Regular Functions and Cost Register Automata

Rajeev Alur*, Loris D'Antoni*, Jyotirmoy Deshmukh†, Mukund Raghothaman* and Yifei Yuan*
* University of Pennsylvania † Toyota Technical Center

(Invited Paper)

***Abstract*—We propose a deterministic model for associating costs with strings that is parameterized by operations of interest (such as addition, scaling, and minimum), 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 conditioned on 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 the classical shortest-path algorithms as well as the algorithms designed for computing *discounted costs* can be adapted for solving the min-cost problems for the more general classes of functions specified in our model. Cost register automata with the operations of *minimum* 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.**

I. Introduction

A. 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 each coffee either \$1 or \$2, depending on whether or not you fill out a survey." Such constraints can be captured by the well-studied framework of *weighted automata* ([24], [12]). A weighted automaton is a *nondeterministic* finite-state automaton whose edges are labeled both 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. 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 ([7], [4], [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 minimum). Thus, weighted automata, and other existing frameworks (see [9], [22]), 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. The existing work on "generalized shortest paths" considers extensions that allow costs with future discounting, and while it develops interesting polynomial-time algorithms ([15], [23]), 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?*

The full version of this paper may be found at http://www.cis.upenn.edu/~alur/rca12.pdf. This research was partially supported by the NSF Expeditions in Computing award 1138996.

B. Proposed Definition of Regularity

When should 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 should be 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 $t := +(t,t) \,|\, c$ specifies terms that can be built from constants using a binary operator $+$, and the grammar $t := +(t,c) \,|\, *(t,d) \,|\, c$ specifies terms that can be built from constants using two binary operators $+$ and $*$ in a restricted way. Given a function g mapping 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 $[\![g(w)]\!]$ in the cost domain $\mathbb{D}$. 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 characterizations using transducer models as well as monadic second order logic ([14], [10], [3], [16]). 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 an evaluation according to the interpretation $[\![.]\!]$.*

C. 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* ([2], [3]), 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 $x := y + c$, where x and y are registers and c is a constant, 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, minimum, and scaling by a factor, we explore the following two variants. First, we consider models in which registers hold not only the 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 each step. This "single-use-restriction" plays an important role in the theory of regular tree transducers ([14], [3]), and ensures that the costs (or the sizes of the terms that capture the costs) grow only linearly with the length of the input.

D. *Contributions*

In Section IV we study the class of cost functions over a domain $\mathbb{D}$ with a commutative and associative function $\otimes$; in Section V, 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 VI, we consider different forms of discounted cost functions with scaling and addition. In each case, we identify the operations a CRA must use to be equivalent in expressiveness to the corresponding class of regular functions, and present algorithms for computing the min-cost value and for checking equivalence of two 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 the past and the 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* ([17], [22], [6]). Fifth, it is known that weighted automata are not determinizable, which has sparked extensive research ([12], [19]). Our results show that in presence of multiple registers that can be updated by explicitly applying both the operations of the semiring, the 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 restriction. It is known that checking equivalence of weighted automata over the Min-Plus semiring (natural numbers with min and addition) is undecidable ([20], [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.

