\otimes}\{g\}$ and $\mathrm{P}_{\otimes}\{g\}$. The formula below is an aggregation of many subformulas. Each subformula has some parameters determined by the unique run of $\mathcal{A}$ on a word. Thus, as we shall see, there is at most one subformula different from $id_{\otimes}$.

$$\psi^{\otimes}_{\mathcal{A}} = \bigotimes_{s_0,s_q} \bigotimes(\epsilon)\langle R^{\otimes}_{s_0\to s_q}\rangle(\epsilon).\ \mathrm{S}^{\otimes}_{s_0,s_q} \otimes \mathrm{P}^{\otimes}_{s_0,s_q}, \tag{10}$$

where s_0, s_q vary through all initial and accepting states of the track automaton $\mathcal{T}^{\otimes}$. We use the notation $s_q = (q, A_q, B_q)$ $s_0 = (q_0, A_0, B_0)$ and in general $s_i = (q_i, A_i, B_i)$. For simplicity we denote by $Q^{\otimes}$ the set of states of $\mathcal{T}^{\otimes}$, so $s_i \in Q^{\otimes}$. Since $L(R^{\otimes}_{s_0\to s_q})$ are a partition of Σ^* then for a given word w this formula is equivalent to $\otimes(\epsilon)\langle R^{\otimes}_{s_0\to s_q}\rangle(\epsilon).\ \mathrm{S}^{\otimes}_{s_0,s_q} \otimes \mathrm{P}^{\otimes}_{s_0,s_q}$ for some s_0 and s_q (the other subformulas evaluate to $id_{\otimes}$). We define $\mathrm{S}^{\otimes}_{s_0,s_q}$ as follows.

$$\mathrm{S}^{\otimes}_{s_0,s_q} = \mathrm{S}_{\otimes}\{\mu(q)\} \otimes \bigotimes_{s_1,s_2\in Q^{\otimes},a\in\Sigma} \left(\bigotimes(R^{\otimes}_{s_0\to s_1})\langle a\rangle(R^{\otimes}_{s_2\to s_q}).\ \bigotimes_{x\in A_2} \mathrm{S}_{\otimes}\{\sigma(x)\} \right) \otimes \bigotimes_{x\in A_0} \nu_0(x),$$

additionally we require for every pair s_1, s_2 that $\delta(q_1, a) = (q_2, \sigma)$, where q_i are q-components of s_i. Similarly, A_2 is the A-component of s_2. The intervals here are 1-letter words. Since there is a unique run in $\mathcal{T}^{\otimes}_{s_0\to s_q}$ for a given word w, then for every w_i there is a unique (s_1, a, s_2) transition such that $(R^{\otimes}_{s_0\to s_1})\langle a\rangle(R^{\otimes}_{s_2\to s_q})$ selects w_i. This determines the unique transition $\delta(q_1, w_i) = (q_2, \sigma)$ in the run of $\mathcal{A}$ when reading w_i. It is easy to verify that this formula outputs the same value as the expression $\mathrm{S}_{\otimes}\{g\}$ presented in (8). The difference is that in (8) we aggregate constants with $\bigotimes_{1\le i\le n}\bigotimes_{x\in A_i}$, where i varies through all transitions in the run. In the formula $\mathrm{S}^{\otimes}_{s_0,s_q}$ we also select every transition but we group the transitions by the states and letters that determine them.

The definition of the formula $\mathrm{P}^{\otimes}_{s_0,s_q}$ requires more effort and we need to present some definitions first. The remaining part of the proof is mostly devoted to define the formula $\mathrm{P}^{\otimes}_{s_0,s_q}$.

We start from defining three different automata such that their ground expressions of the runs are subtrees of the ground expressions of the runs in $\mathcal{A}$ with $\overline{\otimes}$ in the root. For every transition $t = (p, a, p', \sigma) \in \delta$ and a register $x \in \mathcal{X}$ if $\mathrm{Root}(\sigma(x)) = \overline{\otimes}$ we define the automaton $\mathcal{A}^{\otimes}[\mu_{(t,x)}] = (Q, \Sigma, \mathcal{X}, \delta, q_0, \nu_0, \mu_{(t,x)})$. It is the same automaton as $\mathcal{A}$ except for the output function, which is defined as

$$\mu_{(t,x)}(q) = \begin{cases} \sigma(x) & \text{if } q = p \\ id_{\otimes} & \text{otherwise.} \end{cases}$$

For every transition $t = (p, a, p', \sigma)$ if $\mathrm{Root}(\sigma(x)) = \otimes$ then for every $e \in \mathrm{SubExp}(\sigma(x))$ we define the automaton $\mathcal{A}^{\otimes}[\mu_{(t,x,e)}] = (Q, \Sigma, \mathcal{X}, \delta, q_0, \nu_0, \mu_{(t,x,e)})$, where the output function is defined as

$$\mu_{(t,x,e)}(q) = \begin{cases} e & \text{if } q = p \\ id_{\otimes} & \text{otherwise.} \end{cases}$$

Similarly for every output function $\mu(q)$ if $\mathrm{Root}(\mu(q)) = \otimes$ then for every $e \in \mathrm{SubExp}(\mu(q))$ we define the automaton $\mathcal{A}^{\otimes}[\mu_{(q,e)}] = (Q, \Sigma, \mathcal{X}, \delta, q_0, \nu_0, \mu_{(q,e)})$, where

$$\mu_{(q,e)}(q) = \begin{cases} e & \text{if } q = p \\ id_{\otimes} & \text{otherwise.} \end{cases}$$

Each automaton with the modified output function recognizes a type of a subtree with $\overline{\otimes}$ in the root; or it outputs $\mathbb{0}$. Since $\mathcal{A}$ has alternation bounded by N then it is easy to see that $\mathcal{A}^{\otimes}[\mu_{(t,x,e)}]$ and $\mathcal{A}^{\otimes}[\mu_{(q,e)}]$ have alternation bounded by $N-1$. The automata $\mathcal{A}^{\otimes}[\mu_{(t,x)}]$ might have alternation N but we shall see that we need only the ones that have alternation bounded by $N-1$.

▸ Claim 28. Consider the run of $\mathcal{A}$ on w in (5), fix $1 \le i \le n$ and the transition $t_i = (p_{i-1}, w_i, p_i, \sigma_i) \in \delta$. The following equalities hold:

1. if $\mathrm{Root}(\sigma_i(x)) = \mathit{single}$ then $[\![\varsigma_{i-1} \circ \mathrm{P}_{\otimes}\{\sigma_i(x)\}]\!] = id_{\otimes}$;
2. if $\mathrm{Root}(\sigma_i(x)) = \overline{\otimes}$ then $[\![\varsigma_{i-1} \circ \mathrm{P}_{\otimes}\{\sigma_i(x)\}]\!] = [\![\mathcal{A}^{\otimes}[\mu_{(t_i,x)}]]\!](w[\cdot, i-1])$;
3. if $\mathrm{Root}(\sigma_i(x)) = \otimes$ then $[\![\varsigma_{i-1} \circ \mathrm{P}_{\otimes}\{\sigma_i(x)\}]\!] = \bigotimes_{e \in \mathrm{SubExp}(\sigma_i(x))} [\![\mathcal{A}^{\otimes}[\mu_{(t_i,x,e)}]]\!](w[\cdot, i-1])$;
4. additionally if $\mathrm{Root}(\mu(p_n)) = \otimes$ then $[\![\varsigma_n \circ \mathrm{P}_{\otimes}\{\mu(p_n)\}]\!] = \bigotimes_{e \in \mathrm{SubExp}(\mu(p_n))} [\![\mathcal{A}^{\otimes}[\mu_{(p_n,e)}]]\!](w)$.

Proof. By definition if $\mathrm{Root}((\sigma_i(x))) = \mathit{single}$ then $\mathrm{P}_{\otimes}\{\sigma_i(x)\} = id_{\otimes}$. This proves 1 from the claim.

The automata $\mathcal{A}^{\otimes}[\mu_{(t_i,x)}]$, $\mathcal{A}^{\otimes}[\mu_{(t_i,x,e)}]$, $\mathcal{A}^{\otimes}[\mu_{(p_n,e)}]$ differ from $\mathcal{A}$ only in the definition of the output function. Thus for every run the values in registers are the same in all these automata. In particular after reading the word $w[\cdot, i-1]$ the values of the registers are defined by the substitution ς_{i-1}. Thus we can focus only on the output functions of these automata. Similarly, the states are the same in all runs for these automata.

