Regular Functions, Cost Register Automata, and Generalized Min-Cost Problems

Rajeev Alur Loris D'Antoni Jyotirmoy V. Deshmukh Mukund Ragothaman Yifei Yuan

November 5, 2012

Abstract

Motivated by the successful application of the theory of regular languages to formal verification of finite-state systems, there is a renewed interest in developing a theory of analyzable functions from strings to numerical values that can provide a foundation for analyzing *quantitative* properties of finite-state systems. In this paper, we propose a deterministic model for associating costs with strings that is parameterized by operations of interest (such as addition, scaling, and min), a notion of *regularity* that provides a yardstick to measure expressiveness, and study decision problems and theoretical properties of resulting classes of cost functions. Our definition of regularity relies on the theory of string-to-tree transducers, and allows associating costs with events that are conditional upon regular properties of future events. Our model of *cost register automata* allows computation of regular functions using multiple "write-only" registers whose values can be combined using the allowed set of operations. We show that classical shortest-path algorithms as well as algorithms designed for computing *discounted costs*, can be adopted for solving the min-cost problems for the more general classes of functions specified in our model. Cost register automata with min and increment give a deterministic model that is equivalent to *weighted automata*, an extensively studied nondeterministic model, and this connection results in new insights and new open problems.

1 Introduction

1.1 Motivation

The classical shortest path problem is to determine the minimum-cost path in a finite graph whose edges are labeled with costs from a numerical domain. In this formulation, the cost at a given step is determined locally, and this does not permit associating alternative costs in a speculative manner. For example, one cannot specify that "the cost of an event e is 5, but it can be reduced to 4 provided an event e' occurs sometime later." Such a constraint can be captured by the well-studied framework of *weighted automata* [31, 14]. A weighted automaton is a *nondeterministic* finite-state automaton whose edges are labeled with symbols in a finite alphabet Σ and costs in a numerical domain. Such an automaton maps a string w over Σ to the minimum over costs of all accepting paths of the automaton over w. There is extensive literature on weighted automata with applications to speech and image processing [26]. Motivated by the successful application of the theory of regular languages to formal verification of finite-state systems, there is a renewed interest in weighted automata as a plausible foundation for analyzing *quantitative* properties (such as power consumption) of finite-state systems [9, 5, 1]. Weighted automata, however, are inherently nondeterministic, and are restricted to cost domains that support two operations with the algebraic structure of a *semiring*, one operation for summing up costs along a path (such as $+$), and one for aggregating costs of alternative paths (such as min). Thus, weighted automata, and other existing frameworks (see [11, 29]), do not provide guidance on how to combine and define costs in presence of multiple operations such as paying incremental costs, scaling by discounting factors, and choosing minimum. In particular, one cannot specify that "the cost of an event e is 10, but for every future occurrence of the event e', we offer a refund of 5% to the entire cost accumulated until e." The existing work on "generalized shortest paths" considers extensions that allow costs with future discounting, and while it presents interesting polynomial-time algorithms [19, 30], does not attempt to identify the class of models for which these algorithmic ideas are applicable. This motivates the problem we address: *what is a plausible definition of regular functions from strings to cost domains, and how can such functions be specified and effectively analyzed?*

1.2 Proposed Definition of Regularity

When should be a function from strings to a cost domain, say the set $\mathbb{N}$ of natural numbers, be considered *regular*? Ideally, we wish for an abstract machine-independent definition with appealing closure properties and decidable analysis questions. We argue that the desired class of functions is parameterized by the operations supported on the cost domain. Our notion of regularity is defined with respect to a regular set T of terms specified using a grammar. For example, the grammar $t := +(t,t) \mid c$ specifies terms that can be built from constants using a binary operator $+$, and the grammar $t := +(t,c) \mid *(t,d) \mid c$ specifies terms that can be built from constants using two binary operators $+$ and $*$ in a left-linear manner. Given a function g that maps strings to terms in T, and an interpretation $[\![.]\!]$ for the function symbols over a domain $\mathbb{D}$, we can define a cost function f that maps a string w to the value $[\![g(w)]\!]$. The theory of tree transducers, developed in the context of syntax-directed program transformations and processing of XML documents, suggests that the class of *regular* string-to-term transformations has the desired trade-off between expressiveness and analyzability as it has appealing closure properties and multiple characterizations using transducer models as well as Monadic-Second-Order logic [17, 12, 4, 20]. As a result, *we call a cost function*

f from strings to a cost domain $\mathbb{D}$ regular with respect to a set T of terms and an interpretation $[\![.]\!]$ for the function symbols, exactly when f can be expressed as a composition of a regular function from strings to T and evaluation according to $[\![.]\!]$.

1.3 Machine Model: Cost Register Automata

Having chosen a notion of regularity as a yardstick for expressiveness, we now need a corresponding machine model that associates costs with strings in a natural way. Guided by our recent work on *streaming transducers* [3, 4], we propose the model of *cost register automata*: a CRA is a deterministic machine that maps strings over an input alphabet to cost values using a finite-state control and a finite set of cost registers. At each step, the machine reads an input symbol, updates its control state, and updates its registers using a parallel assignment, where the definition is parameterized by the set of expressions that can be used in the assignments. For example, a CRA with increments can use multiple registers to compute alternative costs, perform updates of the form $x := y + c$ at each step, and commit to the cost computed in one of the registers at the end. Besides studying CRAs with operations such as increment, addition, min, and scaling by a factor, we explore the following two variants. First, we consider models in which registers hold not only cost *values*, but (unary) *cost functions*: we allow registers to hold pairs of values, where a pair (c, d) can represent the linear function $f(n) = c + n * d$. Operations on such pairs can simulate "substitution" in trees, and allow computing with contexts, where parameters can be instantiated at later steps. Second, we consider "copyless" models where each register can be used at most once in the right-hand-sides of expressions updating registers at any step. This "single-use-restriction", known to be critical in theory of regular tree transducers [17], ensures that costs (or the sizes of terms that capture costs) grow only linearly with the length of the input.

1.4 Contributions

In Section 4, we study the class of cost functions over a domain $\mathbb{D}$ with a commutative associative function $\otimes$; in Section 5, we study the class of cost functions over a semiring structure with domain $\mathbb{D}$ and binary operations $\oplus$ (such as min) and $\otimes$ (such as addition); and in Section 6, we consider different forms of discounted cost functions with scaling and addition. In each case, we identify the operations a CRA must use for expressiveness equivalent to the corresponding class of regular functions, and present algorithms for computing the min-cost value and checking equivalence of CRAs. We summarize some interesting insights that emerge from our results about specific cost models. First, our notion of regularity implies that regular cost functions are closed under operations such as string reversal and regular look-ahead, leading to an appealing symmetry between past and future. Second, the use of multiple registers and explicit combinators allows CRAs to compute all regular functions in a deterministic manner. Third, despite this added expressiveness, decision problems for CRAs are typically analyzable. In particular, we get algorithms for solving min-cost problems for more general ways of specifying discounting than known before. Fourth, since CRAs "construct" costs over an infinite domain, it suffices to use registers in a "write-only" mode without any tests. This critically distinguishes our model from the well-studied models of *register machines*, and more recently, *data automata* [21, 29, 7]: a data automaton accepts strings over an infinite alphabet, and the model allows at least testing equality of data values leading to mostly negative results regarding decidability. Fifth, it is known that weighted automata are not determinizable, which has sparked extensive research [14, 24]. Our results show that in presence of

multiple registers that can be updated by explicitly applying both the operations of the semiring, classical subset construction can be modified to get a deterministic machine. Finally, the class of regular functions over the semiring turns out to be a *strict* subset of functions definable by weighted automata due to the copyless (or linear) restriction. It is known that checking equivalence of weighted automata over the Min-Plus semiring (natural numbers with min and addition) is undecidable [25, 1]. Existing proofs critically rely on the "copyful" nature raising the intriguing prospect that equivalence is decidable for regular functions over the Min-Plus semiring.

2 Cost Register Automata

2.1 Cost Grammars

A ranked alphabet F is a set of function symbols, each of which has a fixed arity. The arity-0 symbols, also called constants, are mapped to domain elements. To allow infinite domains such as the set $\mathbb{N}$ of natural numbers, we need a way of encoding constants as strings over a finite set of symbols either in unary or binary, but we suppress this detail, and assume that there are infinitely many constant symbols. The set T_F of terms over a ranked alphabet is defined in the standard fashion: if c is a constant symbol in F, then $c \in T_F$, and if $t_1, \ldots, t_k \in T_F$ and f is an arity-k symbol in F, for $k > 0$, then $f(t_1, \ldots, t_k) \in T_F$. A *cost grammar* G is defined as a tuple (F, T) where F is a ranked alphabet and T is a *regular* subset of T_F. In this paper, we define this regular subset using a grammar containing a single nonterminal. In particular, we focus on the following grammars: For a binary function symbol $+$, the terms of the *additive-grammar* $G(+)$ are specified by $t := +(t, t) \,|\, c$, where c is a constant, and the terms of the *increment-grammar* $G(+c)$ are specified by $t := +(t, c) \,|\, c$. Given binary functions min and $+$, the terms of the *min-inc-grammar* $G(\min, +c)$ are given by $t := \min(t, t) \,|\, +(t, c) \,|\, c$, which restricts the use of addition operation. Given binary functions $+$ and $*$, the terms of the *inc-scale Grammar* $G(+c, *d)$ are generated by the left-linear grammar $t := +(t, c) \,|\, *(t, d) \,|\, c$, that uses both operations in a restricted manner, and c and d denote constants, ranging over possibly different subsets of domain elements.

2.2 Cost Models

Given a cost grammar $G = (F, T)$, a cost model $\mathbb{C}$ is defined as the tuple $(G, \mathbb{D}, [\![.]\!])$, where the *cost domain* $\mathbb{D}$ is a finite or infinite set. For each constant c in F, $[\![c]\!]$ is a unique value in the domain $\mathbb{D}$, and for each function symbol f of arity k, $[\![f]\!]$ defines a function $[\![f]\!] : \mathbb{D}^k \mapsto \mathbb{D}$. We can inductively extend the definition of $[\![.]\!]$ to assign semantics to the terms in T in a standard fashion. For a numerical domain $\mathbb{D}$ such as $\mathbb{N}$ (the set of natural numbers) and $\mathbb{Z}$ (the set of integers), we use $\mathbb{C}(\mathbb{D}, +)$ to denote the cost model with the cost grammar $G(+)$, domain $\mathbb{D}$, and $[\![+]\!]$ as the standard addition operation. Similarly, $\mathbb{C}(\mathbb{N}, +c)$ denotes the cost model with the cost grammar $G(+c)$, domain $\mathbb{N}$, and $[\![+]\!]$ as the standard addition operation. For cost grammars with two operations we denote the cost model by listing the domain, the two functions, and sometimes the subdomain to restrict the set of constants used by different rules: $\mathbb{C}(\mathbb{Q}^+, +c, [0, 1], *d)$ denotes the cost model with the cost grammar $G(+c, *d)$, the set $\mathbb{Q}^+$ of non-negative rational numbers as the domain, $+$ and $*$ interpreted as standard addition and multiplication, and the rational numbers in the interval $[0, 1]$ as the range of encodings corresponding to the scaling factor d.

2.3 Cost Register Automata

A *cost register automaton* (CRA) is a deterministic machine that maps strings over an input alphabet to cost values using a finite-state control and a finite set of cost registers. At each step, the machine reads an input symbol, updates its control state, and updates its registers using a parallel assignment. It is important to note that the machine does not test the values of registers, and thus the registers are used in a "write-only" mode. The definition of such a machine is parameterized by the set of expressions that can be used in the assignments. Given a set X of registers and a cost grammar G, we define the set of assignment expressions $E(G, X)$ by extending the set of terms in G so that each internal node can be replaced by a register name. For example, for the additive grammar $G(+)$, we get the set $E(+, X)$ of expressions defined by the grammar $e := +(e, e) \mid c \mid x$, for $x \in X$; and for the $G(+c, *d)$, we get the expressions $E(+c, *d, X)$ defined by the grammar $e := +(e, c) \mid *(e, d) \mid c \mid x$, for $x \in X$. We assume that the ranked alphabet contains a special constant symbol, denoted 0, used as the initial value of the registers.

Formally, a cost register automaton M over a cost grammar G is a tuple $(\Sigma, Q, q_0, X, \delta, \rho, \mu)$ where Σ is a finite input alphabet, Q is a finite set of states, $q_0 \in Q$ is the initial state, X is a finite set of registers, $\delta : Q \times \Sigma \mapsto Q$ is the state-transition function, $\rho : Q \times \Sigma \times X \mapsto E(G, X)$ is the register update function, and $\mu : Q \mapsto E(G, X)$ is a *partial* final cost function.

The semantics of such an automaton is defined with respect to a cost model $\mathbb{C} = (G, \mathbb{D}, [\![.]\!])$, and is a partial function $[\![M, \mathbb{C}]\!]$ from Σ^* to $\mathbb{D}$. A configuration of M is of the form (q, ν), where $q \in Q$ and the function $\nu : X \mapsto \mathbb{D}$ maps each register to a cost in $\mathbb{D}$. Valuations naturally map expressions to cost values using the interpretation of function symbols given by the cost model. The initial configuration is (q_0, ν_0), where ν_0 maps each register to the initial constant 0. Given a string $w = a_1 \dots a_n \in \Sigma^*$, the run of M on w is a sequence of configurations $(q_0, \nu_0) \dots (q_n, \nu_n)$ such that for $1 \leq i \leq n$, $\delta(q_{i-1}, a_i) = q_i$ and for each $x \in X$, $\nu_i(x) = [\![\nu_{i-1}(\rho(q_{i-1}, a, x))]\!]$. The output of M on w, denoted by $[\![M, \mathbb{C}]\!](w)$, is undefined if $\mu(q_n)$ is undefined, and otherwise it equals $[\![\nu_n(\mu(q_n))]\!]$.

2.4 CRA-definable Cost Functions

Each cost model $\mathbb{C} = (G, \mathbb{D}, [\![.]\!])$ defines a class of cost functions $\mathbb{F}(\mathbb{C})$: a partial function f from Σ^* to $\mathbb{D}$ belongs to this class iff there exists a CRA M over the cost grammar G such that f equals $[\![M, \mathbb{C}]\!]$. The class of cost functions corresponding to the cost model $\mathbb{C}(\mathbb{N}, +)$ is abbreviated as $\mathbb{F}(\mathbb{N}, +)$, the class corresponding to the cost model $\mathbb{C}(\mathbb{Q}^+, +c, [0, 1], *d)$ as $\mathbb{F}(\mathbb{Q}^+, +c, [0, 1], *d)$, etc.

2.5 Examples

Figure 1 shows examples of cost register automata for $\Sigma = \{a, b, e\}$. Consider the cost function f_1 that maps a string w to the length of the substring obtained by deleting all b's after the last occurrence of e in w. The automaton M_1 computes this function using two cost registers and increment operation. The register y is incremented on each symbol, and hence equals the length of string processed so far. The register x is not incremented on b symbols, but is updated to the total length stored in y when e symbol is encountered. This example illustrates the use of two registers: the computation of the desired function f_1 in register x update crucially relies on the auxiliary register y.