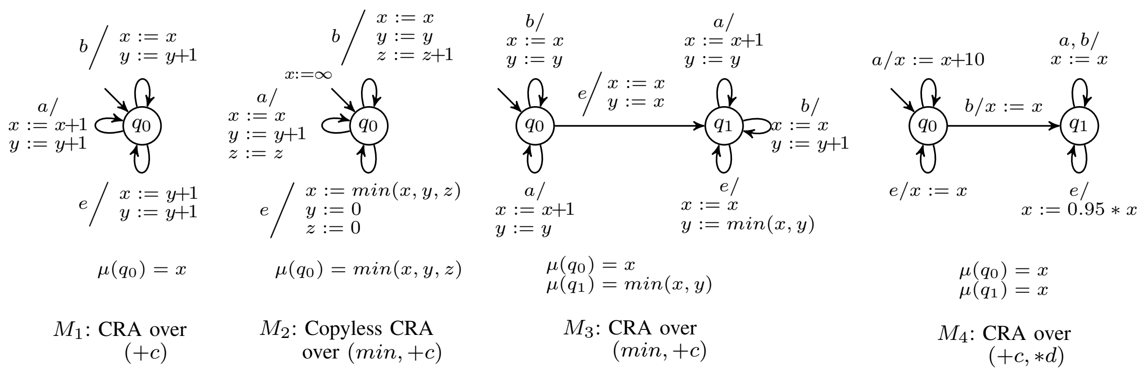
II. Cost Register Automata

A. *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 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 (that is, the set of parse trees is a regular tree language). Usually 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 $t := +(t, t) \,|\, c$, and the terms of the *increment-grammar* $G(+c)$, which restricts the use of the addition operation 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 $t := \min(t, t) \,|\, +(t, c) \,|\, c$. 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.

B. *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. $\mathbb{C}(\mathbb{Q}^+, +c, [0, 1], *d)$ denotes the cost model with the cost grammar $G(+c, *d)$, the set $\mathbb{Q}^+$ of non-negative rational numbers as the domain for the $+$ operator (interpreted as standard addition), and the closed



$b/\ x := x,\ y := y{+}1$
$a/\ x := x{+}1,\ y := y{+}1$
$e/\ x := y{+}1,\ y := y{+}1$
$\mu(q_0) = x$
M_1: CRA over $(+c)$
$b/\ x := x,\ y := y,\ z := z{+}1$
$x{:=}\infty$
$a/\ x := x,\ y := y{+}1,\ z := z$
$e/\ x := min(x, y, z),\ y := 0,\ z := 0$
$\mu(q_0) = min(x, y, z)$
M_2: Copyless CRA over $(min, +c)$
$b/\ x := x,\ y := y$
$e/\ x := x,\ y := x$
$a/\ x := x{+}1,\ y := y$
$a/\ x := x{+}1,\ y := y$
$b/\ x := x,\ y := y{+}1$
$e/\ x := x,\ y := min(x, y)$
$\mu(q_0) = x$
$\mu(q_1) = min(x, y)$
M_3: CRA over $(min, +c)$
$a/x := x{+}10$
$b/x := x$
$a, b/\ x := x$
$e/x := x$
$e/\ x := 0.95 * x$
$\mu(q_0) = x$
$\mu(q_1) = x$
M_4: CRA over $(+c, *d)$

Fig. 1. Examples of Cost Register Automata

interval $[0, 1]$ of rational numbers as the domain for the $*$ operator (interpreted as multiplication).

C. 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. The definition of such a machine is parameterized by the set of updates to its registers. For the 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 leaf node in a term 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 $e := +(e, e) \,|\, c \,|\, x$, for $x \in X$; and for the $G(+c, *d)$, we get the expressions $E(+c, *d, X)$ defined by the grammar $e := +(e, c) \,|\, *(e, d) \,|\, c \,|\, x$, for $x \in X$. We assume that the ranked alphabet contains a special constant symbol 0, which will be used to initialize 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 (dependent on the domain). Given a string $w = a_1 \ldots a_n \in \Sigma^*$, the run of M on w is a sequence of configurations $(q_0, \nu_0) \ldots (q_n, \nu_n)$ 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 $[\![\nu_n(\mu(q_n))]\!]$.

D. 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.

E. 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.

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 $w_1\, e\, w_2 \ldots e\, w_{n-1}\, e\, w_n$, where each block w_i contains only a's and b's, let $f_2(w)$ be the minimum of the set $\{|w_i|_a, |w_i|_b \mid \forall i, 1 \leq i \leq n\}$. 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 $G(+c, *d)$.

F. 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 each input symbol

a and each register $x \in X$, the register x appears at most once in the set of expressions $\{\rho(q,a,y) \mid y \in X\}$ 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 M_2 and M_4 are copyless, while M_1 and M_3 are not.