Suppose that $\mathrm{Root}(\sigma_i(x)) = \overline{\otimes}$. The automaton $\mathcal{A}^{\otimes}[\mu_{(t_i,x)}]$ is in state p_{i-1} after reading $w[\cdot, i{-}1]$ and its output function is defined as $\sigma_i(x)$. Since $\mathrm{Root}(\sigma_i(x)) = \overline{\otimes}$ then $\mathrm{P}_{\otimes}\{\sigma_i(x)\} = \sigma_i(x)$ by definition. This proves 2 from the claim.

Now, suppose that $\mathrm{Root}\,\sigma_i(x) = \otimes$. Again all automata $\mathcal{A}^{\otimes}[\mu_{(t,x,e)}]$ are in state p_{i-1} after reading $w[\cdot, i-1]$. The output function of $\mathcal{A}^{\otimes}[\mu_{(t_i,x,e)}]$ is e. Since $\mathrm{P}_{\otimes}\{\sigma_i(x)\} = \otimes\,\mathrm{SubExp}(\sigma_i(x))$ this proves 3 from the claim. Here we use the fact that $\mathrm{SubExp}(\sigma_i(x))$ is a multiset, because there might be repetitions among the expressions.

Finally suppose that $\mathrm{Root}(\mu(p_n)) = \otimes$. All automata $\mathcal{A}^{\otimes}[\mu_{(p_n,e)}]$ are in state p_n after reading w and their output function is defined as e. Since $\mathrm{P}_{\otimes}\{\mu(p_n)\} = \otimes\,\mathrm{SubExp}(\mu(p_n))$ this proves the last part of the claim. Similarly, as in the previous case here we use the fact that $\mathrm{SubExp}(\mu(p_n))$ is a multiset. ◂

This gives us automata that work on prefixes of w to recognize parts of the expression $\mathrm{P}_{\otimes}\{g\}$. Combining Claim 28 with the representation of $\mathrm{P}_{\otimes}\{g\}$ in (9) we get the following.

$$\mathrm{P}_{\otimes}\{g\} = \varsigma_n \circ \mathrm{P}_{\otimes}\{\mu(p_n)\} \otimes \bigotimes_{1 \le i \le n} \Big(\bigotimes_{x \in A_i} (\varsigma_{i-1} \circ \mathrm{P}_{\otimes}\{\sigma_i(x)\}) \Big) =$$
$$\bigotimes_{e \in \mathrm{SubExp}(\mu(p_n))} [\![\mathcal{A}^{\otimes}[\mu_{p_n,e}]]\!](w) \otimes \bigotimes_{1 \le i \le n} \bigotimes_{x \in A_i} \gamma_{i,x}, \qquad (11)$$

where

$$\gamma_{i,x} = \begin{cases} id_{\otimes} & \text{if } \mathrm{Root}(\sigma_i(x)) = \mathit{single} \\ [\![\mathcal{A}^{\otimes}[\mu_{(t_i,x)}]]\!](w[\cdot, i-1]) & \text{if } \mathrm{Root}(\sigma_i(x)) = \overline{\otimes} \\ \bigotimes_{e \in \mathrm{SubExp}(\sigma_i(x))} [\![\mathcal{A}^{\otimes}[\mu_{(t_i,x,e)}]]\!](w[\cdot, i-1]) & \text{if } \mathrm{Root}(\sigma_i(x)) = \otimes, \end{cases}$$

where $t_i = (p_{i-1}, w_i, p_i, \sigma_i) \in \delta$ are the transitions in the run (5).

To define the final formula we shall use the induction assumption that for all automata $\mathcal{A}^{\otimes}[\mu_{(t_i,x)}]$, $\mathcal{A}^{\otimes}[\mu_{(t_i,x,e)}]$, $\mathcal{A}^{\otimes}[\mu_{(p_n,e)}]$ there already exists a formula in MP defining the same function. As already mentioned some automata $\mathcal{A}^{\otimes}[\mu_{(t_i,x)}]$ might have alternation N; however, for such automata we do not need formulas because they cannot appear in (11).

Recall that $g = \mathrm{S}_{\otimes}\{g\} \otimes \mathrm{P}_{\otimes}\{g\}$. The automaton $\mathcal{A}^{\otimes}[\mu_{(t_i,x)}]$ outputs ground expressions with $\overline{\otimes}$ in the root (or $\mathbb{0}$). If such an expression would have alternation N and would appear in (11) then g would have alternation $N+1$, because the root of g is $\otimes$. Towards the end of this section we assume that the formulas $\psi_{\mathcal{A}^{\otimes}[\mu_{(t_i,x)}]}$, $\psi_{\mathcal{A}^{\otimes}[\mu_{(t_i,x,e)}]}$ are already defined by induction. If the alternation of $\mathcal{A}^{\otimes}[\mu_{(t_i,x)}]$ is N then we define $\psi_{\mathcal{A}^{\otimes}[\mu_{(t_i,x)}]} = id_{\otimes}$.

The remaining issue is to define the intervals for $\mathcal{A}^{\otimes}[\mu_{(t_i,x)}]$, $\mathcal{A}^{\otimes}[\mu_{(t_i,x,e)}]$ to define the formula equivalent to $\bigotimes_{1 \le i \le n} \bigotimes_{x \in A_i} \gamma_{i,x}$ in (11). There, we use prefixes of w which are not maximal intervals (exactly one of them is maximal). For this we need another modification on BAC automata.

Previously we modified the output function of $\mathcal{A}$, which resulted in automata defined on prefixes of w. The second step is to modify the initial states and the initialization function to get automata that work on infixes of w. We present the results for an arbitrary BAC automaton $\mathcal{B} = (Q^{\mathcal{B}}, \Sigma, \mathcal{X}, \delta^{\mathcal{B}}, q_0^{\mathcal{B}}, \nu_0^{\mathcal{B}}, \mu^{\mathcal{B}})$, but one should think of $\mathcal{B}$ as one of the automata $\mathcal{A}^{\otimes}[\mu_{(t_i,x)}]$, $\mathcal{A}^{\otimes}[\mu_{(t_i,x,e)}]$. For every transition $t = (q,a,p,\sigma) \in \delta^{\mathcal{B}}$ consider the automaton $\mathcal{B}[q_0^t, \nu_0^t] = (Q^{\mathcal{B}}, \Sigma, \mathcal{X}, \delta^{\mathcal{B}}, q_0^t, \nu_0^t, \mu^{\mathcal{B}})$. It is the same automaton as $\mathcal{B}$ except for the initial state defined as $q_0^t = p$ and the initialization function ν_0^t defined below.

$$\nu_0^t(x) = \begin{cases} [\![\sigma(x)]\!] & \text{if } \mathrm{Var}(\sigma(x)) = \varnothing \\ id_{\otimes} & \text{otherwise.} \end{cases}$$

It is easy to see that $\mathcal{B}[q_0^t, \nu_0^t]$ is also a BAC automaton and its alternation is bounded by the alternation of $\mathcal{B}$. Consider the runs of $\mathcal{B}$ and its track automaton $\mathcal{T}_{\mathcal{B}}^{\otimes}$ on a word $w = w_1 \dots w_n$.

$$(p_0, \nu_0), \dots, (p_n, \nu_n), \tag{12}$$

$$(p_0, A_0, B_0), \dots, (p_n, A_n, B_n). \tag{13}$$

▸ Claim 29. Let $i \ge 0$ be an index such that $B_i = \varnothing$, $B_{i+1} \ne \varnothing$. Then $[\![\mathcal{B}]\!](w) = [\![\mathcal{B}[q_0^t, \nu_0^t]]\!](w[i+1, \cdot])$, where $t = (p_i, w_{i+1}, p_{i+1}, \sigma_{i+1})$. If such an index i exists then it is unique; moreover $B_j = \varnothing$ for every $j < i$.