For a string w and symbol a, let $|w|_a$ denote the count of a symbols in w. For a given string w of the form $w_1\, e\, w_2 \dots e\, w_{n-1}\, e\, w_n$, where each block w_i contains only a's and b's, let $f_2(w)$ be

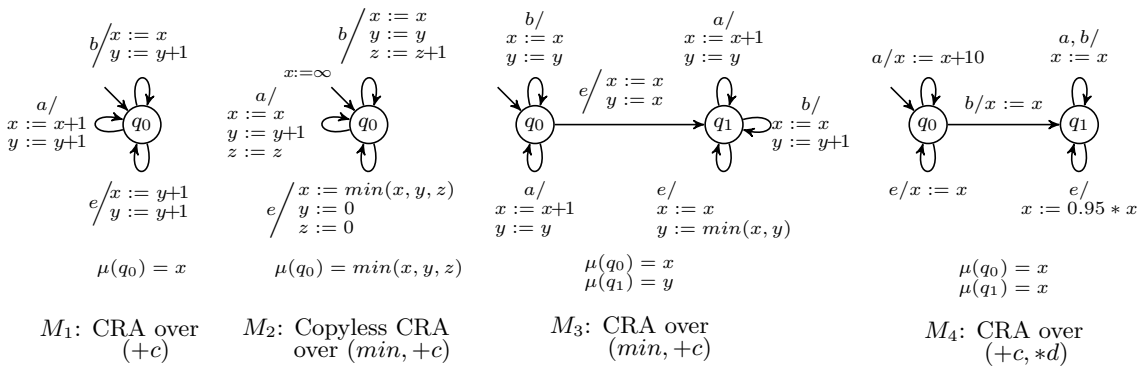


Figure 1: Examples of Cost Register Automata

the minimum of the set $\{|w_i|_a, |w_i|_b\}$. The CRA M_2 over the grammar $G(\min, +c)$ computes this function using three registers by an explicit application of the min operator.

For a given string $w = w_1\, e\, w_2\, e \ldots e\, w_n$, where each w_i contains only a's and b's, consider the function f_3 that maps w to $\min_{j=1}^{n-1}(|w_1|_a + |w_2|_a + \cdots |w_j|_a + |w_{j+1}|_b + \cdots + |w_n|_b)$. This function is computed by the CRA M_3 over the grammar $G(\min, +c)$.

The final example concerns use of scaling. Consider a computation where we wish to charge a cost of 10 upon seeing an a event until a b event occurs. Once a b event is triggered, for every subsequent e event, the cost is discounted by 5%. Such a cost function is computed by the CRA M_4 over the grammar $G(+c, *d)$.

2.6 Copyless Restriction

A CRA M is said to be *copyless* if each register is used at most once at every step: for each state q and input symbol a and each register $x \in X$, the register x appears at most once in the set of expressions $\{\delta(q, a, y) \mid y \in X\}$ and x appears at most once in the output expression $\mu(q)$. Each cost model $\mathbb{C} = (G, \mathbb{D}, [\![.]\!])$ then defines another class of cost functions $\mathbb{F}^c(\mathbb{C})$: a partial function f from Σ^* to $\mathbb{D}$ belongs to this class iff there exists a copyless CRA M over the cost grammar G such that f equals $[\![M, \mathbb{C}]\!]$. In Figure 1, the automata for function f_2 and f_4 are copyless, while the ones for f_1 and f_3 are not.

2.7 Regular Look-Ahead

A CRA M^R with *regular look-ahead* (CRA-RLA) is a CRA that can make its decisions based on whether the remaining suffix of the input word belongs to a regular language. Let L be a regular language, and let A be a DFA for $reverse(L)$ (such a DFA exists, since regular languages are closed under the reverse operation). Then, while processing an input word, testing whether the suffix $a_j \ldots a_k$ belongs to L corresponds to testing whether the state of A after processing $a_k \ldots a_j$ is an accepting state of A. We now try to formalize this concept Let $w = a_1 \ldots a_k$ be a word over Σ, and let A be a DFA with states R processing words over Σ. Then the *A-look-ahead labeling* of w, is the word $w_A = r_1 r_2 \ldots r_k$ over the alphabet R such that for each position $1 \leq j \leq k$, the corresponding symbol is the state of the DFA A after reading $a_k \ldots a_j$ (it reads the reverse of the word). A CRA-RLA consists of an DFA A over Σ with states R, and a CRA M over the input alphabet R.

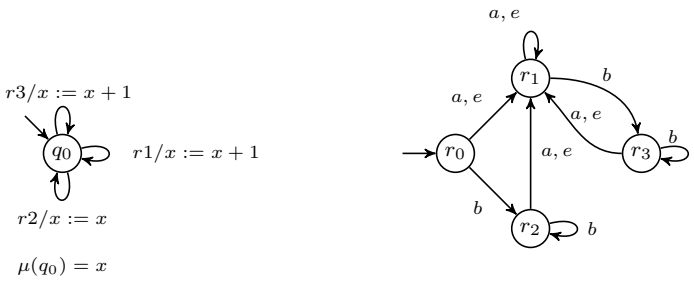


Figure 2: On the left a CRA-RLA over $(+c)$ corresponding to M_1 in Figure 1. On the right the corresponding labeling automaton. The states of A correspond to the languages used in the informal description of M_1.

The output of CRA-RLA (M, A) on w, denoted by $[\![(M, A), \mathbb{C}]\!](w)$, is defined as $[\![M, \mathbb{C}]\!](w_A)$. In Figure 2 we show the CRA-RLA for M_1 of Figure 1.

3 Regular Cost Functions

Consider a cost grammar $G = (F, T)$. The terms in T can be viewed as trees: an internal node is labeled with a function symbol f of arity $k > 0$ and has k children, and each leaf is labeled with a constant. A *deterministic streaming string-to-tree transduction* is a (partial) function $f : \Sigma^* \mapsto T$. The theory of such transductions has been well studied, and in particular, the class of *regular* string-to-tree transductions has appealing closure properties, and multiple characterizations using Macro-tree-transducers (with single-use restriction and regular look-ahead) [17], Monadic-Second-Order logic definable graph transformations [12], and streaming tree transducers [4]. We first briefly recap the model of streaming string-to-tree transducers.

3.1 Streaming String-to-Tree Transducers (SSTT)

A streaming string-to-tree transducer is a deterministic machine model that can compute regular transformations from strings to ranked trees in a single pass. An SSTT can be viewed as a variant of CRA where each register stores a term, that is, an *uninterpreted expression*, and these terms are combined using the rules allowed by the grammar. To obtain a model whose expressiveness coincides with the regular transductions, we must require that the updates are copyless, but need to allow terms that contain "holes", *i.e.*, parameters that can be substituted by other terms.

Let $G = (F, T)$ be a cost grammar. Let ? be a special 0-ary symbol that denotes a place holder for the term to be substituted later. We obtain the set $T^?$ by adding the symbol ? to F, and requiring that each term has at most one leaf labeled with ?. For example, for the cost grammar $G(+c)$, the set $T^?$ of parameterized terms is defined by the grammar $t := +(t, c) \,|\, c \,|\, ?$; and for the cost grammar $G(+)$, the set $T^?$ of parameterized terms is defined by the grammar $t' := +(t, t') \,|\, +(t', t) \,|\, c \,|\, ?$, where t stands for (complete) terms generated by the original grammar $t := +(t, t) \,|\, c$. A parameterized term such as $\min(5, ? + 3)$ stands for an incomplete expression, where the parameter ? can be replaced by another term to complete the expression. Registers of an SSTT hold parameterized terms. The expressions used to update the registers at every step are given by the cost grammar, with an additional rule for *substitution*: given a parameterized expression e

and another expression e', the expression $e[e']$ is obtained by substituting the sole ?-labeled leaf in e with the expression e'.

Given a set X of registers, the set $E^?(G, X)$ represents parameterized expressions that can be obtained using the rules of G, registers in X, and substitution. For example, for the grammar $G(+c)$ and a set X of registers, the set $E^?(+c, X)$ is defined by the grammar $e := +(e, c) \mid c \mid ? \mid x \mid e[e]$, for $x \in X$. The output of an SSTT is a (complete) term in T defined using the final cost function. The register update function and the final cost function are required to be *copyless*: each register is used at most once on the right-hand-side in any transition. The semantics of an SSTT gives a partial function from Σ^* to T. We refer the reader to [4] for details.

3.2 Regular Cost Functions

Let Σ be a finite input alphabet. Let $\mathbb{D}$ be a cost domain. A *cost function* f maps strings in Σ^* to elements of $\mathbb{D}$. Let $\mathbb{C} = (G, \mathbb{D}, [\![.]\!])$ be a cost model. A cost function f is said to be *regular with respect to the cost model* $\mathbb{C}$ if there exists a regular string-to-tree transduction g from Σ^* to T such that for all $w \in \Sigma^*$, $f(w) = [\![g(w)]\!]$. That is, given a cost model, we can define a cost function using an SSTT: the SSTT maps the input string to a term, and then we evaluate the term according to the interpretation given by the cost model. The cost functions obtained in this manner are the regular functions. We use $\mathbb{R}(\mathbb{C})$ to denote the class of cost functions regular with respect to the cost model $\mathbb{C}$.

As an example, suppose $\Sigma = \{a, b\}$. Consider a vocabulary with constant symbols 0, c_a and c_b, and the grammar $G(+c)$. Consider the SSTT U with a single register that is initialized to 0, and at every step, it updates x to $+(x, c_a)$ on input a, and $+(x, c_b)$ on input b. Given input $w_1 \dots w_n$, the SSTT generates the term $e = +(\cdots + (+(0, c_1), c_2) \cdots c_n)$, where each $c_i = c_a$ if $w_i = a$ and $c_i = c_b$ otherwise. To obtain the corresponding cost function, we need a cost model that interprets the constants and the function symbol $+$, and we get the cost of the input string by evaluating the expression e. Now, consider another SSTT U' that uses a single register initialized to ?. At every step, it updates x to $x[+(?, c_a)]$ on input a, and $x[+(?, c_b)]$ on input b, using the substitution operation. The output is the term $x[0]$ obtained by replacing the parameter by 0. Given input $w_1 \dots w_n$, the SSTT generates the term $e' = +(\cdots + (0, c_n), \cdots c_1)$, where each $c_i = c_a$ if $w_i = a$ and $c_i = c_b$ otherwise. Note that the SSTT U builds the cost term by adding costs on the right, while the SSTT U' uses parameter substitution to build costs terms in the reverse order. If the interpretation of the function $+$ is not commutative, then these two mechanisms allow to compute different functions, both of which are regular.

Let's now consider a different grammar. Consider the constant symbols 1, c_a and c_b, and the grammar $G(+c, *d)$. Consider the SSTT U with a single register that is initialized to 1 and, at every step, it updates x to $+(*(x, c_a), 1)$ on input a and to $+(*(x, c_b), 1)$ on input b. Given input $w_1 \dots w_n$, the SSTT generates the term $e = +(*(+(*(\cdots + (*(1, c_b), 1) \cdots, c_{n-1}), 1), c_n), 1)$, where each $c_i = c_a$ if $w_i = a$ and $c_i = c_b$ otherwise. Now, consider another SSTT U' that uses a single register initialized to ?. At every step, it updates x to $x[+(*(?, c_a), 1)]$ on input a, and $x[+(?, c_b)]$ on input b, using the substitution operation. The output is the term $x[1]$ obtained by replacing the parameter by 1. Given input $w_1 \dots w_n$, the SSTT generates the term $e = +(*(+(*(\cdots + (*(1, c_b), 1) \cdots, c_2), 1), c_1), 1)$, where each $c_i = c_a$ if $w_i = a$ and $c_i = c_b$ otherwise. The same considerations as before follow.

3.3 Closure Properties

If f is a regular cost function from Σ^* to a cost domain $\mathbb{D}$, then the *domain* of f, *i.e.*, the set of strings w such that $f(w)$ is defined, is a regular language. Closure properties for regular string-to-tree transductions immediately imply certain closure properties for regular cost functions. For a string w, let w^r denote the reverse string. We define a *reverse function* $f^r : \Sigma^* \mapsto \mathbb{D}$ such that for all $w \in \Sigma^*$, $f^r(w) = f(w^r)$. Given cost functions f_1, f_2 from Σ^* to $\mathbb{D}$ and a language $L \subseteq \Sigma^*$, the *choice* function "if L then f_1 else f_2" maps an input string w to $f_1(w)$ if $w \in L$, and to $f_2(w)$ otherwise. If the two cost functions f_1 and f_2 are regular and if L is a regular language, then the choice function is also regular. We now show that regular cost functions are closed under reverse and regular choice.

Theorem 1 (Closure Properties of Regular Cost Functions) *For every cost model $\mathbb{C}$,*

(a) if a cost function f belongs to the class $\mathbb{R}(\mathbb{C})$, then so does the function f^r;

(b) if cost functions f_1 and f_2 belong to the class $\mathbb{R}(\mathbb{C})$, then so does the function "if L then f_1 else f_2" for every regular language L.

Proof. (a) The proof of the first statement follows from the application of theorems from [2, 4]. Let f be a regular cost function belonging to the class $\mathbb{R}(\mathbb{C})$, and let U be an SSTT that computes the function f. A streaming string transducer (SST) is a machine similar to an SSTT. It maps strings to strings with the help of a fixed number of registers that store strings, and uses updates that involve concatenating registers and strings in a copyless fashion. Given an alphabet Σ, computing the reverse w^r of strings $w \in \Sigma$ is an SST-definable transduction [2]. Let SST U' that computes such a transduction. As proved in [4], the composition of an SST-definable transduction and an SSTT definable transduction is an SSTT definable transduction. Thus the sequential composition of U' and U is also SSTT definable. Thus, for every regular function f in $\mathbb{R}(\mathbb{C})$, we can construct the SSTT $U \circ U'$ that maps every string w to the corresponding term $f(w^r)$, which is, by definition, the cost function $f^r(w)$. Thus, if f belongs to $\mathbb{R}(\mathbb{C})$, so does f^r.

(b) To prove the second statement, we show how we can construct an SSTT U_c that defines the function "if L then f_1 else f_2." As f_1 and f_2 are regular cost functions, they are definable by SSTT U_1 and U_2 respectively. The SSTT U_c maintains 2 disjoint sets of registers, where the first set corresponds to registers of U_1 and the second to the registers of U_2. A state of U_c is a tuple (q, q_1, q_2), where q_1 and q_2 exactly track the states of U_1 and U_2, and q is the state of the DFA corresponding to the regular language L. The output function of U_c is defined such that if the input word w is in L (*i.e.*, it is accepted by the corresponding DFA), then U_c uses the first set of registers to compute the output, and, otherwise uses the second set of registers. For any cost model, the U_c exactly defines the choice function. □

An SSTT with regular look ahead is a pair (U, A) where U is an SSTT and A a DFA. As discussed earlier regular-look-ahead tests allow machines to make its decisions based on whether the remaining suffix of the input word belongs to a given regular language. SSTT are closed under the operation of regular-look-ahead [4], which implies the same for regular cost functions.

Theorem 2 (Closure Under RLA [4]) *For every* SSTT *with regular-look-ahead (U, A), there exists an* SSTT *U' without regular-look-ahead which computes the same function.*

3.4 Constant Width and Linear Size of Output Terms

While processing symbols of an input string w, in each step an SSTT performs a copyless update. Thus, the sum of the sizes of all terms stored in registers grows only by a constant additive factor. It follows that $|U(w)|$ is $O(|w|)$. Viewed as a tree, the depth of $U(w)$ can be linear in the length of w, but its width is constant, bounded by the number of registers. This implies that if f is a cost function in $\mathbb{R}(\mathbb{N}, +c)$, then $|f(w)|$ must be $O(|w|)$. In particular, the function $f(w) = |w|^2$ is not regular in this cost model. Revisiting the examples in Sec. 2, it turns out that the function f_1 is regular for $\mathbb{C}(\mathbb{N}, +c)$, and the function f_2 is regular for $\mathbb{C}(\mathbb{N}, \min, +c)$. The function f_3 does not appear to be regular for $\mathbb{C}(\mathbb{N}, \min, +c)$, as it seems to require $O(|w|^2)$ terms to construct it.