G. Regular Look-Ahead

A CRA M^R with *regular look-ahead* (CRA-RLA) is an extension of the CRA model in which the machine 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 $a_j \dots a_k$ belongs to L corresponds to testing whether the state of A after processing $a_k \dots a_j$ is an accepting state of A. This motivates the following formalization. Let $w = a_1 \dots a_k$ 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 $w_A = r_1 r_2 \dots r_k$ over the alphabet R such that for each position $1 \le j \le k$, the corresponding symbol is the state of the DFA A after reading $a_k \dots a_j$ (it reads the reverse of the word). A CRA-RLA consists of a DFA A over Σ with states R, and a CRA M over the input alphabet R. The output of CRA-RLA (M, A) on w, denoted by $[\![(M,A), \mathbb{C}]\!](w)$, is defined as $[\![M, \mathbb{C}]\!](w_A)$.

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

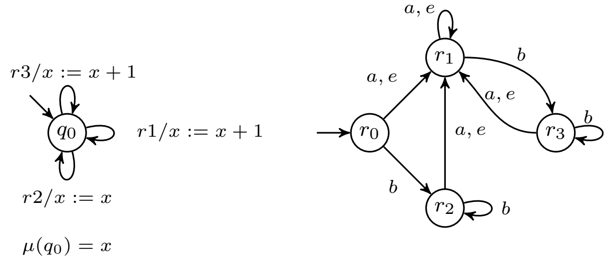


Fig. 2. CRA with Regular Look-Ahead

III. 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) [14], Monadic-Second-Order logic definable graph transformations [10], and streaming tree transducers [3]. We first briefly recap the model of streaming string-to-tree transducers.

A. 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 $t := +(t, c) \mid c \mid ?$; and for the cost grammar $G(+)$, the set $T^?$ of parameterized terms is defined by the grammar $t' := +(t, t') \mid +(t', t) \mid c \mid ?$, where t stands for (complete) terms generated by the original grammar $t := +(t,t) \mid c$. 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 [3] for details.

B. 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 that are 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 the input $w_1 \ldots w_n$, 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 the input $w_1 \ldots w_n$, 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.

C. 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.

For a string w, let w^r denote the reverse string. We define the *reverse function* $f^r : \Sigma^* \mapsto \mathbb{D}$ such that for all $w \in \Sigma^*$, $f^r(w) = f(w^r)$. Given regular 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. As the SSTT class is closed under reverse and regular choice, closure under these properties for regular cost functions follows.

Theorem 1 (Closure Properties of Regular Cost Functions) *For every cost model $\mathbb{C}$, if a cost function f belongs to the class $\mathbb{R}(\mathbb{C})$, then so does the function f^r. If two cost functions f_1 and f_2 belong to the $\mathbb{R}(\mathbb{C})$, then so does the function "if L then f_1 else f_2" for every regular language L.*

An SSTT with regular look ahead is a pair (U, A) where A is a DFA, U is an SSTT whose alphabet is the set of states of A. 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 [3], which implies the same for regular cost functions.

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

D. Linear Outputs

When an SSTT U computes the output term corresponding to an input string w, while processing each symbol of w, it uses exactly one transition with copyless update, and 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 $O(|w|)$, and in particular, the function $f(w) = |w|^2$ is not regular in this cost model. Revisiting the examples in Section 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 is not regular for $\mathbb{C}(\mathbb{N}, \min, +c)$, as it requires $O(|w|^2)$ terms to construct it.

IV. Commutative-Monoid Cost Functions

Assume that the cost model is a commutative monoid $(\mathbb{D}, \otimes)$: $\mathbb{D}$ is a set, the interpretation $[\![\otimes]\!]$ is a commutative and associative function, and the interpretation of the initial constant $[\![0]\!]$ is the identity element of this function.

A. 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 via the grammar $G(\otimes c)$ and the class $\mathbb{R}(\mathbb{D}, \otimes)$ defined via the grammar $G(\otimes)$. Relying of 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.