Proof. The automata $\mathcal{B}$ and $\mathcal{B}[q_0^t, \nu_0^t]$ differ only in the initial state and the initial function. The automaton $\mathcal{B}$ after reading $w[\cdot, i]$ is in state p_{i+1} which is also the initial state of $\mathcal{B}[q_0^t, \nu_0^t]$. Thus the run of $\mathcal{B}[q_0^t, \nu_0^t]$ on $w[i+1, \cdot]$ is

$$(p_i, \nu_0^t), (p_{i+1}, \nu_{i+1}^t) \dots, (p_n, \nu_n^t),$$

where p_i are the same states as in the run (12). Since the substitutions are the same as in $\mathcal{B}$ then the ground expression of the run of $\mathcal{B}[q_0^t, \nu_0^t]$ on $w[i+1, \cdot]$ is $\nu_0^t \circ g_{i+2}$, where g_i is the ground sequence of the run of $\mathcal{B}$ on w and ν_0^t is the initialization function of $\mathcal{B}[q_0^t, \nu_0^t]$.

On the other side by (7) $B_i = \mathrm{Var}(g_{i+1})$ and $B_{i+1} = \mathrm{Var}(g_{i+2})$. Since $B_i = \varnothing$ then g_{i+1} is a ground expression and $g_0 = g_{i+1}$. By definition we have $B_i = \bigcup_{x \in B_{i+1}} \mathrm{Var}(\sigma_{i+1}(x))$. This proves that for every $x \in B_{i+1}$ we have $\mathrm{Var}(\sigma_{i+1}(x)) = \varnothing$ and so $g_0 = g_{i+1} = \nu_0^t \circ g_{i+2}$, which proves the first part of the claim.

To prove that i is unique recall that $B_j = \bigcup_{x \in B_{j+1}} \mathrm{Var}(\sigma_{j+1}(x))$. Obviously if $B_{j+1} = \varnothing$ then $B_j = \varnothing$. Thus there can be at most one index i such that $B_{i+1} \ne \varnothing$ and $B_i = \varnothing$. ◂

Let us combine the modifications of changing the initial function and state with changing the output function. By $\mathcal{A}^{\otimes}[q_0^t, \nu_0^t, \mu_{(t',x)}]$, $\mathcal{A}^{\otimes}[q_0^t, \nu_0^t, \mu_{(t',x,e)}]$ we denote the cost register automata, defined as $\mathcal{A}$, with the initial state and initialization function changed as in

$\mathcal{A}^{\otimes}[q_0^t, \nu_0^t]$ and the output functions changed as for $\mathcal{A}^{\otimes}[\mu_{(t',x)}]$, $\mathcal{A}^{\otimes}[\mu_{(t',x,e)}]$. We denote the track automata of the modified automata in the natural way, e.g. the track automaton of $\mathcal{A}^{\otimes}[q_0^t, \nu_0^t, \mu_{(t',x)}]$ is $\mathcal{T}^{\otimes}[q_0^t, \nu_0^t, \mu_{(t',x)}]$.

Claim 29 intuitively shows that the important part of the run is where the B-component of the track automaton is nonempty. This motivates the following definition.

▸ **Definition 30.** Fix a set of registers $B' \subseteq \mathcal{X}$. For a given track automaton $\mathcal{T}^{\otimes}$ we define the following restricted variant $\mathcal{T}^{\otimes,B'}$: the initial states are (q_0, A, B') for all $A \subseteq \mathcal{X}$, where q_0 is the initial state of $\mathcal{A}$.

The only difference between $\mathcal{T}^{\otimes,B'}$ and $\mathcal{T}^{\otimes}$ is that the B-component is fixed. Recall that $\mathcal{T}^{\otimes}$ is unambiguous and recognizes Σ^*. The languages recognized by the automata $\mathcal{T}^{\otimes,B'}$ form a partition of Σ^* (up to the empty language). Observe that if $B' \neq \varnothing$ then all states in a run of $\mathcal{T}^{\otimes,B'}$ have the B-component nonempty. This follows from the last part of Claim 29. We denote by $R^{\otimes,B'}$ the regular expression recognizing $L(\mathcal{T}^{\otimes,B'})$. For track automata of the modified automata we use the natural notation, e.g. the corresponding regular language for the track automaton $\mathcal{T}^{\otimes,B'}[q_0^t, \nu_0^t, \mu_{(t',x)}]$ is denoted $R^{\otimes,B'}[q_0^t, \nu_0^t, \mu_{(t',x)}]$.

Before defining the formula $\mathrm{P}^{\otimes}_{s_0,s_q}$ we change the semantics of the selectors. As discussed in Section 3 defining intervals with selectors is equivalent to defining intervals with an MSO formula with two free variables. One can think of selectors as MSO formulas with two free variables. To define $\mathrm{P}^{\otimes}_{s_0,s_q}$ it is more convenient to select intervals from a word that could be selected by a rigid formula, without the maximal semantics. Thus slightly abusing notation we shall use selectors that select all intervals; and prove that the selectors correspond to rigid formulas. By Proposition 8 every rigid formula can be expressed by a sum of formulas with the maximal semantics; thus our selectors can be turned into sums of selectors with maximal semantics.

We define $\mathrm{P}^{\otimes}_{s_0,s_q}$.

$$\mathrm{P}^{\otimes}_{s_0,s_q} = \bigotimes_{e \in \mathrm{SubExp}(\mu(s_q))} \psi_{\mathcal{A}^{\otimes}[\mu_{(s_q,e)}]} \otimes \varphi_{s_0,s_q}.$$

As explained for (10), the formula $\mathrm{P}^{\otimes}_{s_0,s_q}$ is applied only when s_0 and s_q are the initial and final state of the run of $\mathcal{T}^{\otimes}$ on the given word w. Let us recall the characterization of $\mathrm{P}_{\otimes}\{g\}$ in (11).

$$\mathrm{P}_{\otimes}\{g\} = \bigotimes_{e \in \mathrm{SubExp}(\mu(p_n))} [\![\mathcal{A}^{\otimes}[\mu_{(p_n,e)}]]\!](w) \otimes \bigotimes_{1 \le i \le n} \bigotimes_{x \in A_i} \gamma_{i,x},$$

where p_n is the final state of the run of $\mathcal{T}^{\otimes}$. By (10) in this setting the final state is s_q. The formula $\bigotimes_{e \in \mathrm{SubExp}(\mu(s_q))} \psi_{\mathcal{A}^{\otimes}[\mu_{(s_q,e)}]}$ is defined by induction because automata $\mathcal{A}^{\otimes}[\mu_{(s_q,e)}]$ have alternation at most $N-1$. The remaining formula φ_{s_0,s_q} will be equivalent to $\bigotimes_{1 \le i \le n} \bigotimes_{x \in A_i} \gamma_{i,x}$. Before we define φ_{s_0,s_q} let us recall the definition of $\gamma_{i,x}$ from (11)

$$\gamma_{i,x} = \begin{cases} id_{\otimes} & \text{if } \mathrm{Root}(\sigma_i(x)) = \mathit{single} \\ [\![\mathcal{A}^{\otimes}[\mu_{(t_i,x)}]]\!](w[\cdot, i-1]) & \text{if } \mathrm{Root}(\sigma_i(x)) = \overline{\otimes} \\ \bigotimes_{e \in \mathrm{SubExp}(\sigma_i(x))} [\![\mathcal{A}^{\otimes}[\mu_{(t_i,x,e)}]]\!](w[\cdot, i-1]) & \text{if } \mathrm{Root}(\sigma_i(x)) = \otimes. \end{cases} \tag{14}$$

Notice that automata run on strict prefixes of the given word w. They run up to $i-1$ and $i \le n$, where n is the length of the word.

Recall the shorthands for the states of track automata $s_q = (q, A_q, B_q)$, $s_i = (q_i, A_i, B_i)$. We define selectors that we will use later.

$$(R^{\otimes}_{s_0\to s_1} \cdot a)\langle R^{\otimes}_{s_2\to s_3}\rangle(b \cdot R^{\otimes}_{s_4\to s_q}), \tag{15}$$

$$(\epsilon)\langle R^{\otimes}_{s_0\to s_3}\rangle(b \cdot R^{\otimes}_{s_4\to s_q}), \tag{16}$$

for some fixed $s_i \in Q^{\otimes}$, states of the track automaton $\mathcal{T}^{\otimes}$; and $a, b \in \Sigma$. We additionally require that (s_1, a, s_2) and (s_3, b, s_4) are transitions in $\mathcal{T}^{\otimes}$ and thus $t_a = (q_1, a, q_2, \sigma), t_b = (q_3, b, q_4, \sigma')$ are transitions in $\mathcal{A}$ for some substitutions σ, σ'. Observe that every interval in a given word w, that is not a suffix, is selected by one of the selectors in (15, 16). This is because these selectors reflect the run of $\mathcal{T}^{\otimes}$ divided into parts. We explain how (15) selects infixes:

- the expression $R^{\otimes}_{s_0\to s_1}$ corresponds to the starting part of the run from the initial state s_0 to the state s_1;
- then there is the transition (s_1, a, s_2);
- the expression $R^{\otimes}_{s_2\to s_3}$ corresponds to the selected part of the run from the state s_2 to the state s_3;
- then there is the transition (s_3, b, s_4);
- the expression $R^{\otimes}_{s_4\to s_q}$ corresponds to the final part of the run from the state s_4 to the final state s_q.