4 Commutative-Monoid Cost Functions

In this section we explore and analyze cost functions for the cost models of the form $(\mathbb{D}, \otimes)$, where $\mathbb{D}$ is a cost domain (with a designated identity element) and the interpretation $[\![\otimes]\!]$ is a commutative and associative function.

4.1 Expressiveness

Given a cost model $(\mathbb{D}, \otimes)$, we can use regular string-to-tree transductions to define two (machine-independent) classes of functions: the class $\mathbb{R}(\mathbb{D}, \otimes c)$ defined by the grammar $G(\otimes c)$ and the class $\mathbb{R}(\mathbb{D}, \otimes)$ defined by the grammar $G(\otimes)$. Relying on commutativity and associativity, we show these two classes to be equally expressive. This class of "regular additive cost functions" corresponds exactly to functions computed by CRAs with increment operation, and also, by copyless-CRAs with addition. We are going to show the result via intermediate results.

Lemma 3 *For cost domain $\mathbb{D}$ with a commutative and associative operation $[\![\otimes]\!]$, $\mathbb{R}(\mathbb{D}, \otimes c) = \mathbb{R}(\mathbb{D}, \otimes)$.*

Proof. $\mathbb{R}(\mathbb{D}, \otimes c) \subseteq \mathbb{R}(\mathbb{D}, \otimes)$ is trivially true. We now prove the other direction. Consider a function $f : \Sigma^* \mapsto \mathbb{D}$ over $\mathbb{R}(\mathbb{D}, \otimes)$ defined by the SSTT U. In effect, U outputs trees over the grammar $G(\otimes)$. We show that there exists an SSTT U' that outputs trees over the grammar $G(\otimes c)$ and computes f. As shown in [4] the class of string-to-tree transductions computable by an SSTT coincides with the class of transductions computed by MSO transducers [17]. Thus, for a given SSTT U, there is an MSO string-to-tree transducer M that computes f.

The *yield* of a tree t is the string obtained by concatenating all the leaves of t as they appear in an pre-order search. In Lemma 7.6 of [17], the authors prove that computing the yield of a tree is an MSO-definable transduction. Consider the MSO tree-to-string transducer M' that computes the yield of trees over the grammar $G(\otimes)$. Essentially, the yield of a tree t generated by the SSTT U contains constant symbols. If $[\![\otimes]\!]$ is a commutative and associative operator, the order in which the symbols appear is not important, and any tree t' with internal nodes $\otimes$ and leaves corresponding to the yield of t represents an equivalent expression to the one represented by t.

M' outputs strings over C where C is the set of constants in the output alphabet of M. Given a string $c_1 \dots c_n$ over C we want to produce the string $c_1 \otimes (c_2 \otimes (\dots (c_n \otimes (0))))$ that belongs to the grammar $G(\otimes c)$. This transduction is clearly MSO-definable (it is a simple relabeling). Let M'' be the MSO transducer that computes this relabeling.

As MSO transducers are closed under composition [17] the transduction $M_f = M \circ M' \circ M''$ is also MSO-definable. Observe that for any string $w \in \Sigma^*$, $M_f(w)$ is a term equivalent to the function $f(w)$, but is a term over the cost grammar $G(\otimes c)$ (due to associativity and commutativity of $[\![\otimes]\!]$). Clearly, M_f computes the function f. As MSO-definable transductions are equivalent to SSTT-definable transductions, there is an SSTT U' equivalent to M_f. Thus for every f in $\mathbb{R}(\mathbb{D}, \otimes)$ (defined by the SSTT U) there is an SSTT U' that defines an equivalent function in $\mathbb{R}(\mathbb{D}, \otimes c)$. $\square$

Lemma 4 *For cost domain $\mathbb{D}$ with a commutative associative operation $\otimes$, $\mathbb{F}(\mathbb{D}, \otimes c) \subseteq \mathbb{R}(\mathbb{D}, \otimes c)$.*

Proof. Consider a cost function $f : \Sigma^* \mapsto \mathbb{D}$ belonging to $\mathbb{F}(\mathbb{D}, \otimes c)$. We show that we can construct an SSTT U such that for all $w \in \Sigma^*$, $[\![U]\!](w) = f(w)$.

Recall that a cost function in $\mathbb{F}(\mathbb{D}, \otimes c)$ is definable by a CRA M over $(\mathbb{D}, \otimes c)$, where M is given by the tuple $M = (\Sigma, Q, q_0, X, \delta, \rho, \mu)$. In order to construct the desired SSTT U, we first construct an SSTT with regular-look-ahead, denoted by (U', A) that computes f. Here U' is an SSTT defined by the tuple $(Q, q_0, \{v\}, \delta', \rho', \mu')$ and A is a DFA (R, r_0, δ_R) specifying the regular-look-ahead. Given an input string w, recall that U' reads R-labeled words corresponding to the run of A on the reverse string w^r.

The final cost function μ of the CRA M maps a state to a term in $E(G, X)$. This can be extended to the partial function $\mu^* : Q \times \Sigma^* \mapsto E(G, X)$ as follows. For all q, $\mu^*(q, \varepsilon) = \mu(q)$, and $\mu^*(q, aw)$ is obtained by replacing each x in $\mu^*(\delta(q, a), w)$ by the expression $\rho(q, a, x)$. $\mu^*(q, w)$ gives the output of M starting in state q after reading w. For the grammar $G(\otimes c)$, it is easy to show by induction that for all q and w, the expression $\mu^*(q, w)$ contains at most one register name.

We now describe how the RLA automaton $A = (R, r_0, \delta_R)$ is constructed. Consider an input string $w = w_1 \dots w_n$, and recall that A reads the reverse string w^r. At position i, we need A to report the register name that will contribute to the final output, *i.e.*, the register that "flows" into the final output after M reads the remaining suffix $w_{j+1} \dots w_n$. To do so, each state of A is a pair of the form (a, χ) such that $a \in (\Sigma \cup \{\varepsilon\})$ and $\chi : Q \mapsto X \cup \{\varepsilon\}$ is a function mapping every state in Q to a register name or a special empty symbol. While reading a string $w_1 \dots w_n$ in the reverse order, the invariant maintained by a state (w_i, χ) of A is that if for each state $q \in Q$, if the CRA M reads the symbol w_i, then the register name that flows into the final output after reading the string $w_{i+1} \dots w_n$ is $\chi(q)$.

The initial state of A, $r_0 = (\varepsilon, \chi_0)$, where $\chi_0(q) = x$ if x is the (only) register name appearing in $\mu(q)$, and is ε if no register name appears in $\mu(q)$. Note that the first component of the state, *i.e.*, the input symbol is not used at this point as this corresponds to the case where the SSTT has reached the end of the string. We define δ_R using the register update functions of M as follows:

Suppose A is in state (w_{i+1}, χ) and it reads the symbol w_i. We define $\delta_R((w_{i+1}, \chi), w_i) = (w_i, \chi')$, where the function χ' is defined as follows:

$$\forall q \in Q \text{ s.t. } \delta(q, w_i) = q',\ \chi'(q) = \begin{cases} \varepsilon & \text{if } \chi(q') = \varepsilon \text{ or if } \rho(q', w_i, x) = c \\ y & \text{if } \rho(q', w_i, x) = y + c. \end{cases}$$

In the above definition, c is some constant in $\mathbb{D}$. We can now define how the state transition function δ' and register update function ρ' of U' are defined: $\delta'(q, (a, \chi)) = q'$ if in M, $\delta(q, a) = q'$. In state q, U' exactly knows the register $\chi(q)$ that contributes to the final output by reading the symbol (a, χ). Thus, it is enough for U' to have just one register (denoted v). For an expression t in $E(G, X)$, let $t[x \mapsto v]$ be the expression obtained by renaming the register x to v. The register update function $\rho'(q, (a, \chi), v)$ is defined to be $\rho(q, a, \chi(q))[v \mapsto x]$ if $\chi(q) \neq \varepsilon$, and 0 otherwise.

Finally, as SSTT are closed under regular-look-ahead, there exists an SSTT U equivalent to the SSTT with regular-look-ahead U'. Thus for every function definable by CRA over $(\mathbb{D}, \otimes c)$, there exists an SSTT U that computes f, which means that f is in $\mathbb{R}(\mathbb{D}, \otimes c)$. $\square$

Lemma 5 *For cost domain $\mathbb{D}$ with a commutative associative operation $\otimes$, $\mathbb{F}^c(\mathbb{D}, \otimes) \subseteq \mathbb{F}(\mathbb{D}, \otimes c)$.*

Proof. Consider a function $f : \Sigma^* \mapsto \mathbb{D}$ belonging to $\mathbb{F}^c(\mathbb{D}, \otimes)$. Let $M = (\Sigma, Q, q_0, X, \delta, \rho, \mu)$ be a copyless CRA over $(\mathbb{D}, \otimes)$ that computes f. We construct an CRA $M' = (\Sigma, Q, q_0, 2^X, \delta, \rho', \mu')$ over $(\mathbb{D}, \otimes c)$ that also computes f.

For every subset $S \subseteq X$, M' maintains a register denoted by x_S. M' maintains the following invariant: If the configuration of M is (q, ν), then the corresponding configuration of M' is (q, ν') such that for all $x_S \in 2^X$, $\nu'(x_S) = \bigotimes_{x \in S} \nu(x)$. Informally, each register x_S maintains the sum of the registers in the set S.

The update function ρ' of M' corresponding to the updates of M can be defined as follows. Let $u_S = \bigotimes_{x \in S} \rho(q, a, x)$. Note that the expression u_S is composed of two parts: an expression denoting the sum of register names, and a constant obtained by summing all the constants in each of the $\rho(q, a, x)$ expressions. Let $\text{REG}(u_S)$ denote the set of registers appearing in u_S and let $\text{CO}(u_S)$ denote the computed constant. Then, we define $\rho'(q, a, x_S) = x_{\text{REG}(u_S)} + \text{CO}(u_S)$. Note that as M is copyless, for every S, any register $x \in X$ appears in the expression u_S at most once. Also note that ρ' may not be copyless as the registers x_R denoting the same subset R may appear in two or more expressions $\rho'(q, a, x_S)$. The output function can be define in a similar fashion by defining the expression u_S to be $\bigotimes_{x \in S} \mu(q)$, and setting $\mu'(q) = x_{\text{REG}(u_S)} + \text{CO}(u_S)$. $\square$

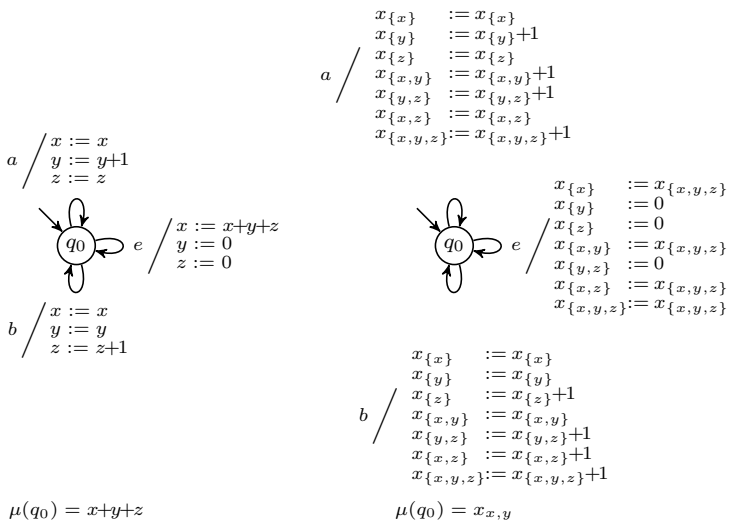
See Fig. 3 for an example of this construction. The CRA on the right of the figure will have one register for every possible subset of registers of the CRA on the left of the figure. Let's consider the transition on the symbol b. In the original CRA the update performed on x is $x := x + y + z$ and all the other registers are reset to 0. This means that for all the S containing x, x_S is updated to $x_{\{x,y,z\}}$ while all the other registers are reset.

Lemma 6 *For a cost domain $\mathbb{D}$ with a commutative and associative operation $[\![\otimes]\!]$, $\mathbb{R}(\mathbb{D}, \otimes c) \subseteq \mathbb{F}^c(\mathbb{D}, \otimes)$.*

Proof. Consider a function $f : \Sigma^* \mapsto \mathbb{D}$ belonging to $\mathbb{R}(\mathbb{D}, \otimes c)$, *i.e.*, there is an SSTT $U = (Q, q_0, X, \delta, \rho, \mu)$ over the cost grammar $G(\otimes c)$ such that $[\![U]\!] = f$, for the cost model $(\mathbb{D}, \otimes c, [\![.]\!])$. We show how we can construct a copyless CRA $M = (\Sigma, Q, q_0, X, \delta, \rho', \mu')$ over $(\mathbb{D}, \otimes)$ that also computes f.

The CRA faithfully mimics the computation of the SSTT U in its state. The only difference is the register update function and the final output function. The translation ensures that M maintains the invariant that if a configuration of U is (q, ν), the corresponding configuration for M is (q, ν') such that for all $x \in X$, $[\![\nu(x[0])]\!] = \nu'(x)$.

A register update expression $\rho(q, a, x)$ for U has one of the following forms: $x := \otimes(x, c)$, $x := c$, $\{x := x[y], y :=?\}$. Except for the last assignment, each RHS expression is in $E(G(\otimes c), X)$, and with $[\![\otimes]\!] = +$, can be can be directly mimicked by M by setting $\rho'(q, a, x)$ to be the expressions $x := x + c$ and $x := c$ respectively. To simulate $\{x := x[y], y :=?\}$, we note that as $\otimes$ is associative and commutative, for a term in $E(G(\otimes c), X)$, the term $x[y]$ is equivalent to the term $\otimes(x, y)$. Thus, with $[\![\otimes]\!] = +$, M can simulate parameter substitution by the *copyless* assignment $\{x := x \otimes y, y := 0\}$. The output function $\mu'(q)$ can be mimicked in a similar fashion for the corresponding expressions in $\mu(q)$. $\square$



(a) CRA (+) (b) Corresponding CRA (+c)

Figure 3: Translation from copyless CRA (+) to CRA (+c)

Theorem 7 (Expressiveness of Additive Cost Functions) *For cost domain $\mathbb{D}$ with a commutative associative operation $\otimes$, $\mathbb{F}(\mathbb{D}, \otimes c) = \mathbb{F}^c(\mathbb{D}, \otimes) = \mathbb{R}(\mathbb{D}, \otimes c) = \mathbb{R}(\mathbb{D}, \otimes)$.*

Proof. Follows from Lemma 3,4,5 and 6. □

We can also establish the following results establishing an expressiveness hierarchy between different classes. First we show that the copyless restriction for CRA over $(\mathbb{D}, \otimes c)$ reduces the expressivity.

Theorem 8 $\mathbb{F}(\mathbb{D}, \otimes c) \not\subseteq \mathbb{F}^c(\mathbb{D}, \otimes c)$.

Proof. Consider the function f_1 computed by the CRA M_1 in Fig. 1 that maps a string w to the length of the substring obtained by deleting all b's after the last occurrence of e in w. We show that for any fixed k, there does not exist a copyless CRA capable of computing this function.

Assume that M is a copyless CRA over $(\mathbb{D}, \otimes c)$ that can compute this function with k registers. Without loss of generality we can assume that in every assignment of the form $x := y + c$, y and x are the same register (if the original machine is doing some copyless renaming, we can remember the renaming in the state).