Theorem 3 (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 Sketch: We first show that the function-classes $\mathbb{R}(\mathbb{D}, \otimes)$ and $\mathbb{R}(\mathbb{D}, \otimes c)$ are equivalent. This proof is based on the properties of the model of *streaming tree transducers* (STT) that map trees to trees [3]. Given an STT U from Σ^* to terms of $G(\otimes)$, we show how to construct an STT U' from terms over $G(\otimes)$ to equivalent terms over $G(\otimes c)$ relying on the associativity of $\otimes$ to rewrite the parse tree in a left-linear form. The proof follows since STTs are closed under sequential composition. Next we show that the class $\mathbb{F}(\mathbb{D}, \otimes c)$ is included in the class $\mathbb{R}(\mathbb{D}, \otimes c)$. Given a CRA M using the increment operation, we construct an SSTT U that captures its computations. The proof relies on the observation that at every step of the computation of M, at most one register can contribute to the final output, and this variable can be specified using a regular-look-ahead (a test that checks whether the input suffix is in a regular language). The proof follows from the fact that SSTTs are closed under regular look-ahead. For the inclusion of the class $\mathbb{R}(\mathbb{D}, \otimes c)$ in the class $\mathbb{F}^c(\mathbb{D}, \otimes)$, observe that the parameter substitution in a unary tree can be simulated by a normal assignment for a commutative operator ($e[e']$ is equivalent to $\otimes(e, e')$). Finally, given a copyless CRA

M with $\otimes$, we construct an equivalent CRA M' that uses only increments. For every subset S of registers in M, M' maintains a register x_S that maintains the sum of all the values of the registers in S. The simulation requires updates in M' to allow the same subset to be used in different ways, that is, in a non-copyless manner. $\square$

We can also establish the following results regarding expressiveness of different classes. First, requiring CRAs with only increment to be copyless is too limiting: there exists a regular additive cost function that is not in $\mathbb{F}^c(\mathbb{D}, \otimes c)$. Second, removing the copyless restriction from CRAs with addition is too permissive as it would allow computing cost functions that grow exponentially: the cost function $f(w) = 2^{|w|}$ is not additive regular, but is in $\mathbb{F}(\mathbb{N}, +)$. Third, having multiple registers is essential for expressive completeness: for every k, consider the cost function $f_k : \mathbb{N} \to \mathbb{N}$ (over unary alphabet) defined by $f_k(n) = ((n \bmod k) + 1) * n$; every CRA with increment implementing f_k must have at least k registers.

B. Weighted Automata

A weighted automaton [12] over an input alphabet Σ is a *nondeterministic* finite-state automaton whose edges are labeled with input symbols in Σ and costs in domain $\mathbb{D}$. For an input string w, the automaton can have multiple accepting paths from its initial state to some accepting state. The semantics of the automaton is defined using two commutative and associative functions $\oplus$ and $\otimes$ such that $\otimes$ distributes over $\oplus$. 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 the accepting paths of the automaton over w. A weighted automaton is called *single valued* if each input string has at most one accepting path. Thus to interpret a single-valued weighted automaton, we need only the interpretation for $\otimes$:

Theorem 4 (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 Sketch: For every single-valued weighted automaton W, we can construct an SSTT U that constructs the cost term corresponding to the sum of costs along the accepting path. Even though the automaton W is nondeterministic, since it is single-valued, the SSTT can use regular-look-ahead to choose deterministically the next transition that contributes to the accepting path. The proof then relies on the closure under regular-look-ahead for SSTTs. In the other direction, consider a CRA M using increments. At every step, the weighted automaton needs to guess which register of the machine M will contribute to the final output. The states of the weighted automaton are state-register pairs (q, x). There is a transition of cost k and label a from (q, x) to (q', x') to simulate the update: $x' := \otimes(x, k)$ on the a-labeled transition from q to q'. $\square$

C. Decision Problems

The min-cost problem for a CRA M is to find a string w whose cost is the minimum, *i.e.*, $\min\{M(w) \mid w \in \Sigma^*\}$. For CRAs with increment interpreted over numerical domains (that support addition), we can solve the min-cost problem by reducing it to the classical shortest path computation using the translation from CRAs with increment to single-valued weighted automata. If the CRA has n states and k registers, the graph has $n \cdot k$ vertices:

Theorem 5 (Min-cost for CRAs with $+c$) *Given a CRA M over the cost model $(\mathbb{Q}, +c)$, computing $\min\{M(w) \mid w \in \Sigma^*\}$ is solvable in* PTIME.