Similarly (16) selects prefixes. Since $\mathcal{T}^{\otimes}$ is unambiguous and recognizes Σ^*, then every interval (that is not a suffix) is selected uniquely (recall that we do not use maximal semantics).

We start with the following formula

$$\varphi'_{s_0,s_q} = \bigotimes_{b\in\Sigma} \bigotimes_{s_1,s_2,s_3,s_4\in Q^{\otimes}} \bigotimes_{x\in A_4} \eta, \tag{17}$$

where

$$\eta = \begin{cases} id_{\otimes} & \text{if } \mathrm{Root}(\sigma'(x)) = single \\ \otimes(\epsilon)\langle R_{s_0\to s_3}\rangle(b \cdot R_{s_4\to s_q}).\ \psi_{\mathcal{A}[\mu_{(t_b,x)}]} & \text{if } \mathrm{Root}(\sigma'(x)) = \overline{\otimes} \\ \otimes_{e\in\mathrm{SubExp}(\sigma'(x))} & \\ \quad \otimes(\epsilon)\langle R_{s_0\to s_3}\rangle(b \cdot R_{s_4\to s_q}).\ \psi_{\mathcal{A}[\mu_{(t_b,x,e)}]} & \text{if } \mathrm{Root}(\sigma'(x)) = \otimes \end{cases}$$

where (s_3, b, s_4) is a transition in $\mathcal{T}^{\otimes}$ and $t_b = (q_3, b, q_4, \sigma')$ is its corresponding transition in $\mathcal{A}$ for some substitution σ'. The formula η depends on s_1, s_2, s_3, s_4, b and x but for the sake of readability we skip the indexes. The formula η corresponds to the formula $\gamma_{i,x}$ (14). As explained before the selectors $(\epsilon)\langle R^{\otimes}_{s_0\to s_3}\rangle(b \cdot R^{\otimes}_{s_4\to s_q})$ select all prefixes uniquely. The position of the letter b corresponds to the position i in $\gamma_{i,x}$.

Let us explain that φ'_{s_0,s_q} is equivalent to $\otimes_{1\le i\le n} \otimes_{x\in A_i} \gamma_{i,x}$. The aggregation of all prefixes $\otimes_{1\le i\le n}$ is split into pieces: first with $\otimes_{b\in\Sigma} \otimes_{s_1,s_2,s_3,s_4\in Q^{\otimes}}$ we parametrize prefixes depending on the partial run of $\mathcal{T}^{\otimes}$; second $(\epsilon)\langle R^{\otimes}_{s_0\to s_3}\rangle(b \cdot R^{\otimes}_{s_4\to s_q})$ selects all prefixes of the given type. The formulas $\psi_{\mathcal{A}[\mu_{(t_b,x)}]}$, $\psi_{\mathcal{A}[\mu_{(t_b,x,e)}]}$ exist by the induction assumption. Unfortunately, the formula φ'_{s_0,s_q} is not rigid (all selected intervals are prefixes), and we need to improve φ'_{s_0,s_q}.

We divide formula φ_{s_0,s_q} into two subformulas defining $\varphi_{s_0,s_q} = \varphi_{prefixes} \otimes \varphi_{infixes}$. The formulas $\varphi_{prefixes}, \varphi_{infixes}$ depend on s_0 and s_q, but we skip these indexes for readability. The message of Claim 29 is that it suffices to run the automaton on the interval, where the B-component is nonempty. The formula $\varphi_{infixes}$ is the subformula for automata that have the B-component empty at some point, and the formula $\varphi_{prefixes}$ is for automata that have always the B-component nonempty.

For two regular expressions R, R' we denote by $R \cap R'$ the regular expression recognizing $L(R) \cap L(R')$. Let us define $\varphi_{prefixes}$.

$$\varphi_{prefixes} = \bigotimes_{B\subseteq\mathcal{X}, B\neq\varnothing} \bigotimes_{b\in\Sigma} \bigotimes_{s_1,s_2,s_3,s_4\in Q^{\otimes}} \bigotimes_{x\in A_4} \eta,$$

where

$$\eta = \begin{cases} id_{\otimes} & \text{if } \mathrm{Root}(\sigma'(x)) = single \\ \bigotimes(\epsilon)\langle R_{s_0\to s_3} \cap R^{\otimes,B}[\mu_{(t_b,x)}]\rangle(b\cdot R_{s_4\to s_q}).\ \psi_{\mathcal{A}[\mu_{(t_b,x)}]} & \text{if } \mathrm{Root}(\sigma'(x)) = \overline{\otimes} \\ \bigotimes_{e\in\mathrm{SubExp}(\sigma'(x))} & \\ \quad \bigotimes(\epsilon)\langle R_{s_0\to s_3} \cap R^{\otimes,B}[\mu_{(t_b,x,e)}]\rangle(b\cdot R_{s_4\to s_q}).\ \psi_{\mathcal{A}[\mu_{(t_b,x,e)}]} & \text{if } \mathrm{Root}(\sigma'(x)) = \otimes \end{cases}$$

where (s_3, b, s_4) is a transition in $\mathcal{T}^{\otimes}$ and $t_b = (q_3, b, q_4, \sigma')$ is a transition in $\mathcal{A}$ for some substitution σ'. This formula is a modified variant of the formula φ'_{s_0,s_q} from (17). We consider the cases of runs where Claim 29 could not be applied, i.e., all B-components are nonempty. To ensure that this is the case we use regular expressions $R^{\otimes,B}[\mu_{(t_b,x)}]$, $R^{\otimes,B}[\mu_{(t_b,x,e)}]$ corresponding to automata from Definition 30. They check that the B-component is never empty on the selected intervals.

The formula $\varphi_{infixes}$ is also a modified variant of the formula φ'_{s_0,s_q} from (17). The first modification is that instead of selecting the prefixes, we select the infixes with the selector from (15)

$$(R^{\otimes}_{s_0\to s_1}\cdot a)\langle R^{\otimes}_{s_2\to s_3}\rangle(b\cdot R^{\otimes}_{s_4\to s_q}).$$

The goal is to select the intervals, where the B-component is nonempty. Let t_a be the transition preceding the selected interval, and t_b the transition after the selected interval. In φ'_{s_0,s_q} we used automata $\mathcal{A}[\mu_{(t_b,x)}], \mathcal{A}[\mu_{(t_b,x,e)}]$ on prefixes. On infixes we shall use $\mathcal{A}^{\otimes}[q_0^{t_a}, \nu_0^{t_a}, \mu_{(t_b,x)}], \mathcal{A}^{\otimes}[q_0^{t_a}, \nu_0^{t_a}, \mu_{(t_b,x,e)}]$. To select intervals with the B-component nonempty we need to find the biggest position, where the B-component becomes empty.