Now consider a string of the form $w = b^n a e b^n a e \ldots$, where n is greater than the number of states in M. Thus, for each i, while processing the i^{th} block of b^n, some state q_i (possibly depending on the block number i) must be visited at least twice. For each i, let x_i be the register used to calculate the output after reading the input string $(b^n a e)^i a$. Now observe that for all $i < j$, x_i and x_j are necessarily distinct: if we pump w to $w' = (b^n a e)^{i-1} b^{n+l} a e (b^n a e)^{j-i} \ldots$, the value of x_i after i blocks must be unchanged, but the value of x_j must change. We have thus established that a different register must be used to produce the output after each block, but this is not possible if the machine has only a finite number (k) of registers. □

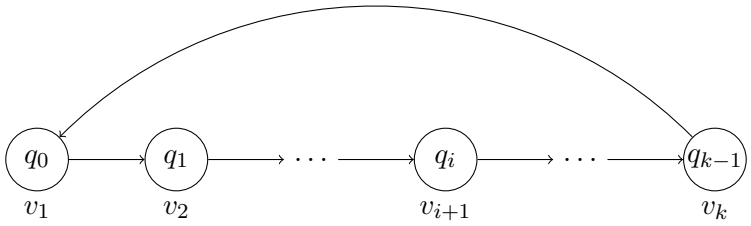


Figure 4: CRA M_k implementing f_k. Each transition performs $v_i := v_i + i$, for all i.

We then show that removing the copyless restriction from CRAs over $(\mathbb{D}, \otimes)$ is too permissive as it allows computing cost functions that grow exponentially.

Theorem 9 *There exists a CRA M over $(\mathbb{D}, \otimes)$ that cannot be expressed as a copyless CRA M over $(\mathbb{D}, \otimes)$,* i.e.*, $\mathbb{F}(\mathbb{D}, \otimes) \nsubseteq \mathbb{F}^c(\mathbb{D}, \otimes)$.*

Proof. We prove this result by contradiction. We create a CRA M over $(\mathbb{D}, \otimes)$ over the alphabet $\{a\}$ such that on the first a it perform the update $x := 2$ and on the subsequent a's it performs the update $x := x + x$. Given a string $w \in a^+$, the function computed is $f = 2^{|w|}$. Let's assume that there exists a copyless CRA M' that can compute f. By Theorem 7, there must be an SSTT U in $\mathbb{R}(\mathbb{D}, \otimes c)$ that also computes f. However, as the output is not linearly bounded, this function cannot be computed by an SSTT and so we reach a contradiction. □

Finally we show that having multiple registers is essential for expressive completeness.

Theorem 10 *For every $k \in \mathbb{N}$, there is a cost function f so that every CRA M over $\mathbb{F}(\mathbb{N}, +c)$ has at least k registers.*

Proof. For each k, consider the function $f_k : \mathbb{N} \to \mathbb{N}$ defined as $f_k(x) = ((x \bmod k) + 1) \cdot x$. The input x is expressed in a unary alphabet $\Sigma = \{1\}$. This function outputs one of x, $2x$, $3x$, …, kx, depending on the length of x.

First, these functions can be implemented by a CRA M_k (shown in Fig. 4). M_k has k registers v_1, …, v_k, all initialized to 0 and has k states q_0, …, q_{k-1}. For each i, $\delta(q_i, 1) = q_{(i+1) \bmod k}$, $\rho(q_i, 1, v_i) = v_i + i$, and $\mu(q_i) = v_{(i+1) \bmod k}$.

We now show that at least k registers are necessary. Consider otherwise, and say we are able to produce such an CRA M with $k-1$ registers. The main idea is that the difference between any two "sub"-functions ax and bx, $a \neq b$, grows without bound. Since M works over a unary input alphabet, the only form it can assume is that of a lasso (Fig. 5). Say there are $m \geq 0$ states in the initial approach to the loop, and $n \geq 1$ states in a single pass of the loop.

In the following argument, let c denote the largest constant that appears in the description of M. Without loss of generality, we can assume that no register renaming occurs, as for a CRA with register renaming, there is an equivalent CRA with no register renaming, by tracking register renaming as part of its state. Thus, if in state q, if $\rho(q, 1, v_i)$ is the expression $v_j + c_1$ (where $i \neq j$), then there is no v_ℓ such that v_i appears in $\rho(q, 1, v_\ell)$.

Also observe that no register that ever gets reset during the loop can contribute to the output afterwards. Say there is some transition during which v_i is updated as $v_i := c_3$. If this register influences the output $n' \geq 0$ states later, then for $x > cn' + c_3$, the output must be incorrect.

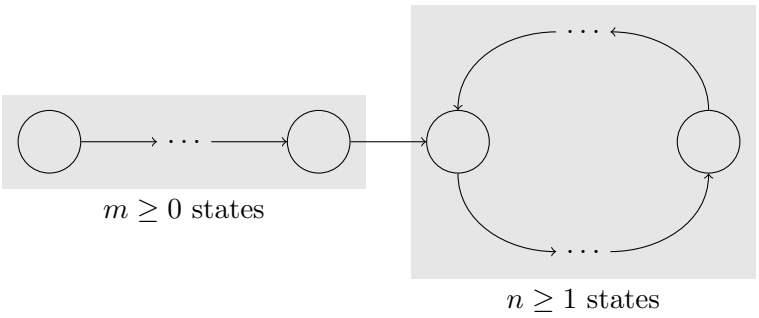


Figure 5: The structure of a possible $k-1$-register CRA M implementing f_k.

Now pick some state q occurring in the loop. Say that the output in q depends on some register v_i. Let us call q good if register v_i in q influences the output in some state q' which is $p < k$ transitions from q. At least one good state has to exist, since the machine has at most $k-1$ registers. We now use the presence of this good state to derive a contradiction: the outputs in q and q' can differ by no more than $(p+2).c$. But since q and q' are closer than k steps apart, they output necessarily different functions, and hence for $x > (p+2).c$, the outputs in these two states are required to differ by more than this amount. The contradiction is complete. □

We now prove that CRAs over commutative monoid $(\mathbb{D}, \otimes)$ are closed under the $\otimes$ operation.

Theorem 11 (Addition) *Given two CRAs M_1 and M_2 over $(\mathbb{D}, \otimes)$, there exists a M over $(\mathbb{D}, \otimes)$ such that $\forall w \in \Sigma^*$. $M(w) = M_1(w) \otimes M_2(w)$.*

Proof. Given $M_i = (\Sigma, Q^i, q_0^i, X^i, \delta^i, \rho^i, \mu^i)$ (where $i \in \{1,2\}$) we construct $M = (\Sigma, Q^1 \times Q^2, (q_0^1, q_0^2), X^1 \times X^2 \cup X^1 \cup X^2, \delta, \rho, \mu)$. We assume X_1 and X_2 are disjoints sets. The registers in M are pairs of the form $(x,y) \in X^1 \times X^2$ or singletons of the form $x \in X^1 \cup X^2$.

After processing an input w, M will be in the configuration $((q_1, q_2), \nu)$ where

- $q_i = \delta^*(q_0^i, w)$,
- $\nu((x,y)) = \nu_1(x) \otimes \nu_2(y)$,
- $\nu(x) = \nu_1(x)$ if $x \in X^1$, and
- $\nu(y) = \nu_2(y)$ if $y \in X^2$.

We now define the functions δ and ρ. Given a state $(q_1, q_2) \in Q_1 \times Q_2$, a symbol $a \in \Sigma$, and a variable $x \in X^1 \times X^2 \cup X^1 \cup X^2$,

- $\delta((q_1, q_2), a) = (\delta^1(q_1, a), \delta^2(q_2, a))$, and
- if $x = (x_1, x_2) \in X^1 \times X^2$, and $t = \rho^1(q_1, a, x_1) \otimes \rho^2(q_1, a, x_2)$ then $\rho((q_1, q_2), a, (x_1, x_2)) = \textsc{reg}(t) \otimes \textsc{co}(t)$.
- if $x \in X^i$, and $t = \rho^i(q_1, a, x)$ then $\rho((q_1, q_2), a, x) = \textsc{reg}(t) \otimes \textsc{co}(t)$.

where, given an expression e, $\text{REG}(e)$ is equal to x iff x is the only variable appearing in e, to (x, y) when $x \in X^1, y \in X^2$ are the only two variable appearing in e and 0 when no variable appears in e, while $\text{CO}(e)$ is the sum of all the constants appearing in e. Note that, at every point, at most 2 registers can appear in the combined right-hand side. The output function can be defined in a similar way. Given a state $(q_1, q_2) \in Q_1 \times Q_2$, if $t = \mu^1(q_1, a, x_1) \otimes \mu^2(q_1, a, x_2)$, then $\mu((q_1, q_2)) = \text{REG}(t) \otimes \text{CO}(t)$. By construction M computes the correct function. □

4.2 Weighted Automata

A weighted automaton [14] over an input alphabet Σ and a cost domain $\mathbb{D}$ is a *nondeterministic* finite-state automaton whose edges are labeled with input symbols in Σ and costs in $\mathbb{D}$. For an input string w, the automaton can have multiple accepting paths from its initial state to an accepting state. The semantics of the automaton is defined using two binary functions $\oplus$ and $\otimes$ such that $\oplus$ is associative and commutative, and $\otimes$ distributes over $\oplus$ (to be precise, form a *semiring* algebraic structure). The cost of a path is the sum of the costs of all the transitions along the path according to $\otimes$, and the cost of a string w is obtained by applying $\oplus$ to the set of costs of all accepting paths of the automaton over w.

Let K be the semiring $(\mathbb{D}, \oplus, \otimes, \bar{0}, \bar{1})$. Formally, a weighted automaton WA with weights from K, from an input alphabet Σ into the domain $\mathbb{D}$ is a tuple $W = (\Sigma, P, I, F, E, \lambda, \rho)$ where Σ is a finite input alphabet, P is a finite set of states, $I \subseteq P$ the set of initial states, $F \subseteq P$ the set of final states, E a finite multiset of transitions, which are elements of $P \times \Sigma \times \mathbb{D} \times P$, $\lambda : I \mapsto \mathbb{D}$ an initial weight function, and $\rho : F \mapsto \mathbb{D}$ a final weight function mapping F to $\mathbb{D}$.

Consider a string $w = w_1 \dots w_n \in \Sigma^*$ and a WA W. A sequence $\pi = (q_0, c_0), (q_1, c_1), \dots, (q_n, c_n)$ is an accepting in sequence for w if, for every $1 \leq i \leq n$, $(q_{i-1}, w_i, c_i, q_i) \in E$ and $\lambda(q_0) = c_0$. The weight of π (denoted as $w(\pi)$) is computed as $(\otimes_{0 \leq i \leq n} c_i) \otimes \rho(q_n)$. Given a word w we denote by $P(w)$ the set of all the accepting paths of w. The weight $T(w)$ of the string w is defined as:

$$T(w) = \bigoplus_{\pi \in P(s)} w[\pi]$$

where $\oplus$ is also an operation over S. A weighted automaton is called *single-valued* if each input string has at most one accepting path[1]. To interpret a single-valued weighted automaton, we need only an interpretation for $\otimes$. Thus, we can compare the class of functions definable by such automata with regular additive functions.

Theorem 12 (Single-valued Weighted Automata) *A cost function $f : \Sigma^* \mapsto \mathbb{D}$ is in $\mathbb{R}(\mathbb{D}, \otimes c)$ iff it is definable by a single valued weighted automaton.*

Proof. Let $W = (\Sigma, P, I, F, E, \lambda, \rho)$ a *single valued* weighted automaton that computes the function $f : \Sigma^* \mapsto \mathbb{D}$. We construct an SSTT with *regular look-ahead* $(U, A) = (P \cup \{I\}, I, \{v\}, \delta, \rho, \mu)$, (R, r_0, δ_R) that computes f. The SSTT U uses the cost grammar $G(\oplus c)$ to construct its terms. We then use Theorem 2 to show that there exists an SSTT that computes f, which means that f is regular.

Even though W is nondeterministic, since it is *single valued*, it will have only one accepting path. We construct $A = (R, r_0, \delta_R)$ such that the states in R give information on what is the

[1] In some of the literature these are called *unambiguous* weighted automata, while a *single-valued* weighted automaton is one where the weights of all the accepting paths are the same. The two notions are proved to be equivalent.

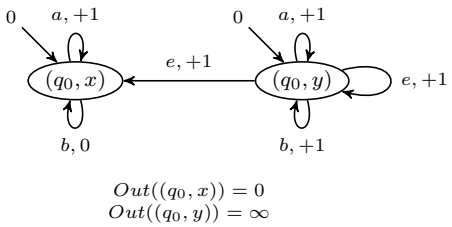


Figure 6: Weighted Automaton corresponding to M_1 in Figure 1

next transition to take to reach an accepting path. Every state $r \in R$ is a pair (a, f) where $a \in (\Sigma \cup \{\varepsilon\})$ and f is a partial function from P to P. After reading the i^{th} symbol of the input word $w = w_1 \dots w_n$, $r = (w_i, f)$ and $f(q) = q'$ if: 1) $(q, a, c, q') \in E$ for some c, and 2) if W starts reading $w_{i+1} \dots w_n$ in q', it will reach an accepting state.

The initial state r_0 is defined as (ε, f_0). and for every $q \in F$ $f_0(q) = q$. The initial state does not encode any information as it corresponds to the case where the U has reached the end of the string w. We now define δ_R. Suppose A is in state (a, f) and it is reading the input b. The new state will be (b, f') where $f'(q) = q'$ if $f(q')$ is defined and $(q, b, c, q') \in E$ for some c.

We now define the state transition function for the SSTT U. We define $\delta(q, (a, f)) = q'$ if $f(q) = q'$. Particular attention must be made for the case when U is in state I. In this case on input symbol (a, f), $\delta(I, (a, f)) = q'$ where q' is the only state in I such that $f(q) = q'$. Notice that there can be only one state of this form otherwise W would not be single valued. For every transition of the form $(q, a, c, f(q))$ in W, in U, the register update function $\rho(q, a, v)$ maps v to the expression $\otimes(v, c)$. This shows that for every weighted automaton W, we can construct an SSTT U over the cost grammar $G(\oplus c)$ such that for all input strings w, $[\![U]\!](w) = W(w)$.

We now prove the other direction. By Theorem 7 we know that every function f in $\mathbb{R}(\mathbb{D}, \otimes c)$ can be computed by a CRA M over $(\mathbb{D}, \otimes c)$. Let $M = (\Sigma, Q, q_0, X, \delta, \rho, \mu)$. We show how we can construct a *single valued* weighted automaton $W = (\Sigma, Q \times (X \cup \{r\}), \{q_0\} \times (X \cup \{r\}), Q \times X, E, \lambda, \rho)$ that also computes f.

After processing an input word w, if M has the configuration (q, ν), we have that: corresponding to every $x \in X$ there exists a path in W from the initial state such that the cost along that path is equal to $\nu(x)$. Let's now give the definition of E: (1) if $\delta(q, a) = q'$ and $\rho(q, a, x) = \otimes(y, c)$, then $((q, y), a, c, (q', x)) \in E$, (2) if $\delta(q, a) = q'$ and $\rho(q, a, x) = c$, then $((q, r), a, c, (q', x)) \in E$, and (3) if $\delta(q, a) = q'$, $((q, r), a, 0, (q', r)) \in E$, The nodes (q, r) are always reachable with cost 0 and are used to represent resets, but none of them is accepting. ρ can be defined in a similar way.

A simple inductive proof establishes that in a CRA over $(\mathbb{D}, \otimes c)$, in any state, only one register eventually contributes to the final output, or in other words, only one value flows to the final output. Thus, the constructed weighted automaton is is single-valued. □

An example of the translation of the function M_1 of Figure 1 is in Figure 6. The machine has only two states (q_0, x), and (q_0, y). If we take for example the transition of the first automaton when reading c, we can see that x is updated to $y+1$. In the automaton of Figure 6 this is reflected by the transition from (q_0, y) to (q_0, x) with label c and with weight $+1$.

