Maximal Partition Logic: Towards a Logical Characterization of Copyless Cost Register Automata

Filip Mazowiecki[1] and Cristian Riveros[2]

1 **University of Warsaw, Poland**
2 **Pontificia Universidad Católica de Chile, 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.CSL.2015.144

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.

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 144–159

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

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:

$$R := \varnothing \mid \epsilon \mid a \mid R \cdot R \mid R + R \mid R^*$$

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:

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

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 $[\![\hat{\nu}_n(\mu(q_n))]\!]$.

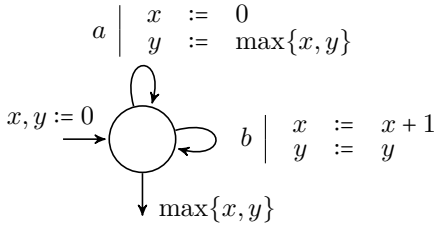
The run of $\mathcal{A}$ over w can be equally defined in terms of ground expressions rather than values. A ground configuration of $\mathcal{A}$ is a tuple (q, ς) where $q \in Q$ and $\varsigma \in \mathrm{Subs}(\mathcal{X})$ is a ground substitution. Given a string $w = a_1 \dots a_n \in \Sigma^*$, 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.

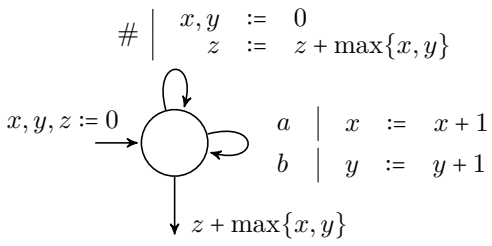
▶ **Example 1.** Let $\mathbb{S}$ be the max-plus semiring $\mathbb{N}_{-\infty}(\max, +)$ and $\Sigma = \{a, b\}$. Consider the function f_1 that for a given string $w \in \Sigma^*$ computes the longest substring of b's. This can be easily defined by the following CRA $\mathcal{A}_1$ with two registers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



$\mathcal{A}_1$ 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 $w_0 \# w_1 \# \dots \# w_n$ 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 $\text{Int}(w)$ 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:

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

▶ **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 $\text{Max}_\sqsubseteq(I)$ 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:

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

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:

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

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:

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

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 $\text{Max}\ \mathtt{R}.\ \varphi$, $\sum \mathtt{R}.\ \varphi$ 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 $\text{Max}\ \ b^+$ 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 $\mathbb{N}_{-\infty}(\max,+)$:

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

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 $\max\left\{\ \sum a.\ 1\ ,\ \sum b.\ 1\ \right\}$ 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: $w \vDash \varphi(i,j)$. 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 $x \leq y$.

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 $w \vDash \varphi(i,j)$, and at most one $j' \in \mathrm{dom}(w)$ such that $w \vDash \varphi(j',i)$; 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:

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

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* $\mathtt{R}_1,\ldots,\mathtt{R}_n$ *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 $\mathrm{Expr}_0^{\otimes}(\mathcal{X})$ with 0-alternation by $\mathrm{Expr}_0^{\otimes} = \mathcal{X} \cup \mathbb{S}$. For any $N \geq 1$, we define the set of expressions $\mathrm{Expr}_N^{\otimes}(\mathcal{X})$ as the $\otimes$-closure of $\mathrm{Expr}_{N-1}^{\otimes}(\mathcal{X})$, namely, $\mathrm{Expr}_N^{\otimes}(\mathcal{X})$ is the minimal set of expressions that contains $\mathrm{Expr}_{N-1}^{\otimes}(\mathcal{X})$ and satisfies $e_1 \otimes e_2 \in \mathrm{Expr}_N^{\otimes}(\mathcal{X})$ for all $e_1, e_2 \in \mathrm{Expr}_N^{\otimes}(\mathcal{X})$. We denote by $\mathrm{Expr}_N(\mathcal{X}) = \mathrm{Expr}_N^{\oplus}(\mathcal{X}) \cup \mathrm{Expr}_N^{\odot}(\mathcal{X})$ 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 $\varphi = \otimes R\langle S\rangle T.\ \psi$ 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]):

$$\theta \coloneqq \varphi \mid s \mid (\theta \oplus \theta) \mid (\theta \odot \theta) \mid \bigoplus x.\ \theta(x) \mid \bigodot x.\ \theta(x) \mid \bigoplus X.\ \theta(X)$$

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 $w \vDash \varphi(\sigma)$ and $\mathbb{0}$ otherwise. The semantics of a quantitative formula θ over w and σ is defined as follows.

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

▶ **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:

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

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 $\mathrm{WMSO}[\bigoplus_X \bigodot_x^1]$. 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 $\mathrm{WMSO}[\bigodot_x^1]$ and $\mathrm{WMSO}[\bigoplus_x \bigodot_x^1]$. By results in [13, 14] this shows that in terms of expressiveness, these fragments are strictly contained in each other:

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

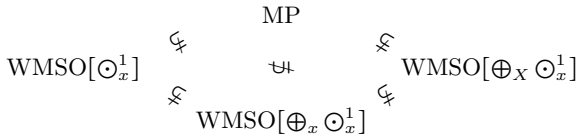
We compare the expressiveness of MP with WMSO by exploiting the relation with copyless CRA (Theorem 9). The first question is whether MP is more expressive than $\mathrm{WMSO}[\bigoplus_X \odot_x^1]$. At a first sight, one could believe that this is possible, since the syntax of MP is symmetric with 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 $\mathrm{WMSO}[\bigoplus_X \odot_x^1]$. 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* $\mathrm{WMSO}[\bigoplus_X \odot_x^1]$ *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 $\mathrm{WMSO}[\odot_x^1]$. 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* $\mathrm{WMSO}[\odot_x^1]$ *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 $\mathrm{WMSO}[\odot_x^1]$ 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 $\mathrm{WMSO}[\bigoplus_X \odot_x^1]$ and, moreover, is does not contain $\mathrm{WMSO}[\bigoplus_x \odot_x^1]$. Summing up, we get the following diagram representing the expressiveness of MP in terms of WMSO.



We conjecture that MP is not contained in $\mathrm{WMSO}[\bigoplus_x \odot_x^1]$, 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 Shaull Almagor, Udi Boker, and Orna Kupferman. What's decidable about weighted automata? In *ATVA*, pages 482–491, 2011.

3 Rajeev Alur, Loris D'Antoni, Jyotirmoy Deshmukh, Mukund Raghothaman, and Yifei 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 Benedikt Bollig, Paul Gastin, Benjamin Monmege, and Marc 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 Thomas Colcombet, Clemens Ley, and Gabriele Puppis. On the use of guards for logics with data. In *Mathematical Foundations of Computer Science 2011*, pages 243–255. Springer, 2011.

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

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

8 Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata*. Springer, 1st edition, 2009.

9 Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn 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 Jeffrey Friedl. *Mastering regular expressions*. " O'Reilly Media, Inc.", 2006.

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

12 Benny 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 Ines Klimann, Sylvain Lombardy, Jean Mairesse, and Christophe Prieur. Deciding unambiguity and sequentiality from a finitely ambiguous max-plus automaton. *Theor. Comput. Sci.*, 327(3):349–373, 2004.

14 Stephan Kreutzer and Cristian 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 Daniel Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. In *ICALP*, pages 101–112, 1992.

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

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

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

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

20 Jacques 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.