By $(R)\langle S\rangle(T) \cap (R')\langle S'\rangle(T')$ we denote the selector $(R\cap R')\langle S\cap S'\rangle(T\cap T')$.

$$\varphi_{infixes} = \bigotimes_{B\subseteq\mathcal{X}} \bigotimes_{a,b\in\Sigma} \bigotimes_{s_1,s_2,s_3,s_4\in Q^{\otimes}} \bigotimes_{x\in A_4} \eta,$$

where

$$\eta = \begin{cases} id_{\otimes} & \text{if } \mathrm{Root}(\sigma'(x)) = single \\ \bigotimes(R^{\otimes}_{s_0\to s_1}\cdot a)\langle R^{\otimes}_{s_2\to s_3}\rangle(b\cdot R_{s_4\to s_q}) & \\ \cap \quad (\Sigma^*)\langle R^{\otimes,B}[q_0^{t_a}, \nu_0^{t_a}, \mu_{(t_b,x)}] \cap R_B\rangle(\Sigma^*).\ \psi_{\mathcal{A}^{\otimes}[q_0^{t_a},\nu_0^{t_a},\mu_{(t_b,x)}]} & \text{if } \mathrm{Root}(\sigma'(x)) = \overline{\otimes} \\ \bigotimes_{e\in\mathrm{SubExp}(\sigma'(x))} \bigotimes(R^{\otimes}_{s_0\to s_1}\cdot a)\langle R^{\otimes}_{s_2\to s_3}\rangle(b\cdot R^{\otimes}_{s_4\to s_q}) & \\ \cap \quad (\Sigma^*)\langle R^{\otimes,B}[q_0^{t_a}, \nu_0^{t_a}, \mu_{(t_b,x,e)}] \cap R_B\rangle(\Sigma^*).\ \psi_{\mathcal{A}^{\otimes}[q_0^{t_a},\nu_0^{t_a},\mu_{(t_b,x,e)}]} & \text{if } \mathrm{Root}(\sigma'(x)) = \otimes \end{cases}$$

where (s_1, a, s_2), (s_3, b, s_4) are transitions in $\mathcal{T}^{\otimes}$; $t_a = (q_1, a, q_2, \sigma), t_b = (q_3, b, q_4, \sigma')$ are transitions in $\mathcal{A}$ for some substitutions σ, σ'; and

$$R_B = \begin{cases} \Sigma^* & \text{if } B \neq \varnothing \\ \{\epsilon\} & \text{otherwise.} \end{cases}$$

Additionally we require $\mathrm{Var}(\sigma(x)) = \varnothing$ for every $x \in B$ (if this is not true, we simply define $\eta = id_{\otimes}$). Let us explain $\varphi_{infixes}$ step by step.

First let us assume that $B \neq \varnothing$. Then we can ignore R_B because it is equal to Σ^*. With the expressions $R^{\otimes,B}[q_0^{t_a}, \nu_0^{t_a}, \mu_{(t_b,x,e)}]$ we assure that the B-component is nonempty on the selected interval. With the additional assumption $\mathrm{Var}(\sigma(x)) = \varnothing$ for every $x \in B$ we assure that the position of the letter a (in (15)) is the first position where the B-component is empty. Thus by Claim 29 the output of $\mathcal{A}[\mu_{(t_b,x)}], \mathcal{A}[\mu_{(t_b,x,e)}]$ on prefixes is equal to the output of $\mathcal{A}^{\otimes}[q_0^{t_a}, \nu_0^{t_a}, \mu_{(t_b,x)}], \mathcal{A}^{\otimes}[q_0^{t_a}, \nu_0^{t_a}, \mu_{(t_b,x,e)}]$ on the corresponding suffixes.

Now let $B = \varnothing$. These are runs that have always the B-component empty. This happens when the output function is an expression without registers. Then $R_B = \{\epsilon\}$ is the language containing only the empty word. Since we do not need to read anything to know the output, we select the empty word ϵ as the interval.

We need to show two things. First, that the formulas φ'_{s_0,s_q} and $\varphi_{s_0,s_q} = \varphi_{prefixes} \otimes \varphi_{infixes}$ are equivalent. Second, that the selectors in the final formulas are rigid formulas (recall that we do not work with maximal semantics).

▶ Claim 31. The formulas φ'_{s_0,s_q} and φ_{s_0,s_q} are equivalent.

Proof. The formulas φ'_{s_0,s_q}, φ_{s_0,s_q} are aggregations of subformulas of the shape $\bigotimes \mathcal{R}.\ \phi$ and $id_{\otimes}$. The subformulas $id_{\otimes}$ do not affect the output since in both formulas everything is aggregated with $\otimes$.

The remaining subformulas in φ'_{s_0,s_q} output the run of automata $\mathcal{A}[\mu_{(t_b,x)}]$, $\mathcal{A}[\mu_{(t_b,x,e)}]$ on selected prefixes. The formula φ_{s_0,s_q} is split into two subformulas $\varphi_{prefixes}$ and $\varphi_{suffixes}$. The formula $\varphi_{prefixes}$ is a restricted variant of φ'_{s_0,s_q}. It aggregates all outputs that have the B-component nonempty in the track runs.

The formula $\varphi_{infixes}$ is more involved, it aggregates the remaining outputs. For the remaining runs we are interested in the biggest position in the word, where the B-component is empty. We shall refer to this position by i. In the selector

$$(R^{\otimes}_{s_0 \to s_1} \cdot a)\langle R^{\otimes}_{s_2 \to s_3}\rangle(b \cdot R^{\otimes}_{s_4 \to s_q}).$$

the position i is labeled with a. By $t_a = (q_1, a, q_2, \sigma)$ we denote the transition from i to the next position. The parameter B is used to guess the set of registers in the next position.

If $B = \varnothing$ then the output is defined only by the output function, and the output function does not use registers. There we use the expression R_B, which recognizes the language $\{\epsilon\}$ in this case. By restricting to the empty interval we assure that we aggregate this output exactly once.

If $B \neq \varnothing$ then the regular expression $R^{\otimes,B}[q_0^{t_a}, \nu_0^{t_a}, \mu_{(t_b,x)}]$ assures that we select intervals, where the B-component in the first position is our guessed set B. The additional condition $\mathrm{Var}(\sigma(x)) = \varnothing$ for every $x \in B$ ensures that the B-component is empty in i, and moreover, it is the first position with the B-component empty. We modify the initial states and initial function of the automata $\mathcal{A}[\mu_{(t_b,x)}]$, $\mathcal{A}[\mu_{(t_b,x,e)}]$ to fulfill the assumptions of Claim 29. By Claim 29 the runs of the modified automata on infixes output the same value as the automata $\mathcal{A}[\mu_{(t_b,x)}]$, $\mathcal{A}[\mu_{(t_b,x,e)}]$. For every run if there exists a position i, where the B-component is empty for the first time, then it is also unique. Thus every output defined by a run of $\mathcal{A}[\mu_{(t_b,x)}]$, $\mathcal{A}[\mu_{(t_b,x,e)}]$ is aggregated exactly once by the formula $\varphi_{infixes}$. ◀

▶ Claim 32. The selectors in $\varphi_{prefixes}$ and $\varphi_{infixes}$ are MSO rigid formulas.

Proof. Let $\zeta(x,y)$ be an MSO formula. Recall that the set of intervals selected with $\zeta(x,y)$ on a given word w is the set $\{(i,j) \mid w(i,j) \vDash \zeta(x,y)\}$. By convention the empty interval between positions i and $i+1$ is represented by $(i+1,i)$. A formula $\zeta(x,y)$ is rigid if:

1. for every i there is at most one j such that $w(i,j) \vDash \zeta(x,y)$;
2. for every j there is at most one i such that $w(i,j) \vDash \zeta(x,y)$.

Let us start with the selectors in $\varphi_{prefixes}$. There two types of selectors

$$(\epsilon)\langle R_{s_0\to s_3} \cap R^{\otimes,B}[\mu_{(t_b,x)}]\rangle(b\cdot R_{s_4\to s_q}),$$
$$(\epsilon)\langle R_{s_0\to s_3} \cap R^{\otimes,B}[\mu_{(t_b,x,e)}]\rangle(b\cdot R_{s_4\to s_q}).$$

For the sake of readability we shall focus only on the first type of selectors. The analysis for the second type of selectors is similar. Let $\zeta(x,y)$ be an MSO formula equivalent to one of the selectors and let $w = w_1 \dots w_n$ be a given word. We prove that $\zeta(x,y)$ is a rigid formula. The condition 2 is trivial because the left end of the selected intervals is always at the beginning of the word. Let us prove condition 1, which in fact proves that each selector selects only one interval.

Recall that by definition the set B is nonempty; the states s_0, s_q are the initial and final states of the run of the track automaton of $\mathcal{A}$; and $t_b = (q_3, b, q_4, \sigma')$. Also by definition of $R^{\otimes,B}[\mu_{(t_b,x)}]$ the set B is the B-component in the first state of $\mathcal{T}^{\otimes,B}[\mu_{(t_b,x)}]$.