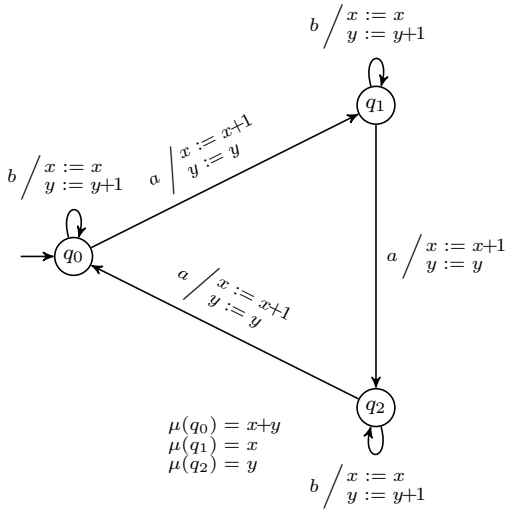


Figure 7: Example CRA over $(\otimes)$ needing less registers than CRA over $(\otimes c)$

4.3 Decision Problems

Minimum Costs. The shortest path problem for CRAs is to find a string w whose cost is the minimum. For numerical domain with addition, for CRAs with increment, we can solve the shortest path problem by reducing it to classical shortest paths using the translation from CRAs with increment to single-valued weighted automata used in the proof of Theorem 12. If the CRA has n states and k registers, the graph has $n \cdot k$ vertices. The exact complexity depends on the weights used: for example, if the costs are nonnegative, we can use Dijkstra's algorithm.

Theorem 13 (Shortest Path for CRAs with Inc) *Given a CRA M over the cost model $(\mathbb{Q}, +c)$, computing $\min\{M(w) \mid w \in \Sigma^*\}$ is solvable in* PTIME.

Proof. We reduce the problem to finding the shortest path in a weighted graph. Using the construction of Theorem 12 we create a weighted graph. The graph has nk nodes and $|\Sigma|nk$ edges where n, k are the number of states and variable of M respectively. If the weights are all positive we can use Dijkstra's algorithm with a final complexity of $O(|\Sigma|nk + nk\log(nk))$ otherwise we can use the Bellman-Ford algorithm, making the complexity $O(n^2k^2\log(nk))$. □

Even though $\mathbb{F}(\mathbb{D}, \otimes c) = \mathbb{F}^c(\mathbb{D}, \otimes)$, the model with addition can be more succinct (see Fig. 7 for an example). To solve minimum-cost problem for copyless-CRAs over the cost model $(\mathbb{D}, \otimes)$, we can use the translation to CRAs over $(\mathbb{D}, \otimes c)$ used in the proof of Theorem 7, which causes a blow-up exponential in the number of registers. We can establish an NP-hardness bound for the min-cost problem by a simple reduction from 3-SAT.

Theorem 14 (Shortest Paths for CRAs with Addition) *Given a copyless-CRA M over the cost model $(\mathbb{Q}, +)$ with n states and k registers, computing $\min\{M(w) \mid w \in \Sigma^*\}$ is solvable in time polynomial in n and exponential in k. Given a copyless CRA M over the cost model $(\mathbb{N}, +)$ and a constant $K \in \mathbb{N}$, deciding whether there exists a string w such that $M(w) \leq K$ is* NP-HARD.

Proof. The first result follows from the complexity of the translation in Lemma 5. For the second part we give a reduction from 3-SAT. Given an instance $V = \{v_1, \ldots, v_n\}, C = \{c_1, \ldots, c_k\}$ where V is the set of literals and C the set of clauses we construct a CRA M over $(\mathbb{N}, +)$. M is defined as the tuple $(\Sigma, Q, q_0, X, \delta, \rho, \mu)$, where $\Sigma = \{0, 1\}$, $Q = \{q_0, \ldots, q_{n+2}\}$, $X = \{x_1, \ldots, x_k\}$. The update functions are defined as follows: For each i, and $b \in \Sigma$, $\delta(q_i, b) = q_{i+1}$, and $\rho(q_i, 0, x_j) = 0$ if the clause c_j becomes *true* when the variable v_i is *false*. Similarly $\rho(q_i, 1, x_j) = 0$ if the clause c_j becomes *true* when v_i is *true*. Finally, we define $\rho(q_{n+1}, b, x_1) = \sum_{i=1}^{k} x_i$, and $\mu(q_{n+2}) = x_1$. It is easy to see that every path in the M corresponds to a unique valuation for the literals $v_1, \ldots, v_n$. Finally, if the minimum-cost computed by M is 0, then we have an instance of SAT, as there is a valuation of the literals that makes every clause *true*. If the minimum-cost computed is greater than 0, then the conjunction of the clauses is unsatisfiable. This means that solving min-value problem for a CRA over $(\mathbb{N}, +)$ is as hard as solving 3-SAT. □

Equivalence and Containment. Given two cost register automata using addition over a numerical domain, checking whether they define exactly the same function is solvable in polynomial time relying on properties of systems of linear equations.

Theorem 15 (Equivalence of CRAs with Addition) *Given two CRAs M_1 and M_2 over the cost model $(\mathbb{Q}, +)$, deciding whether for all w, $M_1(w) = M_2(w)$ is solvable in* PTIME.

Proof. Given $M_i = (\Sigma, Q^i, q_0^i, X^i, \delta^i, \rho^i, \mu^i)$ (where $i \in \{1, 2\}$) we construct $M = (\Sigma, Q^1 \times Q^2, (q_0^1, q_0^2), X^1 \cup X^2, \delta, \rho, \mu)$. We assume the two sets variables X_1 and X_2 are disjoint.

For every $(q_1, q_2) \in Q^1 \times Q^2$ and $a \in \Sigma$, $\delta((q_q, 1_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$. M updates the two constituent sets of registers separately. For $x_1 \in X_1$, $\rho((q_1, q_2), a, x_1) = \rho_1(q_1, a, x_1)$, and for $x_2 \in X_2$ $\rho((q_1, q_2), a, x_1) = \rho_2(q_2, a, x_2)$.

We want to check if along every path of the M, and for every state (q_1, q_2), the equation $\mu_1(q_1) = \mu_2(q_2)$ holds. We adapt the algorithm for checking validity of affine relations over affine programs presented in [27] to do this. The algorithm in [27], checks the validity of affine relations (equality constraints over linear combinations of real-valued or rational-valued program variables and constants), over affine graphs (graphs where each edge is labeled by an affine assignment). We can cast the equivalence check for CRAs over $(\mathbb{Q}, \otimes)$ as a subcase of this problem.

The algorithm propagates the equation $(\mu_1(q_1) = \mu_2(q_2)$ backward along each transition using the register update function: for an edge from q to q' with some label a, every equation $e_1 = e_2$ that must hold at v yields an equation $e_1' = e_2'$ that must hold at u, where the expressions e_1' and e_2' are obtained from e_1 and e_2 using substitution to account for the update of registers along the edge from u to v. e_i' will be equal to e_i where every register $x \in X$ is replaced by $\rho(u, a, x)$. At every step of the back propagation, we compute the basis of the set of equations in every state using Gaussian elimination. If we reach a system of equations with no solution the two machines are inequivalent, while if we reach a fix point where no independent equations can be added, the two machines are equivalent.

As shown in Theorem 2 of [27], such a propagation terminates in $O(nk^3)$ where n is the size of the machine (in our case $|Q^1||Q^2||\Sigma|$) and k is the number of registers (in our case $|X^1| + |X^2|$).

For CRAs that use only increment, the cubic complexity of the Gaussian elimination in the inner loop of the equivalence check can be simplified to quadratic: at every step in the back propagation,

all equations are of the form $x = y + c$. The final complexity is $O(nk^2)$ if we only have increments and $O(nk^3)$ otherwise (k, n are as defined before). □

We now show that general containment is also decidable in polynomial time and that checking if a number is in the range of a CRA over $(\mathbb{Z}, +)$ is decidable in polynomial time.

Theorem 16 ($M_1 \leq M_2$) *Given two CRAs M_1 and M_2 over the cost model $(\mathbb{Q}, +c)$, deciding whether $\forall w \in \Sigma^*.M_1(w) \leq M_2(w)$ is in* PTIME.

Proof. We reduce the problem to shortest path. If $\forall w \in \Sigma^*, M_1(w) \leq M_2(w)$ then $\forall w \in \Sigma^*, M_1(w) - M_2(w) \leq 0$. By changing the sign of all the edges in M_1 we can construct $M_1' = -M_1$. Using Theorem 11 we can construct a CRA M which is equivalent to $M_2(w) + M_1'(w)$ and has size polynomial in M_1 and M_2. If there exists a w such that $M(w) < 0$ then $M_1 \not\leq M_2$. We can find if such $w \in \Sigma^*$ exists using the shortest path algorithm of Theorem 13. □

Theorem 17 ($k \in$ Range) *Given two CRAs M over the cost model $(\mathbb{Z}, +c)$ and a constant $k \in \mathbb{Z}$,, deciding whether $\exists w \in \Sigma^*.M(w) = k$ is in* NLOGSPACE.

Proof. We reduce the problem to 0 reachability in a weighted graph. Given M we construct M' such that $M'(w) = M(w) - k$ for every $w \in \Sigma^*$. M' is same as M except for the output function. For every state q of M, $\mu'(q) = \mu(q) - k$ iff $\mu(q)$ is defined. Now we want to check $\exists w \in \Sigma^*.M'(w) = 0$. We can create the same graph of shortest path and look for a path of cost 0. The problem of 0-reachability over finite graphs is known to be in NLOGSPACE. □

5 Semiring Cost Models

In this section, we consider the cost models which result when the cost model supports two binary operations, $\oplus$ and $\otimes$, that impose a semiring structure (see subsection 4.2 for the definition of semiring). This structure has been studied extensively in the literature on weighted automata and rational power series. A specific case of interest is the *Min-Plus semiring*, where the cost domain is $\mathbb{N} \cup \{\infty\}$, $\oplus$ is the min operation, and $\otimes$ is arithmetic addition. While choosing a grammar, we can restrict either or both of $\oplus$ and $\otimes$ to be "unary" (that is, the second argument is a constant). To study the Min-Plus semiring, it makes sense to choose min to be binary, while addition to be unary. Hence, in this section, we will focus on the grammar $G(\oplus, \otimes c)$, and the class $\mathbb{R}(\mathbb{D}, \oplus, \otimes c)$ of cost functions.

5.1 CRA Models

Our first task is to find a suitable set of operations for cost register automata so as to have expressiveness same as the class $\mathbb{R}(\mathbb{D}, \oplus, \otimes c)$. It turns out that (unrestricted) CRAs with $\oplus$ and $\otimes c$ are too expressive, while their copyless counterparts are too restrictive. We need to enforce the copyless restriction, but allow substitution. In the proposed model, each register x has two fields ranging over values from $\mathbb{D}$: $(x.c, x.d)$. The intuitive understanding is that x represents the expression $(x.d \otimes ?) \oplus x.c$ where ? denotes the parameter. Such a pair can be viewed as the "most evaluated" form of a parameterized term in the corresponding SSTT. Expressions used for the update are given by the grammar

$$e ::= (c, d) \mid x \mid e_1 \underline{\oplus}\, e_2 \mid e_1 \underline{\otimes}\, c \mid e_1\,[e_2]$$

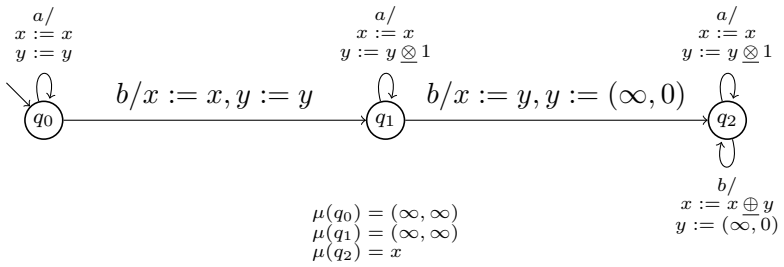


Figure 8: The $\mathbb{F}^c(\mathbb{D} \times \mathbb{D}, \underline{\oplus}, \underline{\otimes}, [\cdot])$ machine for example 18.

where x is a register, and c and d are constants. For the min-inc interpretation, the initial values are of the form $(\infty, 0)$ corresponding to the additive and multiplicative identities. We require that registers are used in a copyless manner, so that any particular register x appears in the update of at most one register. The semantics of the operators on pairs is defined below: $e_1 \underline{\oplus} e_2$ is defined to be $(e_1.c \oplus e_2.c, e_1.d)$; $e_1 \underline{\otimes} d$ equals $(e_1.c \otimes d, e_1.d \otimes d)$; and $e_1[e_2]$ is given by $(e_1.c \oplus e_1.d \otimes e_2.c, e_1.d \otimes e_2.d)$. While registers contain and expressions evaluate to pairs, the output function projects out the "c" component of this pair: this is equivalent to instantiating the parameter ? to 0, the additive identity, since over semirings, the additive identity annihilates any other element under multiplication ($x.d \otimes 0 \oplus x.c = 0 \oplus x.c = x.c$). The resulting model of CRA-definable cost functions is $\mathbb{F}^c(\mathbb{D} \times \mathbb{D}, \underline{\oplus}, \underline{\otimes} c, [\cdot])$

Example 18 *Consider strings $w \in \{a, b\}^*$, so that $f(w)$ is the number of a's between the closest pair of b's. This function is in $\mathbb{F}(\mathbb{D}, \min, +c)$, but not in the more restricted classes: $\mathbb{F}(\mathbb{D}, \min(\cdot, d), +c)$ and $\mathbb{F}(\mathbb{D}, +c)$. In figure 8, we show a $\mathbb{F}^c(\mathbb{D} \times \mathbb{D}, \underline{\oplus}, \underline{\otimes}, [\cdot])$ machine that can compute f. The output in both q_0 and q_1 is identically ∞, while the output in q_2 is the "c" component of the output function x: $x.c$.*

5.2 Expressiveness

The next theorem summarizes the relationship between functions definable by different CRA models. The rest of the session contains the proof of this theorem.

Theorem 19 (Expressiveness of Semiring Cost Functions) *If $(\mathbb{D}, \oplus, \otimes)$ forms a semiring, then*

$$\mathbb{F}^c(\mathbb{D}, \oplus, \otimes c) \subset \mathbb{F}^c(\mathbb{D} \times \mathbb{D}, \underline{\oplus}, \underline{\otimes} c, [\cdot]) = \mathbb{R}(\mathbb{D}, \oplus, \otimes d) \subset \mathbb{F}(\mathbb{D}, \oplus, \otimes c)$$

We split the proof into the following lemmas.

Lemma 20 *If $(\mathbb{D}, \oplus, \otimes)$ forms a semiring, then*

$$\mathbb{F}^c(\mathbb{D}, \oplus, \otimes c) \subset \mathbb{F}^c(\mathbb{D} \times \mathbb{D}, \underline{\oplus}, \underline{\otimes} c, [\cdot])$$

Proof. We first show that the containment holds and then that it is strict. Copyless CRAs with $\oplus$ and $\otimes c$ can be simulated by copyless CRAs operating over pairs and performing $\underline{\oplus}$, $\underline{\otimes}$, and $[\cdot]$. Given a $\mathbb{F}^c(\mathbb{D}, \oplus, \otimes c)$ machine M_1, construct an $\mathbb{F}^c(\mathbb{D} \times \mathbb{D}, \underline{\oplus}, \underline{\otimes} c, [\cdot])$ machine M_2 with the same states, and same registers. Replace every occurrence of $\oplus$ and $\otimes c$ in the update expressions to $\underline{\oplus}$ and $\underline{\otimes} c$ respectively.

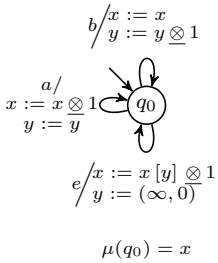


Figure 9: f_1 from figure 1 is in $\mathbb{F}^c\left(\mathbb{D} \times \mathbb{D},\, \underline{\oplus}\,,\, \underline{\otimes}\,, [\cdot]\right)$.

To show strict containment, let $(\mathbb{D}, \oplus, \otimes)$ be the Min-Plus semiring. Our witness function is f_1 from Fig. 1. First off, observe that $f_1 \in \mathbb{F}^c\left(\mathbb{D} \times \mathbb{D},\, \underline{\oplus}\,,\, \underline{\otimes}\, c, [\cdot]\right)$, as shown in figure 9. We now demonstrate that $f_1 \notin \mathbb{F}^c(\mathbb{D}, \min, +c)$, and our proof is similar to that of theorem 8.

We proceed by contradiction. Say we are given a copyless CRA machine M over $(\mathbb{D}, \min, +c)$ that implements f. Without loss of generality, we can assume that in every update, a register x is either reset, or appears in its own update expression: $x := \min(x + c, \ldots)$.

Consider a string of the form $w = b^n a e b^n a e \ldots$, where n is greater than the number of states in M. Thus, for each i, while processing the i^{th} block of b^n, some state q_i (possibly depending on the block number i) must be visited at least twice. For each i, let x_i be the register which influences the output after reading the input string $(b^n a e)^i a$. Now observe that for all $i < j$, x_i and x_j are necessarily distinct: if we pump w to $w' = (b^n a e)^{i-1} b^{n+k} a e (b^n a e)^{j-i} \ldots$, the value of x_i after i blocks must be unchanged, but the value of x_j must change. We have thus established that a different register must be used to produce the output after each block, but this is not possible if the machine has only a finite number (k in this case) of registers. □

Lemma 21 *If $(\mathbb{D}, \oplus, \otimes)$ forms a semiring, then*

$$\mathbb{F}^c\left(\mathbb{D} \times \mathbb{D},\, \underline{\oplus}\,,\, \underline{\otimes}\, c, [\cdot]\right) \;=\; \mathbb{R}(\mathbb{D}, \oplus, \otimes c).$$

Proof. From the definition of CRAs the terms constructed by the SSTTs are in correspondence with their most evaluated versions maintained by CRAs. □

Lemma 22 *If $(\mathbb{D}, \oplus, \otimes)$ forms a semiring, then*

$$\mathbb{F}^c\left(\mathbb{D} \times \mathbb{D},\, \underline{\oplus}\,,\, \underline{\otimes}\, c, [\cdot]\right) \;\subseteq\; \mathbb{F}(\mathbb{D}, \oplus, \otimes c).$$

Proof. Consider a copyless CRA machine M over $(\mathbb{D} \times \mathbb{D},\, \underline{\oplus}\,,\, \underline{\otimes}\,, [\cdot])$. Let V_r be the set of its registers. We construct a copyful CRA M' over $(\mathbb{D}, \oplus, \otimes c)$ equivalent to M. We perform the following subset construction over registers. The states and transitions of M' are the same as in M. The set of registers V_l of M' is the following:

1. $x.c$ and $x.d$ for every $x \in V_r$.
2. for every $S \subseteq V_r$, we maintain $d_S = \otimes_{x \in S} x.d$, and for all $x \notin S$, $x d_S = x.c \otimes d_S$.

The expression on the right of each of the above equalities is the intended invariant we'll maintain. Because of the properties of the semiring, we can simplify the resulting expression into a linear

form (an expression of the form $\oplus_i (X_i \otimes a_i) \oplus c$, for some constants a_i and c, where i ranges over the registers).

We define an elementary update in M as one in which: the value of no register changes, or exactly two registers permute: $\langle x, y\rangle := \langle y, x\rangle$, or exactly one register is reset: $x := (c, d)$, or exactly one register changes: $x := x \underline{\otimes} d$, or exactly two registers change (addition): $\langle x, y\rangle := \langle x \underline{\oplus} y, (0, 1)\rangle$, or exactly two registers change (substitution): $\langle x, y\rangle := \langle x\,[y]\,, (0, 1)\rangle$. Observe that any copyless register update can be written as a finite sequence of elementary updates. Also a finite sequence of updates in a CRA machine over $(\mathbb{D}, \oplus, \otimes c)$ can be summarized into a single update. Thus, if we demonstrate a semantics-preserving transformation from elementary updates to copyful linear updates, we are done.

Given a register $x \in V_l$, let x be its value before the update, and x' be its intended value after. We show that x' in each case can be written as a linear combination of the old values, thus giving a linear update rule $x := Expr$. Only the last two cases are interesting:

1. Addition: $\langle x, y\rangle := \langle x \underline{\oplus} y, (0, 1)\rangle$.

   (a) $x.c' = x.c \oplus y.c$, $x.d' = x.d$. $y.c' = 0$ and $y.d' = 1$.

   (b) For $z \neq x, y$. $z.c' = z.c$ and $z.d' = z.d$.

   (c) For S, $x, y \notin S$. $d'_S = d_S$, $xd'_S = (x.c \oplus y.c) \otimes d'_S = xd'_S \oplus yd'_S$. $yd'_S = 0$, and $zd'_S = zd_S$.

   (d) For S, $x \in S$, but $y \notin S$. $d'_S = d_S$, $yd'_S = 0$, and $zd'_S = zd_S$. (Exactly the same as the previous case.)

   (e) For S, $x \notin S$, but $y \in S$. Let $S' = S \setminus \{y\}$. $d'_S = y.d' \otimes d'_{S'} = d_{S'}$. $xd'_S = x.c' \otimes d'_S = (x.c \oplus y.c) \otimes d_{S'} = xd_{S'} \oplus yd_{S'}$. $zd'_S = z.c' \otimes d'_S = z.c \otimes d_{S'} = zd_{S'}$.

   (f) For S, $x, y \in S$. $S' = S \setminus \{x, y\}$. $d'_S = x.d' \otimes y.d' \otimes d'_{S'} = x.d \otimes d_{S'} = d_{S' \cup \{x\}}$. $zd'_S = z.c' \otimes d'_S = zd_{S' \cup \{x\}}$.

2. Substitution: $\langle x, y := x\,[y]\,, (0, 1)\rangle$. This shows why we needed to keep the subset registers.

   (a) $x.c' = x.c \oplus x.d \otimes y.c = x.c \oplus yd_{\{x\}}$. $x.d' = x.d \otimes y.d = d_{\{x,y\}}$. $y.c' = 0$ and $y.d' = 1$.

   (b) For $z \neq x, y$, $z.c' = z.c$ and $z.d' = z.d$.

   (c) For S, $x, y \notin S$. $d'_S = d_S$, $xd'_S = x.c' \otimes d'_S = (x.c \oplus x.d \otimes y.c) \otimes d_S = xd_S \oplus yd_{S \cup \{x\}}$. $yd'_S = 0$, and $zd'_S = zd_S$.

   (d) For S, $x \in S$ but $y \notin S$. Let $S' = S \setminus \{x\}$. $d'_S = x.d' \otimes d'_{S'} = d_{\{x,y\}} \otimes d_{S'} = d_{S \cup \{y\}}$. $yd'_S = 0$, and $zd'_S = z.c' \otimes d'_S = z.c \otimes d_{S \cup \{y\}} = zd_{S \cup \{y\}}$.

   (e) $x \notin S$, but $y \in S$. Let $S' = S \setminus \{y\}$. $d'_S = y.d' \otimes d'_{S'} = d_{S'}$. $xd'_S = x.c' \otimes d'_S = \left(x.c \oplus yd_{\{x\}}\right) \otimes d_{S'} = xd_{S'} \oplus yd_{S' \cup \{x\}}$. $zd'_S = z.c' \otimes d'_S = z.c \otimes d_{S'} = zd_{S'}$.

   (f) Both $x, y \in S$. Let $S' = S \setminus \{x, y\}$. $d'_S = x.d' \otimes y.d' \otimes d'_{S'} = d_{\{x,y\}} \otimes d_{S'} = d_{S' \cup \{x,y\}} = d_S$. $zd'_S = z.c' \otimes d'_S = z.c \otimes d_S = zd_S$.

□

Finally, the containment established by the above theorem is strict.

Lemma 23 *Over the Min-Plus semiring, there exist functions in* $\mathbb{F}(\mathbb{D}, \oplus, \otimes c)$ *which are not in* $\mathbb{R}(\mathbb{D}, \oplus, \otimes d)$.

Proof. An example of such a function is f_3 in figure 1. Regardless of the string w, $f\left(wb^{|w|}e\right) = |w|_a$. Thus, in any state q, the machine has to contain, in some register x_q (possibly dependent on w), $|w|_a + c_{qx}$. However, for all w and $k > 0$, there is some σ so that $|f(w\sigma) - |w\sigma|_a| \geq k$. Let's identify some witness for this by writing $\sigma(w, k)$. In particular, this means that there has to be some register tracking the value of the function, which is distinct from the register x_q tracking $|w|_a$. We have thus established that at least two registers are necessary.

Consider a machine with two registers. For the largest constant c appearing in the description of the machine, consider the string $a^c\sigma(a^c, c)$. At this point, we have two registers - one containing the number of as, and the other containing the function. If we now feed the machine a suffix $b^{|\sigma(\epsilon,c)|}e$, the value of the function is equal to the number of as in the input string. The machine now has two choices: either copy the value, or choose to track both the functions and the number of as in the same register. This latter choice cannot happen: for we can then feed the suffix $\sigma\left(\sigma(\epsilon, c)\, b^{|\sigma(\epsilon,c)|}e\right) b^{\left|\sigma\left(\sigma(\epsilon,c)b^{|\sigma(\epsilon,c)|}e\right)\right|}e$, and force the machine into making a mistake.

For multiple registers, we perform a multi-step pumping argument similar to the above. Define the sequence: $\sigma_1 = a^c\sigma(a^c, c)$, $\sigma_s = \sigma_1 a^c b^{|\sigma_1 a^c|}e$, $\ldots$, $\sigma_{i+1} = \sigma_i a^c b^{|\sigma_i a^c|}e$, $\ldots$ The argument involves observing after reading each σ_i, the number of "useful" variables, in the absence of copyful assignments, decreases by one. □

5.3 Relation to Weighted Automata

In Section 4.2, we noted that single-valued weighted automata correspond exactly to CRAs with addition. Now we show that nondeterministic weighted automata and (deterministic) CRAs (without the copyless restriction) with $\oplus$ and $\otimes c$ express exactly the same class of functions. The translation from weighted automata to CRAs can be viewed as a generalization of the classical subset construction for determinization.

Theorem 24 (Weighted Automata Expressiveness) *If* $(\mathbb{D}, \oplus, \otimes)$ *forms a semiring, then the class of functions* $\mathbb{F}(\mathbb{D}, \oplus, \otimes c)$ *is exactly that representable by weighted automata.*

Proof. Let $W = (\Sigma, P, I, F, E, \lambda, \rho)$ be a weighted automaton. We construct the corresponding CRA M over $(\mathbb{D}, \oplus, \otimes c)$: The set of states $Q = 2^P$. The state set is obtained by the standard subset construction. The intuition is that M is in state $q \subseteq P$ after processing a string w if q is exactly the set of states reached in W after processing w. Thus, $\delta(q, a) = \{p' \in P \mid \exists p \in q, p \to^a_W p'\}$. The initial state q_0 is the set of initial states of the weighted automaton, I. The set of registers is $X = \{x_p \mid p \in P\}$: there is a register x_p for every state $p \in P$. The following is the intuition behind these registers: consider some state p, and all paths from the set of initial states I to p. Along each path, take the $\otimes$-product of the weights, and $\oplus$-add the values thus obtained for all paths. The intent is for x_p to hold this value. For each state $p \in I$, the register x_p is initialized to $\lambda(p)$. Even though in the definition of CRAs, registers were initialized to 0 (or some other constant), by simply adding a new initial state which explicitly initializes registers before use, we can simulate registers being initialized to anything we choose. When the CRA makes a transition $q \to^a q'$, the register update is given by $\forall p' \in q'$, $x_{p'} := \oplus\{x_p \otimes c \mid p \overset{a,c}{\to}_W p'\}$. That is, to obtain the value of $x_{p'}$, we consider each state p such that there is an a-labeled transition from p to p' with cost c in

the weighted automaton, add c to x_p according to $\otimes$, and take $\oplus$ over all such values. In state $q \subseteq P$, the output function is defined as $\oplus \{x_p \otimes \rho(p) \mid p \in q\}$. The output function is the $\oplus$-sum of all the product of all paths. To prove the correctness of this construction, observe the inductive invariant: for all w, and for all states p, the register x_p in the CRA M contains the $\oplus$-sum of the $\otimes$-product of the weights of all paths leading from some initial state to p on w. This depends on the distributivity of $\otimes$ over $\oplus$. Note that the same register x_p contributes to all $x_{p'}$s for all its a-successor states p'. Thus, the update is not necessarily copyless.

In the reverse direction, let M be a CRA over $(\mathbb{D}, \oplus, \otimes c)$ with states Q and registers X. Construct the following weighted automaton W:

1. The state set P is $Q \times X \cup Q$. Intuitively, a state $(q, x) \in Q \times X$ calculates transformations happening to individual registers, and states $q \in Q$ calculates the constant offset possibly imposed by the output function. Formally, after processing some word w, if M reaches state q, then the value of register x is equal to the value reaching state (q, x): along each path from some initial state to (q, x), multiply all the weights, and add the values thus obtained along all such paths. Also, such paths exist iff processing w takes M to state q.

2. The initial states are $I = \{q_0\} \times X \cup \{q_0\}$. All initial weights are equal to the multiplicative identity.

3. Say the output function at $q \in Q$ in M is given by $\mu_q(\mathbf{x}) = \oplus_i (x_i \otimes a_i) \oplus c_q$. Then, for each i, the output weight $\rho(q, x_i) = a_i$, and $\rho(q) = c_q$.

4. For every transition $q \to^a q'$ in M, create the transition $q \to^{a,1} q'$ in W. Also, for every variable update $x := \oplus_i (x_i \otimes a_i) \oplus c_x$ that occurs during this transition, create the transitions $(q, x_i) \to^{a,a_i} (q', x)$ in W (for each i). Also add the transition $q \to^{a,c_x} (q', x)$. That the intended invariant is maintained follows from the distributivity properties of a semiring.

□

5.4 Decision Problems for Min-Plus Models

Now we turn our attention to semirings in which the cost domain is a numerical domain such as $\mathbb{N} \cup \{\infty\}$, $\oplus$ is the minimum operation, and $\otimes$ is the addition. First let us consider shortest path problems for CRAs over the cost model $(\mathbb{Q} \cup \{\infty\}, \min, +c)$. Given a CRA over such a cost model, we can construct a weighted automaton using the construction in the proof of Theorem 24. Shortest paths in a weighted automaton can be solved in polynomial-time using standard algorithms [14].

Theorem 25 (Shortest Paths in CRAs over min and $+c$) *Given a CRA M over the cost model $(\mathbb{Q} \cup \{\infty\}, \min, +c)$, computing $\min\{M(w) \mid w \in \Sigma^*\}$ is solvable in* PTIME.