Let $g_{n+1},\dots,g_0$ be the ground sequence of the run of $\mathcal{A}$ on w and let $g'_{m+1},\dots,g'_0$ be the ground sequence of the run of $\mathcal{A}^{\otimes,B}[\mu_{(t_b,x)}]$ on a selected interval $w[\cdot,m]$. Recall that by (6),(7) we have equalities on A- and B-components of $\mathcal{T}^{\otimes}$: $A_i = \mathrm{Var}_{\otimes}(g_{i+1})$, $B_i = \mathrm{Var}(g_{i+1})$. Then we have:

- By definition of $\varphi_{prefixes}$ we have $x \in A_{m+1}$. Since $A_{m+1} = \mathrm{Var}_{\otimes}(g_{m+2})$, then $\sigma'(x)$ is a subexpression of g_{m+1}.
- By definition of $\mathcal{A}^{\otimes,B}[\mu_{(t_b,x)}]$ we have $g'_{m+1} = \sigma'(x)$.

Altogether this implies that g'_{m+1} is a subexpression of g_{m+1}. By induction down from $m+1$ to 1 it is easy to show that g'_1 is a subexpression of g_1. Recall that the set B is the B-component in the first state of $\mathcal{T}^{\otimes,B}[\mu_{(t_b,x)}]$. This means that $B = \mathrm{Var}(g'_1)$. Now, suppose that $\zeta(x,y)$ is not a rigid formula and that there are two different intervals selected by $\zeta(x,y)$. This means that there are two different ground sequences g' and g'' such that g'_1 and g''_1 are subexpressions of g_1. But $B = \mathrm{Var}(g'_1) = \mathrm{Var}(g''_1)$ and since B is nonempty then the expression g_1 is not copyless. This is a contradiction with the fact that $\mathcal{A}$ is copyless.

Now let us move to the selectors of $\varphi_{infixes}$

$$(R^{\otimes}_{s_0\to s_1}\cdot a)\langle R^{\otimes}_{s_2\to s_3}\rangle(b\cdot R_{s_4\to s_q}) \cap (\Sigma^*)\langle R^{\otimes,B}[q_0^{t_a},\nu_0^{t_a},\mu_{(t_b,x)}] \cap R_B\rangle(\Sigma^*)$$
$$(R^{\otimes}_{s_0\to s_1}\cdot a)\langle R^{\otimes}_{s_2\to s_3}\rangle(b\cdot R_{s_4\to s_q}) \cap (\Sigma^*)\langle R^{\otimes,B}[q_0^{t_a},\nu_0^{t_a},\mu_{(t_b,x,e)}] \cap R_B\rangle(\Sigma^*).$$

Let $\zeta(x,y)$ be an MSO formula equivalent to one of the selectors. For $B = \varnothing$ the formula is trivially rigid, because $R_B = \{\epsilon\}$. Otherwise if the condition 1 would not hold, then like for $\varphi_{prefixes}$ it would violate the fact that $\mathcal{A}$ is copyless.

It remains to prove the condition 2. Recall that $t_a = (q_1, b, q_2, \sigma)$. We assumed that $B \neq \varnothing$ and by definition the selectors are defined only if $\mathrm{Var}(\sigma(x)) = \varnothing$ for every $x \in B$. By Claim 29 such a position is unique. ◂

Thus we have defined the formula $\mathrm{P}^{\otimes}_{s_0,s_q}$ that defines the same functions as $\mathrm{P}_{\otimes}\{g\}$; earlier we defined $\mathrm{S}^{\otimes}_{s_0,s_q}$ corresponding to $\mathrm{S}_{\otimes}\{g\}$; where g is the ground expression defined by the run of $\mathcal{A}$. By (3) the formula $\mathrm{P}^{\otimes}_{s_0,s_q} \otimes \mathrm{S}^{\otimes}_{s_0,s_q}$ outputs $[\![g]\!]$, which is the output of $\mathcal{A}$.

Recall that at some point we assumed that we consider only runs where the root of the ground expression is $\otimes$. This gives us two formulas $\psi^{\oplus}_{\mathcal{A}}$ and $\psi^{\odot}_{\mathcal{A}}$.; and leaves the case of runs, where the ground expression is a constant. For them we define the following formula.

$$\psi^{single}_{\mathcal{A}} = \bigotimes_{s_0,s_q} \bigotimes (\epsilon)\langle R_{s_0\to s_q}\rangle(\epsilon).\ \mathrm{S}^{\oplus}_{s_0,s_q}.$$

This is the same formula as $\psi^{\oplus}_{\mathcal{A}}$ without the component $\mathrm{P}^{\oplus}_{s_0,s_q}$. Like before for every word there is exactly one pair (s_0, s_q) for which the interval is selected. It is easy to verify that the formula $\mathrm{S}^{\oplus}_{s_0,s_q}$ outputs the unique constant that is the output of $\mathcal{A}$. Notice that the choice of the operator $\oplus$ over $\odot$ is arbitrary, since there is only one constant to aggregate. It remains to define the final formula

$$\psi_{\mathcal{A}} = \left(\bigoplus(\epsilon)\langle R^{\oplus}\rangle(\epsilon).\ \psi^{\oplus}_{\mathcal{A}}\right) \oplus \left(\bigoplus(\epsilon)\langle R^{\odot}\rangle(\epsilon).\ \psi^{\odot}_{\mathcal{A}}\right) \oplus \left(\bigoplus(\epsilon)\langle R^{single}\rangle(\epsilon).\ \psi^{single}_{\mathcal{A}}\right)$$

where $R^{\oplus}, R^{\odot}, R^{single}$ are regular expressions that partition Σ^* into three languages depending on the root of the ground expression defined by the run of $\mathcal{A}$. The definition of finite automata recognizing $L(R^{\oplus}), L(R^{\odot}), L(R^{single})$ is very similar to the definition of track automata $\mathcal{T}^{\otimes}$ and we leave it to the reader. It is straightforward that on a given word $\psi_{\mathcal{A}}$ is equivalent to $\psi^{r}_{\mathcal{A}}$, where r is the root of the ground expression defined by the run of $\mathcal{A}$.

To conclude notice that we used the induction assumption only to define $\mathrm{P}^{\otimes}_{s_0,s_q}$. In the base steps of induction for $N = 0, 1$ there is no alternation between the operators $\odot$ and $\oplus$. Then we do not have to define the subformula $\mathrm{P}^{\otimes}_{s_0,s_q}$ and we do not use the induction assumption.

D Proof of Proposition 10

The proof is by induction over the size of a formula. For the basic case if $\varphi = c$ then the formula φ is trivially invariant under the orientation of the word. If $\varphi = \varphi_1 \otimes \varphi_2$ for $\otimes \in \{\oplus, \odot\}$ then using the formulas φ^r_1, φ^r_2 defined by induction assumption we define $\varphi^r = \varphi^r_1 \otimes \varphi^r_2$. For the last case let $\phi = \bigotimes t.\ \psi$, where $t = R\langle S\rangle T$. We define $\varphi^r = \bigotimes t^r.\ \psi^r$, where ψ^r is defined by induction assumption; $t^r = T^r\langle S^r\rangle R^r$; and R^r, S^r, T^r are the regular languages recognizing the reverse of languages of R, S, T. It is straightforward that φ^r is the reverse formula of φ.

E Proof of Proposition 13

We prove this proposition by showing that for every copyless CRA there exists a weighted automaton that defines the same function. Since weighted automata were shown in [7] to be equally expressive than $\mathrm{WMSO}[\bigoplus_X \odot^1_x]$, by Theorem 9 this shows that MP is contained in $\mathrm{WMSO}[\bigoplus_X \odot^1_x]$.

The following result will be useful during this proof. It shows the form of a copyless expression when it is rewrite as a sum of monomials.

▶ **Lemma 33.** *For any copyless expression e, there exist an equivalent expression e' of the form:*

$$e' \equiv \bigoplus_{i=1}^{k} \left(c_i \bigodot_{x \in X_i} x \right)$$

where $X_1, \ldots, X_k$ is a sequence of different sets over $\mathcal{X}$ and $c_1, \ldots, c_k$ is a sequence of values over $\mathbb{S}$ for $k \geq 0$.