It is known that the equivalence problem for weighted automata over the Min-Plus semiring is undecidable. It follows that checking whether two CRAs over the cost model $(\mathbb{N} \cup \{\infty\}, \min, +c)$ compute the same cost function, is undecidable. The existing proofs of the undecidability of equivalence rely on the unrestricted non-deterministic nature of weighted automata, and thus on the copyful nature of CRAs with min and $+c$. We conjecture that the equivalence problem for copyless CRAs over $(\mathbb{N} \cup \{\infty\}, \min, +c)$, and also for the class $\mathbb{R}(\mathbb{N} \cup \{\infty\}, \min, +c)$ is decidable.

6 Discounted Costs

In this section, we focus on the class of regular cost functions definable using $+c$ and $*d$. Such cost functions allow both adding costs and scaling by discount factors.

6.1 Past Discounts

First let us focus on CRAs over the cost model $\mathbb{C}(\mathbb{Q}, +c, *d)$. At every step, such a machine can set a register x to the value $d * x + c$: this corresponds to discounting previously accumulated cost in x by a factor d, and paying an additional new cost c. We call such machines the *past-discount* CRAs (see f_4 of Figure 1 for an example). Note that the use of multiple registers means that this class of cost functions is closed under regular choice and regular look-ahead: the discount factors can depend conditionally upon future events. It is easy to check that the cost functions definable by past-discount CRAs belong to the class $\mathbb{R}(\mathbb{Q}, +c, *d)$. Our main result for past-discount CRAs is that the min-cost problem can be solved in polynomial-time. First, multiple registers can be handled by considering a graph whose vertices are pairs of the form (q, x), where q is a state of the CRA, and x is a register. Second, classical shortest path algorithm can be easily modified when the update along an edge scales the prior cost before adding a weight to it.

Theorem 26 (Shortest Paths for Past Discounts) *Given a past-discount CRA M over the cost model $(\mathbb{Q}, +c, *d)$, computing $\min\{M(w) \mid w \in \Sigma^*\}$ is solvable in* PTIME.

Proof. We reduce this problem to the *generalized shortest path* problem (see [6, 30]) on a graph, where edges are parameterized by cost $c(e)$ and weight $w(e)$ and the cost of a path $p = (e_1, \ldots, e_n)$ is $c(p) = c(e_1) + w(e_1) * (c(e_2) + w(e_2) * (\ldots + w(e_{n-1}) * c(e_n)))$, while the weight of p is $w(p) = w(e_1) * \ldots * w(e_n)$.

Consider a past-discount CRA $M = (\Sigma, Q, q_0, X, \delta, \rho, \mu)$. Without loss of generality, we can assume that for each $q, q' \in Q$, $a \in \Sigma$, if $\delta(q, a) = q'$, then for all $y \in X$, $\rho(q, a, y)$ has the form $dx + c$ for some $x \in X$; and for all $q \in Q$, if $\mu(q)$ is defined, then it has the form $dx + c$ for some $x \in X$. We construct the following graph $G = (V, E)$. For each $q \in Q$ and $x \in X$, G has a vertex (q, x). Moreover, G has a source vertex s, a target vertex t. The graph G maintains the invariant that there is a path from (q, x) to t if and only if there is a run $(q_0, \nu_0) \ldots (q_n, \nu_n)$ of M, such that $\nu_n(x) = c(p)$. Formally, for each $q, q' \in Q$ and $a \in \Sigma$, such that $\delta(q, a) = q'$, for any register $y \in X$: if $\rho(q, a, y) = dx + c$, for some x, G has an edge e from (q', y) to (q, x) with weight $w(e) = d$ and cost $c(e) = c$. For all $q \in Q$, if $\mu(q) = dx + c$, G has an edge e from s to (q, x) with weight $w(e) = d$, cost $c(e) = c$; for each $x \in X$, G has an edge e from (q_0, x) to t, with weight $w(e) = 0$, cost $c(e) = 0$.

Given a run $(q_0, \nu_0) \ldots (q_n, \nu_n)$ of M on input string $w = w_1 \ldots w_n$, if the output is defined, we claim that there is an $s - t$ path p in G such that the cost of p is equal to the output of M on w. For convenience, let's define r_i be the register that contributes to final output on state q_i, i.e. $\mu(q_n) = dr_n + c$; $\rho(q_i, w_i, r_{i+1}) = dr_i + c$ for all $i < n$. Let's define $f_i(r_i)$ be the function that outputs on state q_i, i.e. $f_n(r_n) = \mu(q_n)$, and for $i < n$, $f_i(r_i) = f_{i+1}(\rho(q_i, w_i, r_{i+1}))$. It's clear that $f_0(0)$ is the final output of M on w. Let p_i be the path $(s, v_n, \ldots, v_i)$, where $v_j = (q_j, r_j)$. It is easy to see that the cost of p_0 is equal to the cost of the path $p = (s, v_n, \ldots, v_0, t)$. We inductively prove that $f_i(r_i) = w(p_i)r_i + c(p_i)$. In the base case, if $\mu(q_n) = dr_n + c$, by construction, $c(s, v_n) = c$ and $w(s, v_n) = d$. Therefore, $f_n(r_n) = dr_n + c = w(p_n)r_n + c(p_n)$. Suppose, for all $i \geq k$, the inductive invariant holds. If $\rho(q_{k-1}, w_k, r_k) = dr_{k-1} + c$, then $f_{k-1}(r_{k-1}) = f_k(dr_{k-1} + c) = w(p_k)(dr_{k-1} +$

$c) + c(p_k) = dw(p_k)r_{k-1} + w(p_k)c + c(p_k) = w(p_{k-1})r_{k-1} + c(p_{k-1})$, since $w(v_k, v_{k-1}) = d$ and $c(v_k, v_{k-1}) = c$. Therefore, the output of M on w $f_0(0) = c(p_0) = c(p)$. On the other hand, given an $s-t$ path $p = (s, (q_n, x_n), \ldots, (q_0, x_0), t)$, it is easy to see that there exists a run $(q_0, \nu_0) \ldots (q_n, \nu_n)$ of M on some string w, that the output of M is equal to $c(p)$. □

6.2 Future Discounts

Symmetric to past discounts are future discounts: at every step, the machine wants to pay an additional new cost c, and discount all future costs by a factor d. While processing an input $w_1 \ldots w_n$, if the sequence of local costs is $c_1, \ldots c_n$ and discount factors is $d_1, \ldots d_n$, then the cost of the string is the value of the term $(c_1 + d_1 * (c_2 + d_2 * (\cdots)))$. *Future-discount* CRAs are able to compute such cost functions using registers that range over $\mathbb{Q} \times \mathbb{Q}$ and substitution: each register holds a value of the form (c, d) where c is the accumulated cost and d is the accumulated discount factor, and updates are defined by the grammar $e := (c, d)|e[c, d]|x$. The interpretation for $e[c, d]$ is defined to be $(e.c + c * e.d, e.d * d)$ (that is, the current discount factor $e.d$ is scaled by new discount d, and current cost $e.c$ is updated by adding new cost c, scaled by the current discount factor $e.d$). Like past-discount CRAs,future-discount CRAs are closed under regular choice and regular look-ahead. Processing of future discounts in forward direction needs maintaining a pair consisting of cost and discount, and the accumulated costs along different paths is not totally ordered due to these two objectives. However, if we consider paths in "reverse", a single cost value updated using assignments of the form $x := d * x + c$ as in past-discount CRAs suffices.

Theorem 27 (Shortest Paths for Future Discounts) *Given a future-discounted CRA M over the cost model $(\mathbb{Q}, +c, *d)$, computing $\min\{M(w) \mid w \in \Sigma^*\}$ is solvable in* PTIME.

Proof. We reduce this problem to the *generalized shortest path* problem (see [6, 30]) on a graph.

Consider a future-discount CRA $M = (\Sigma, Q, q_0, X, \delta, \rho, \mu)$. First we construct an equivalent future-discount CRA M' that every updating function and output function has the form $x[c, d]$. Formally, $M' = (\Sigma, Q, q_0, X', \delta, \rho', \mu')$, where $X' = X \cup \{\varepsilon\}$, such that $\varepsilon \notin X$. For each $q, q' \in Q$, and $a \in \Sigma$, such that $\delta(q, a) = q'$, for each $y \in X$:

1. if $\rho(q, a, y) = x[c, d]$ for some x, $\rho'(q, a, y) = x[c, d]$;
2. if $\rho(q, a, y) = (c, d)$, $\rho'(q, a, y) = \varepsilon[c, d]$;
3. $\rho'(q, a, \varepsilon) = \varepsilon(0, 1)$.

For each $q \in Q$, $\mu'(q) = \varepsilon(c, d)$ if $\mu(q) = (c, d)$, $\mu'(q) = \mu(q)$ otherwise. Second, we construct the graph $G = (V, E)$ with source s and target t, such that there is a path p from s to $v(\neq t)$ if and only if there is a run $(q_0, \nu_0) \ldots (q_n, \nu_n)$ of M' and some $x \in X$, such that $\nu_n(x) = (c(p), w(p))$. Formally, for each $q \in Q$, and each $x \in X'$, G has a vertex (q, x). Moreover, G has a source vertex s, a target vertex t. For each $q, q' \in Q$ and $a \in \Sigma$, such that $\delta(q, a) = q'$, for each register $y \in X$: if $\rho'(q, a, y) = x[c, d]$ for some x, G has an edge e from (q, x) to (q', y), with weight $w(e) = d$, cost $c(e) = c$; Finally, for each $q \in Q$, if $\mu'(q) = x[c, d]$, G has an edge e from (q, x) to t with weight $w(e) = d$, cost $c(e) = c$; for each $x \in X$, G has an edge e from s to (q_0, x), with weight $w(e) = 1$, cost $c(e) = 0$.

Given a run $(q_0, \nu_0) \ldots (q_n, \nu_n)$ of M' on input string $w = w_1 \ldots w_n$, if the output is defined, we claim that there is an $s - t$ path p in G such that the cost of p is equal to the output of M' on

w. Let's define r_i be the register that contributes to final output on state q_i, i.e. $\mu'(q_n) = r_n[c, d]$; $\rho'(q_i, w_i, r_{i+1}) = r_i[c, d]$ for all $i < n$. Let p_i be the path $(s, v_0, \ldots, v_i)$, where $v_j = (q_j, r_j)$ and $p = (s, v_0, \ldots, v_n, t)$. For simplicity. we abuse the notation of the name of register and functions to mean their values under interpretation. We inductively prove that $r_i = (c(p_i), w(p_i))$. In the base case, $r_0 = (0, 1) = (c(p_0), w(p_0))$ by construction. Suppose, for all $i \leq k$, the inductive invariant holds. If $\rho(q_k, w_k, r_{k+1}) = r_k[c, d]$, then $r_{k+1} = (r_k.c + c * r_k.d, r_k.d * d) = (c(p_k) + c * w(p_k), w(p_k) * d) = (c(p_{k+1}), w(p_{k+1}))$,since by construction, $c(v_k, v_{k+1}) = c$ and $w(v_k, v_{k+1}) = d$. Therefore, the output of M' on w $\mu'(q_n) = r_n[c, d] = (r_n.c + c * r_n.d, r_n.d * d) = (c(p_n) + c * w(p_n), w(p_n) * d) = (c(p), w(p))$, since by construction $c(v_n, t) = c$ and $w(v_n, t) = d$. On the other hand, given an $s - t$ path $p = (s, (q_0, x_0), \ldots, (q_n, x_n), t)$, it is easy to see that there exists a run $(q_0, \nu_0) \ldots (q_n, \nu_n)$ of M' on some string w, that the output of M' is equal to $c(p)$. □

6.3 Global Discounts

A *global-discount CRA* is capable of scaling the global cost (the cost of the entire path) by a discount factor. As in case of future-discount CRAs, it uses registers that hold cost-discount pairs. We now assume that discounts range over $[0, 1]$ and costs range over $\mathbb{Q}^+$. The registers are updated using the grammar $e := (0, 1) \mid e \pm (c, d) \mid x$. The interpretation for $e \pm (c, d)$ is defined to be $(d * e.c + e.d * c, e.d * d)$ (that is, the current discount factor $e.d$ is scaled by new discount d, and current cost $e.c$ is updated by first scaling it by the new discount, and then adding new cost c scaled by the current discount factor $e.d$). Analyzing paths in a global-discount CRA requires keeping track of both the accumulated cost and discount. We can show a pseudo-polynomial upper bound; it remains open whether there is a strongly polynomial algorithm for shortest paths for this model:

Theorem 28 (Shortest Paths for Global Discounts) *Given a global-discount CRA M over the cost model $(\mathbb{Q}^+, +c, [0, 1], *d)$ and a constant $K \in \mathbb{Q}^+$, deciding $\min\{M(w) \mid w \in \Sigma^*\} \leq K$ is solvable in* NP*. Computing the minimum is solvable in* PTIME *assuming increments are restricted to adding natural numbers in unary encoding.*

Proof. Consider a global-discount CRA $M = (\Sigma, Q, q_0, X, \delta, \rho, \mu)$, we construct the *global-discount graph* $G = (V, E)$, that each edge $e \in E$ is parameterized with a cost $c(e)$ and a discount $d(e)$ as we did above. For a path $p = (e_1, \ldots, e_n)$, the cost of the path is defined as $c(p) = (c(e_1), d(e_1)) \pm (c(e_2), d(e_2)) \pm \ldots \pm (c(e_n), d(e_n)) = \sum_i c(e_i) \prod_i d(e_i)$. Following the proof above, it is easy to see solving the shortest path in M is equivalent to solving the shortest path in G.

To prove the NP bound, we first observe that if there is a reachable cycle that contains an edge e with $d(e) < 1$, then repeating this cycle drives the global discount to 0, and thus, existence of such a cycle implies that the min-cost (at the limit) is 0. Notice that the shortest path doesn't need to involve a cycle in which all discount factors are equal to 1 (since costs are non-negative). The NP-bound follows from following fact. An NP algorithm guesses: 1) a reachable cycle and verifies if there is an edge e with $d(e) < 1$ in this cycle, or 2) a simple path and verify if its cost is less than K. Suppose incremental costs c_i's are small natural numbers. The pseudo-polynomial algorithm for this case relies on the following idea: for a given value c and a vertex v, computing the "best" global discount over all paths from source to v with sum of incremental costs equal to c, can be solved by adopting shortest path algorithms, and the set of interesting choices of c can be bound by nk for a graph with n vertices if each increment is a number between 0 to k. Thus a variation of Bellman-Ford suffices (See algorithm 1). □

Algorithm 1 Shortest Path Algorithm on Global-Discount Graph

Input: A global-discount graph G with source s and target t. Let n be the number of vertices in G and k be the largest cost over edges.
Output: infimum of the cost of s-t paths in G
//$Discount[i, v, c]$ stores the best global discount from s to v among paths with length $\leq i$, when the sum of incremental cost is c.
Delete all the vertices which cannot reach t
$Discount[0, v, c] = \infty$, for every node v and $0 < c \leq nk$
$Discount[0, s, 0] = 1$
for $i = 1$ to n **do**
  for all v in G and $0 < c \leq nk$ **do**
    $Discount[i, v, c] = \min\{Discount[i-1, v, c], \min_{e=(u,v)}\{Discount[i-1, u, c - c(e)] * d(e)\}\}$
  end for
end for
//check if there is a cycle with discount ≤ 1
if $\exists v, c$, s.t. $Discount[n-1, v, c] \neq Discount[n, v, c]$ **then**
  return 0
end if
//Output the shortest simple path
return $\min_{0<c\leq nk}\{Discount[n-1, t, c] * c\}$

6.4 Regular Functions for Inc-Scale Model

The class of regular functions for the cost model $(\mathbb{Q}, +c, *d)$ is defined via SSTTs over the inc-scale grammar $G(+c, *d)$. It is to show that:

Theorem 29 (Expressiveness of Inc-Scale Models) *The cost functions definable by past-discount CRAs, by future-discount CRAs, and by global-discount CRAs all belong to* $\mathbb{R}(\mathbb{Q}, +c, *d)$.

The min-cost problem for this class of functions is still open. However, we can show the equivalence problem to be decidable. First, using the construction similar to the one used to establish $\mathbb{R}(\mathbb{D}, \oplus, \otimes c) \subset \mathbb{F}(\mathbb{D}, \oplus, \otimes c)$ (see Theorem 19), we can represent cost functions in $\mathbb{R}(\mathbb{Q}, +c, *d)$ using (copyful) CRAs that use $+$ and $*d$. Such CRAs have *linear* updates, and the algorithm for checking equivalence of CRAs with addition can be used for this case also.

Theorem 30 *Given two functions $f_1, f_2 \in \mathbb{R}(\mathbb{Q}, +c, *d)$ represented by SSTTs over the cost grammar $G(+c, *d)$, checking whether the two functions coincide, can be solved in time polynomial in the number of states and exponential in the number of registers.*

Proof. Given an SSTT T for $f \in \mathbb{R}(\mathbb{D}, +c, *d)$, we use lemma 22 to construct an equivalent CRA M using $+$ and $*d$. M has the same number of states as T, and an exponential number of variables. We claim that checking equivalence between two CRAs with $+$ and $*d$ is solvable in time polynomial in the number of states and number of variables (the proof is similar to that of theorem 15). Thus, checking the equivalence of two functions f_1 and f_2 expressed as STTs over $G(+c, *d)$ can be done in time polynomial in the number of states and exponential in the number of variables. □

7 Related Work

Weighted Automata (WA) and Logics. Finite-state WA have been an active area of research, with numerous articles studying their algebraic and algorithmic properties. See [14] for a comprehensive exposition. An important problem for WA is that of determinization [14, 24]. A deterministic WA is defined in the usual sense: no two outgoing transitions from a state share the same input label. It has been shown that there are WA that do not admit equivalent deterministic WA. In contrast, the cost register automata that we introduce in this paper are deterministic machines with equivalent expressive power and equally efficient decision problems as weighted automata. We believe that this makes them a more suitable model for expressing weighted computations.

It has been shown that the equivalence problem for WA over the Min-Plus semiring is undecidable using a reduction from Hilbert's tenth problem [25], and by a reduction from the halting problem for two counter machines [1]. The only known class of weighted automata over the Min-Plus semiring with decidable equivalence problem is that of finite-valued weighted automata [32]. For a given k, a weighted automaton is said to be k-valued if the number of distinct values computed along all accepting paths is at most k. A weighted automaton is called finite-valued if there exists a k such that it is k-valued. We conjecture that the equivalence problem for CRA over min and $+c$ with the copyless restriction is decidable. If this is true, it would give the largest known class with decidable equivalence. In [23] the authors provide a randomized algorithm to solve equivalence of weighted automata over the semiring with addition and scaling.

In [13], the authors discuss a weighted MSO-logic that disallows universal second order quantification and places restrictions on universal first order quantification. The authors show that the formal power series definable in this logic coincides with the set of behaviors of weighted automata. In contrast, in this paper, we introduce automata and machines that exactly capture MSO-definable cost functions.

Discounted weighted computations and Generalized Shortest Paths. Generalized network flow problems extend flow problems on directed graphs by specifying multipliers on edges in addition to costs [19, 30]. The problem of finding the minimum cost flow (which in some cases is equivalent to the shortest distance path) from a source to a target can be solved in polynomial time [30, 6]. Future discount machines that we introduce in this paper provide a nice formalism that subsumes such problems, and have strongly polynomial time algorithms for determining the minimum cost path. In this paper, we also introduce past discount and global discount machines that also have efficient algorithms for determining the minimum cost paths.

In [15], the authors introduce weighted logic for infinite words. In order to address convergence of the weighted sum, the authors assume discounting along later edges in a path (*i.e.*, future discounting). Extending the results of this paper to discounted weighted computations over infinite words remains open.

Transducer Models. A wide variety of different models have been proposed to model string and tree transductions. The models that are most relevant to this paper are MSO-definable transductions [12, 16] and macro tree transducers [18, 17]. An MSO-definable graph transduction specifies a function between sets of graphs; the nodes, edges and labels of the output graph are described in terms of MSO formulas over the nodes, edges and labels of a finite number of copies of the input graph. A macro tree transducer (MTT) is a top-down tree to tree transducer equipped with

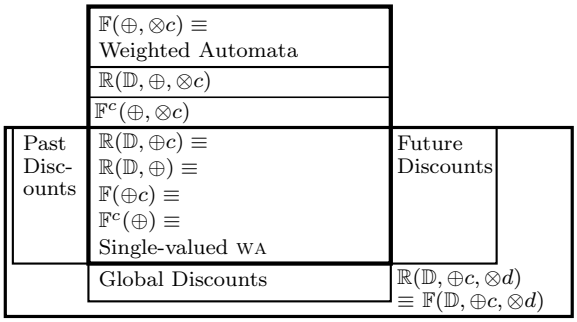
parameters. Parameters can store temporary trees and append them to the final tree during the computation. In general, MTT are more expressive than MSO-definable tree transductions. A subclass of MTTs obtained by restricting the number of times a subtree and a parameter can be used has been shown to be equi-expressive as MSO-definable tree transductions [17]. In addition to these models, formalisms such as attribute grammars [17], attribute tree transducers [8] have also been studied.

Streaming tree transducers [4] (STTs), introduced by two of the co-authors in this paper are a new formalism for expressing MSO-definable tree-to-tree and string-to-tree transductions. In comparison to some of the transducer models discussed above, STTs have distinguishing features that make them desirable as a canonical model for specifying regular or MSO-definable transductions: (1) STTs produce the output in linear time by performing a single pass over the input, (2) they preserve desirable properties such as closure under sequential composition and regular look-ahead, and (3) they have good algorithmic properties such as decidability of functional equivalence.

Regularity over Data Languages. Data languages allow finite strings over data values that can be drawn from a possibly infinite data domain [29], [21] [7]. Register automata are often used as acceptors for data languages. A key feature of such automata is that they allow registers to store and test data values. Beyond the similarity in nomenclature, register automata that are studied in this line of work are quite distinct from cost register automata introduced in this paper. The former are essentially defined over an infinite input alphabet, and the critical difference lies in the fact that almost every variant of data automata allows testing equality of data values, which mostly causes interesting decision problems to become undecidable. Cost register automata use the cost registers in a strictly write-only fashion, which makes them incomparable to variants of data automata that use read/write registers.

Regular Cost Functions. In [10], Colcombet defines a regular cost function as a mapping from words to $\mathbb{N}^\omega$ (the set of nonnegative integers and the ordinal ω). A cost function is precisely defined as an equivalence class over mappings from the set of words to $\mathbb{N}^\omega$, such that functions f and g are in the same equivalence class if for all words w, $f(w)$ is bounded by some constant iff $g(w)$ is bounded is bounded by some constant. The author then defines two classes of automata (B- and S-automata), each of which uses a finite set of counters and allows the counters to be incremented, reset or checked for equality with a constant. The operational semantics of these automata are that the automaton computes the least upper bound or the greatest lower bound over the set of counter values encountered during its run. A cost function is then called regular if it is accepted by a history-deterministic B- or S-automaton. The author also provides an algebraic characterization of regular cost functions in terms of stabilization monoids and equates recognizability of cost functions with regularity. In [11], the authors extend this notion to regular cost functions over trees.

It is clear that the notions proposed in this line of work are orthogonal to our characterization of regularity of cost functions. The authors state that the motivation for the work in [10, 11] is preserving nice algorithmic and closure properties of regular languages for problems such as equivalence and projection. However, the integer values in these functions are considered modulo an equivalence which preserves existence of bounds on the function values, but not the values themselves. We believe that the notion of regularity of cost functions that we propose in this paper is closer to the classical notions of regularity such as MSO-definability.



(a) Hierarchy for CRA models

CRA with	Equivalence	Min-Cost
$(+c)$	PTIME	PTIME
Copyless $(+)$	PTIME	EXPTIME
$(min, +c)$	Undecidable	PTIME
Copyless $(min, +c)$	?	PTIME
Past-discounts	Poly in states Exp in registers	PTIME
Future-discounts		PTIME
Global-discounts		Pseudo-Poly
Inc-Scale		?

(b) Complexity of Decision Problems

Figure 10: Summary of Results

Affine Programs. In [22], and more recently in [27, 28], the authors present the problem of deriving affine relations among variables of a program. An affine relation is a property of the form $a_0 + \sum_{i=1}^{n} a_i.\mathbf{v_i} = 0$, where $\mathbf{v_1}, \ldots, \mathbf{v_n}$ are program variables that range over a field such as the rationals or reals and a_i are constants over the same domain. An affine program is a program with nondeterministic branching where each edge of the program is labeled with an assignment statement of the form $v_1 := v_2 + 2.v_3 + 3$, *i.e.*, where the RHS is an affine expression. We could define a CRA over the cost model $\mathbb{C}(\mathbb{Q}, +, \mathbb{Q}, *d)$, with the cost grammar $t := +(t, t) \mid *(t, d) \mid c$. An affine program is then simply obtained by ignoring the input labels of the transitions in such a CRA. While the cost functions defined by such CRA do not have interesting regularity properties, we remark that the equivalence of such CRA can be checked in polynomial time by using ideas similar to the ones in [27].

Quantitative Languages. A quantitative language [9, 1, 5] over infinite words is a function $\Sigma^\omega \mapsto \mathbb{R}$. Such languages are generated by weighted automata, where the value of a word w is set as the maximal value of all runs over w. By defining various value functions such as *Max*, *Sum*, *LimSup*, *LimInf*, different values can be computed for the run of a weighted automaton on the string w. Quantitative languages use the fixed syntax of weighted automata, and thereby restricted to having a single weight along each transition in their underlying automata. Moreover, they face similar difficulties in determinization: for interesting models of value functions, the corresponding automata cannot be determinized. An extension of CRA to ω-regular cost functions could prove to be a more expressive and robust model to specify quantitative languages and to analyze their decision problems.

8 Conclusions

We have proposed a new approach to define regular functions for associating costs with strings. The results for various classes of functions are summarized in Figure 2. We hope that our work provides new insights into the well-studied topic of weighted automata, and opens a whole range of new problems. First, it is plausible that there is a compelling notion of congruences and canonicity for CRAs with increment. Second, the decidability of copyless-CRAs with min and increment remains an intriguing open problem. Third, we don't have algorithms for the min-cost problem for

the class of regular functions with increment and scaling. While we have not succeeded even in establishing decidability, we suspect that this problem admits efficient approximation algorithms. Fourth, we have considered only a small set of combinations of operations; studying the effects of adding operators such as *max* would be worthwhile. Fifth, our notion of regularity and cost register automata for mapping strings to costs can be extended to infinite strings and trees, as well as to timed and probabilistic systems. Finally, we would like to explore practical applications: our framework seems suitable for expressing complex, yet analyzable, pricing policies, say, for power distribution.

References

[1] S. Almagor, U. Boker, and O. Kupferman. What's decidable about weighted automata? In *Automated Technology for Verification and Analysis*, Lecture Notes in Computer Science, pages 482–491, 2011.

[2] R. Alur and P. Černý. Expressiveness of Streaming String Transducers. In *Proc. of Foundations of Software Technology and Theoretical Computer Science*, pages 1–12, 2010.

[3] R. Alur and P. Černý. Streaming Transducers for Algorithmic Verification of Single-pass List-processing Programs. In *Proc. of Principles of Programming Languages*, pages 599–610, 2011.

[4] R. Alur and L. D'Antoni. Streaming tree transducers. In *Proc. of the 39th Intl. Colloquium on Automata, Languages, and Programming - Volume Part II*, pages 42–53, 2012.

[5] B. Aminof, O. Kupferman, and R. Lampert. Reasoning about online algorithms with weighted automata. *ACM Transactions on Algorithms*, 2010.

[6] V. Batagelj, F. J. Brandenburg, P. O. D. Mendez, and A. Sen. The generalized shortest path problem, 2000.

[7] H. Björklund and T. Schwentick. On notions of regularity for data languages. *Theor. Comput. Sci.*, 411(4-5):702–715, 2010.

[8] R. Bloem and J. Engelfriet. A comparison of tree transductions defined by monadic second order logic and by attribute grammars. *J. Comp. and Sys. Sci.*, 61(1):1 – 50, 2000.

[9] K. Chatterjee, L. Doyen, and T. A. Henzinger. Quantitative languages. *ACM Trans. Comput. Log.*, 11(4), 2010.

[10] T. Colcombet. The theory of stabilisation monoids and regular cost functions. pages 139–150, 2009.

[11] T. Colcombet and C. Loding. Regular cost functions over finite trees. In *Symposium on Logic in Computer Science*, pages 70–79, 2010.

[12] B. Courcelle. Graph Operations, Graph Transformations and Monadic Second-Order Logic: A survey. *Electronic Notes in Theoretical Computer Science*, 51:122 – 126, 2002.

[13] M. Droste and P. Gastin. Weighted automata and weighted logics. In *Proc. of Intl. Colloquium on Automata, Languages and Programming*, pages 513–525, 2005.

[14] M. Droste, W. Kuich, and H. Vogler. *Handbook of Weighted Automata*. Springer, 2009.

[15] M. Droste and G. Rahonis. Weighted automata and weighted logics with discounting. *Theor. Comp. Sci.*, 410(37):3481 – 3494, 2009.

[16] J. Engelfriet and H. J. Hoogeboom. MSO definable String Transductions and Two-way Finite-State Transducers. *ACM Transactions on Computational Logic*, 2(2):216–254, 2001.

[17] J. Engelfriet and S. Maneth. Macro tree transducers, attribute grammars, and mso definable tree translations. *Information and Computation*, 154(1):34 – 91, 1999.

[18] J. Engelfriet and H. Vogler. Macro tree transducers. *Journal of Computer and System Sciences*, 31(1):71 – 146, 1985.

[19] A. V. Goldberg, S. A. Plotkin, and É. Tardos. Combinatorial Algorithms for the Generalized Circulation Problem. In *Foundations of Computer Science*, pages 432–443, 1988.

[20] H. Hosoya. *Foundations of XML Processing: The Tree-Automata Approach*. Cambridge University Press, 2011.

[21] M. Kaminski and N. Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.

[22] M. Karr. Affine relationships among variables of a program. *Acta Inf.*, 6:133–151, 1976.

[23] S. Kiefer, A. S. Murawski, J. Ouaknine, B. Wachter, and J. Worrell. Language equivalence for probabilistic automata. In *Proc. of the 23rd Intl. conference on Computer aided verification*, pages 526–540, 2011.

[24] D. Kirsten and I. Mäurer. On the determinization of weighted automta. *J. Autom. Lang. Comb.*, 10:287–312, 2005.

[25] D. Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. In *Intl. Colloquium on Automata, Languages and Programming*, pages 101–112, 1992.

[26] M. Mohri, F. Pereira, and M. Riley. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88, 2002.

[27] M. Müller-Olm and H. Seidl. A note on Karr's algorithm. In *Intl. Colloquium on Automata, Languages and Programming*, pages 1016–1028, 2004.

[28] M. Müller-Olm and H. Seidl. Precise interprocedural analysis through linear algebra. In *Principles Of Programming Languages*, pages 330–341, 2004.

[29] F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004.

[30] J. D. Oldham. Combinatorial Approximation Algorithms for Generalized Flow Problems. In *Symp. On Discrete Algorithms*, pages 704–714, 1999.

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

[32] A. Weber. Finite-valued distance automata. *Theor. Comput. Sci.*, 134:225–251, November 1994.