Proof. The lemma is shown by induction on the size of e. For the base case, when e is equal to a constant or a variable, the lemma trivially holds by taking $e' = e$. For the inductive case, suppose that $e = e_1 \otimes e_2$ where $\otimes$ is either $\oplus$ or $\odot$. By the inductive hypothesis we know that there exist expressions e'_1 and e'_2 equivalent to e_1 and e_2, respectively, such that for $j \in \{1, 2\}$:

$$e'_j \equiv \bigoplus_{i=1}^{k_j} \left(c^j_i \bigodot_{x \in X^j_i} x \right)$$

where $X_1^j, \dots, X_{k_j}^j$ is a sequence of different sets over $\mathcal{X}$ and $c_1^j, \dots, c_{k_j}^j$ is a sequence of values over $\mathbb{S}$ for $k_j \geq 0$. Given that e is a copyless expression, then we know that $\mathrm{Var}(e_1) \cap \mathrm{Var}(e_2)$. Without lost of generality, we can assume that $X_i^j \subseteq \mathrm{Var}(e_j)$ for $j \in \{1, 2\}$ and $i \leq k_j$. If not and there exists $X_i^j \nsubseteq \mathrm{Var}(e_j)$, then this implies that c_i^j must be equal to $\mathbb{0}$ (i.e. X_i^j is not contributing in e_j') and we can omit X_i^j in e_j'. Then since $X_i^j \subseteq \mathrm{Var}(e_j)$ and $\mathrm{Var}(e_1) \cap \mathrm{Var}(e_2)$, this implies that:

$$X_{i_1}^1 \cap X_{i_2}^2 = \varnothing \qquad \text{for every } i_1 \leq k_1 \text{ and } i_2 \leq k_2. \tag{18}$$

Now we consider when $\otimes$ is either $\oplus$ or $\odot$. If $e = e_1 \oplus e_2$, then by considering $e' = e_1 \oplus e_2$ the lemma is proved given that by (18) all sets of the form X_i^j are different for $j \in \{1, 2\}$ and $i \leq k_j$. Otherwise, $e = e_1 \otimes e_2$ and we get:

$$\begin{aligned} e_1' \odot e_2' &= \bigoplus_{i=1}^{k_1} \Big(c_i^1 \bigodot_{x \in X_i^1} x\Big) \odot \bigoplus_{i=1}^{k_2} \Big(c_i^2 \bigodot_{x \in X_i^2} x\Big) \\ &= \bigoplus_{i_1=1}^{k_1} \bigoplus_{i_2=1}^{k_2} \Big(c_{i_1}^1 \odot c_{i_2}^1 \bigodot_{x \in X_{i_1}^1 \cup X_{i_2}^2} x\Big) \quad = \quad e' \end{aligned}$$

The last derivation holds by (18), i.e., we do not need to consider repetitions in the multiplication of two monomials since $X_{i_1}^1 \cap X_{i_2}^2 = \varnothing$. One can also check that all sets of the form $X_{i_1}^1 \cup X_{i_2}^2$ are different. Indeed, all sets $X_{i_1}^1$ are pairwise different and the same holds for $X_{i_2}^2$. This means by (18) that all sets $X_{i_1}^1 \cup X_{i_2}^2$ must be different as well. Then e' is equivalent to e and it has the form stated in the lemma. ◂

Fix a semiring $\mathbb{S}$ and a finite alphabet Σ. A *weighted automaton* over Σ [20, 8] is a tuple $\mathcal{A} = (Q, \Sigma, E, I, F)$ where Q is a finite set of states, $E : Q \times \Sigma \times Q \to \mathbb{S}$ is a weighted transition relation, and $I, F : Q \to \mathbb{S}$ is the initial and final function, respectively. Usually, if $E(p, a, q) = s$, we denote this transition graphically by $p \xrightarrow{a/s} q$. A *run* ρ of $\mathcal{A}$ is a sequence of transitions:

$$\rho = q_0 \xrightarrow{w_1/s_1} q_1 \xrightarrow{w_2/s_2} \dots \xrightarrow{w_n/s_n} q_n.$$

where $w_i \neq \mathbb{0}$ for all $i \leq n$. We say that ρ is a run of $\mathcal{A}$ over a word $w = a_1 a_2 \dots a_n$ if it also holds that $I(q_0) \neq \mathbb{0}$. Moreover, a run ρ like above is *accepting* if $I(q_0) \neq \mathbb{0}$ and $F(q_n) \neq \mathbb{0}$. In this case, the *weight* of an accepting run ρ of $\mathcal{A}$ over w is defined by $|\rho| = I(q_0) \odot \prod_{i=1}^n s_i \odot F(q_n)$. We define $\mathrm{Run}_{\mathcal{A}}(w)$ as the set of all accepting runs of $\mathcal{A}$ over w. Finally, the weight of $\mathcal{A}$ over a word w is defined by

$$[\![\mathcal{A}]\!](w) = \sum_{\rho \in \mathrm{Run}_{\mathcal{A}}(w)} |\rho|$$

where the sum is equal to $\mathbb{0}$ if $\mathrm{Run}_{\mathcal{A}}(w)$ is empty. The set $\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \mathrm{Run}_{\mathcal{A}}(w) \neq \varnothing\}$ is called the *language* of $\mathcal{A}$.

A weighted automaton $\mathcal{A}$ is called *unambiguous* if $|\mathrm{Run}_{\mathcal{A}}(w)| \leq 1$ for every $w \in \Sigma^*$ and is called *finitely ambiguous* if there exists a uniform bound N such that $|\mathrm{Run}_{\mathcal{A}}(w)| \leq N$ for every $w \in \Sigma^*$ [22, 13]. Furthermore, $\mathcal{A}$ is called *polynomial ambiguous* if the function $|\mathrm{Run}_{\mathcal{A}}(\cdot)|$ is bounded by a polynomial on the input length [22, 13]. For the special case when the number of runs of $\mathcal{A}$ are bounded by a linear function, we say that $\mathcal{A}$ is *linear ambiguous*.

Proof of Proposition 13. We show that for every copyless cost register automaton $\mathcal{A}$ there exists a weighted automaton $\mathcal{W}$ that recognizes the same function.

By Lemma 33, any copyless expression e can be rewritten as an equivalent expression e' of the form:

$$e' \equiv \bigoplus_{i=1}^{k} \left(c_i \bigodot_{x \in X_i} x \right)$$

where $X_1, \ldots, X_k$ is a sequence of pairwise different sets over $\mathcal{X}$ and $c_1, \ldots, c_k$ is a sequence of values over $\mathbb{S}$ for $k \geq 0$. Therefore, for any copyless expression e we denote by $\text{Mon}(e)$ the set of all pairs $(X, c) \in 2^{\mathcal{X}} \times \mathbb{S}$ where $c \odot \bigodot_{x \in X} x$ is a monomial in the above expression e' of e.

Let $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \delta, q_0, \nu_0, \mu)$ be a copyless cost register automaton. The proof idea is to construct a weighted automaton $\mathcal{W}$ equivalent to $\mathcal{A}$ such that each run of $\mathcal{W}$ over a word w guesses a possible monomial that results from distributing products inside sums in output ground expression of $\mathcal{A}$ over w. By the semantics of weighted automata, the output of $\mathcal{W}$ will be the sum of all products and this will correspond to the representation as a sum of monomials of the output ground expression of $\mathcal{A}$ over w.

We define a weighted automaton $\mathcal{W} = (Q', \Sigma, E', I', F')$ that computes the same function as $\mathcal{A}$ such that:

- $Q' = Q \times 2^{\mathcal{X}}$,
- $I'(q, S) = \bigodot_{x \in S} \nu_0(x)$ whenever $q = q_0$ and $\mathbb{0}$ otherwise,
- $F'(q, S) = c$ whenever $(S, c) \in \text{Mon}(\mu(x))$ and $\mathbb{0}$ otherwise.
- $E'((q, S), a, (q', S')) = c$ iff $\delta(q, a) = (q', \sigma)$ and there exists a set $\{(S_x, c_x)\}_{x \in S'}$ (i.e. the set is indexed by elements in S') where $S_x \subseteq \mathcal{X}$ and $c_x \in \mathbb{S}$ such that:
  1. $\{S_x\}_{x \in S'}$ forms a partition of S,
  2. $(S_x, c_x) \in \text{Mon}(\sigma(x))$ for each $x \in S'$, and
  3. $c = \bigodot_{x \in S'} c_x$.

The idea of the transition relation E' is to mimic the transition function δ and to keep just the set of variables that contribute to a variable in S'.

First, we show that for any pairs $S, S' \subseteq \mathcal{X}$ and a copyless substitution σ, then there exists a unique set $\{(S_x, c_x)\}_{x \in S'}$ that satisfies Properties 1 and 2 above. In fact, given that $\{S_x\}_{x \in S'}$ forms a partition of S and $(S_x, c_x) \in \text{Mon}(\sigma(x))$ for each $x \in S'$, then it must hold that $\text{Var}(\sigma(x)) \cap S = S_x$ and, thus, each S_x is uniquely determined by S, S', σ. We conclude that E' is well defined in terms of the set $\{(S_x, c_x)\}_{x \in S'}$.

Next, we prove that for every word $w \in \Sigma^*$ we have that $\mathcal{A}(w) = \mathcal{W}(w)$ by showing the following claim. For a word w over Σ and a state $(q, S) \in Q'$, let $\text{Run}_{\mathcal{W}}(w, (q, S))$ be the set of all runs of $\mathcal{W}$ over w that stops in (q, S). Furthemore, for any run $\rho = (q_0, S_0) \xrightarrow{a_1/s_1} \cdots \xrightarrow{a_n/s_n} (q_n, S_n)$ in $\text{Run}_{\mathcal{W}}(w, (q_n, S_n))$, we denote by $[\![\rho]\!] = I'(q_0, S_0) \odot \prod_{i=1}^{n} s_i$ the partial cost of a *run* ρ of $\mathcal{W}$ over $w = a_1 \ldots a_n$.

▸ Claim 34. For any word $w = a_1 \ldots a_n \in \Sigma^*$, if $(q_0, \nu_0) \xrightarrow{a_1} \ldots \xrightarrow{a_n} (q_n, \nu_n)$ is the run of $\mathcal{A}$ over w then for every $S \subseteq \mathcal{X}$ it holds that:

$$\bigodot_{x \in S} \nu_n(x) = \bigoplus_{\rho \in \text{Run}_{\mathcal{W}}(w, (q_n, S))} [\![\rho]\!]$$

We show this claim by induction over the size of w. For the base case $w = \epsilon$, consider any set $S \subseteq \mathcal{X}$. We know that the unique run in $\text{Run}_{\mathcal{W}}(\epsilon, (q_0, S))$ is the run $\rho = (q_0, S)$. Then the claim holds by definition:

$$[\![\rho]\!] = I'(q_0, S) = \bigodot_{x \in S} \nu_0(x)$$

Now, suppose that the claim holds for any word $w = a_1 \dots a_n$ of size n and consider the word $w \cdot a$ for any $a \in \Sigma$. Take the run $(q_0, \nu_0) \xrightarrow{a_1} \dots \xrightarrow{a_n} (q_n, \nu_n) \xrightarrow{a} (q, \nu)$ of $\mathcal{A}$ over w and a set $S \subseteq \mathcal{X}$. One can easily derive the following equivalence:

$$\begin{aligned}\bigoplus_{\rho \in \mathrm{Run}_{\mathcal{W}}(wa,(q,S))} [\![\rho]\!] &= \bigoplus_{R \subseteq \mathcal{X}} E'((q_n, R), a, (q, S)) \odot \left(\bigoplus_{\rho' \in \mathrm{Run}_{\mathcal{W}}(w,(q_n,R))} [\![\rho']\!] \right) \\ &= \bigoplus_{R \subseteq \mathcal{X}} E'((q_n, R), a, (q, S)) \odot \left(\bigodot_{x \in R} \nu_n(x) \right)\end{aligned}$$

The first equivalence holds by factorizing the set $\mathrm{Run}_{\mathcal{W}}(wa, (q, S))$ under the last transition in a run and the last equivalence holds by applying the inductive hypothesis over the state (q_n, R).

The next step is to unfold the last expression by using the definition of $E'((q_n, R), a, (q, S))$. Suppose that $\delta(q_n, a) = (q, \sigma)$. Given that S is fixed and σ is determined by q_n, a, and q (i.e. is also fixed), we know that for each set R, a set $\{(S_x^R, c_x^R)\}_{x \in S}$ is uniquely determined by S, R, and σ. Thus, by the definition of $E'((q_n, R), a, (q, S))$ we get that:

$$\bigoplus_{R \subseteq \mathcal{X}} E'((q_n, R), a, (q, S)) \odot \left(\bigodot_{x \in R} \nu_n(x) \right) = \bigoplus_{R \subseteq \mathcal{X}} \bigodot_{x \in S} \left(c_x^R \odot \Big(\bigodot_{y \in S_x^R} \nu_n(y) \Big) \right)$$

Take any variable $x^* \in S$. Then the last expression can be factorize by each monomial (P, c) producing the following expression:

$$\begin{aligned}\bigoplus_{R \subseteq \mathcal{X}} \bigodot_{x \in S} \left(c_x^R \odot \Big(\bigodot_{y \in S_x^R} \nu_n(y) \Big) \right) &= \bigoplus_{(P,c) \in \mathrm{Mon}(\sigma(x^*))} c \odot \left(\bigodot_{z \in P} \nu_n(z) \right) \odot \\ &\quad \underbrace{\left(\bigoplus_{R \subseteq (\mathcal{X} - P)} \bigodot_{x \in (S - x^*)} \left(c_x^R \odot \Big(\bigodot_{y \in S_x^R} \nu_n(y) \Big) \right) \right)}_{\varphi_P}\end{aligned}$$

Note that by a monomial (P, c) for a variable x^* is independent of all the other monomials for any variable $x \in (S - x^*)$. This implies that each subexpression φ_P is equal for every monomial (P, c) of $\sigma(x^*)$.

$$\begin{aligned}\bigoplus_{R \subseteq \mathcal{X}} \bigodot_{x \in S} \left(c_x^R \odot \Big(\bigodot_{y \in S_x^R} \nu_n(y) \Big) \right) &= \left(\bigoplus_{(P,c) \in \mathrm{Mon}(\sigma(x^*))} c \odot \left(\bigodot_{z \in P} \nu_n(z) \right) \right) \odot \\ &\quad \left(\bigoplus_{R \subseteq (\mathcal{X} - \mathrm{Var}(\sigma(x^*)))} \bigodot_{x \in (S - x^*)} \left(c_x^R \odot \Big(\bigodot_{y \in S_x^R} \nu_n(y) \Big) \right) \right)\end{aligned}$$

By applying the last factorization recursively, one get the desire proof:

$$\bigoplus_{R \subseteq \mathcal{X}} \bigodot_{x \in S} \left(c_x^R \odot \big(\bigodot_{y \in S_x^R} \nu_n(y) \big) \right) = \bigodot_{x \in S} \left(\bigoplus_{(P,c) \in \mathrm{Mon}(\sigma(x))} c \odot \left(\bigodot_{y \in P} \nu_n(y) \right) \right)$$

$$= \bigodot_{x \in S} \nu(x)$$

◂

F Proof of Proposition 14

For this proof we show that any finitely ambiguous weighted automata (see Section E) can be defined by an MP-formula. Since finitely ambiguous weighted automata were shown in [14] to be equally expressive than WMSO$[\bigodot_x^1]$, this will show that WMSO$[\bigodot_x^1]$ is contained in MP.

To show that any finitely ambiguous weighted automaton can be defined by an MP-formula, we use the results in [13] and [3] combined with Theorem 9. Let $\mathcal{A}$ be a finitely ambiguous weighted automaton. In [13] it was shown that any finitely ambiguous weighted automaton $\mathcal{A}$ can be written as a disjoint union of unambiguous weighted automata $\mathcal{A}_1, \ldots, \mathcal{A}_k$ such that, for every word $w \in \Sigma^*$, it holds that:

$$[\![\mathcal{A}]\!](w) = [\![\mathcal{A}_1]\!](w) \oplus \cdots \oplus [\![\mathcal{A}_k]\!](w)$$

In [3], it was shown that every unambiguous weighted automaton $\mathcal{A}_i$ is equivalent to a copyless CRA $\mathcal{B}_i$ that uses just the $\odot$-operations. In particular, $\mathcal{B}_i$ is a bounded alternation copyless CRA with just one alternation. By Theorem 9, we know that each $\mathcal{B}_i$ is equivalent to an MP-formula φ_i. Then it is straightforward to show that $\mathcal{A}$ is equivalent to the formula $\varphi_1 \oplus \cdots \oplus \varphi_k$.