Even though $\mathbb{F}(\mathbb{D}, \otimes c) = \mathbb{F}^c(\mathbb{D}, \otimes)$, the model with binary addition is more succinct. To solve the min-cost problem for copyless-CRAs over the cost model $(\mathbb{D}, \otimes)$, we can translate these to CRAs over $(\mathbb{D}, \otimes c)$. However, this causes a blow-up exponential in the number of registers: essentially each register of the CRA represents a subset of the registers in the original copyless-CRA. We can establish that the min-cost problem is NP-HARD by a simple reduction from 3SAT.

Theorem 6 (Min-cost for CRAs with $+$) *Given a copyless-CRA M over $(\mathbb{Q}, +)$ 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 $\min\{M(w) \mid w \in \Sigma^*\} \leq K$ is* NP-HARD.

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 the properties of systems of linear equations.

Theorem 7 (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 Sketch: We first take the product $M = M_1 \times M_2$. We want to check if along every path of the product, and for every final state (v_1, v_2), the equation $\mu_1(v_1) = \mu_2(v_2)$ holds. The algorithm propagates such an equation over the register values backwards along each transition: for an edge from u to v, 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. At every step of the back propagation, we maintain 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 [21], 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 variables (in our case $|X^1| + |X^2|$). $\square$

We can also establish the following results regarding decision problems for additive CRAs: (1) *Inclusion:* given two CRAs M_1 and M_2 over the cost model $(\mathbb{Q}, +c)$, deciding whether for all w, $M_1(w) \leq M_2(w)$ is solvable in PTIME; (2) *Range-Membership:* given a CRA M over the cost model $(\mathbb{Z}, +c)$ and $K \in \mathbb{Z}$, deciding whether there exists w such that

$M(w) = K$ is solvable in NLOGSPACE.

V. SEMIRING COST FUNCTIONS

In this section, we consider the cost models with two binary operations, $\oplus$ and $\otimes$, that impose a semiring structure. 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 with identity ∞, and $\otimes$ is arithmetic addition with identity 0. While choosing a grammar, we can restrict either or both of $\oplus$ and $\otimes$ to be "unary", *i.e.*, where one of the arguments of the operator is a constant symbol. To study the Min-Plus semiring, it makes sense to choose $\min$ to be binary, while addition to be unary (*i.e.* increment by a constant). Hence, we will focus on the grammar $G(\oplus, \otimes c)$, and the class $\mathbb{R}(\mathbb{D}, \oplus, \otimes c)$ of cost functions.

A. 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 $e := (c, d) \mid x \mid e_1 \underline{\oplus} e_2 \mid e_1 \underline{\otimes} d \mid e_1 [e_2]$, 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 be used in a copyless manner, so that any particular register x appears in the update of at most one variable. 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 $(e_1.c \oplus e_1.d \otimes e_2.c, e_1.d \otimes e_2.d)$. The resulting model of CRA-definable cost functions is $\mathbb{F}^c(\mathbb{D} \times \mathbb{D}, \underline{\oplus}, \underline{\otimes} d, [\cdot])$.

We summarize the relationships between functions definable by CRAs over semiring cost models in the following theorem:

Theorem 8 (Expressiveness of Semiring Cost Functions) *If* $(\mathbb{D}, \oplus, \otimes)$ *forms a semiring, then* $\mathbb{F}^c(\mathbb{D}, \oplus, \otimes c) \subset \mathbb{F}^c(\mathbb{D} \times \mathbb{D}, \underline{\oplus}, \underline{\otimes} c, [\cdot]) \equiv \mathbb{R}(\mathbb{D}, \oplus, \otimes d) \subset \mathbb{F}(\mathbb{D}, \oplus, \otimes c)$.

Proof Sketch: The copyless-CRAs with operations of $\oplus$ and $\otimes c$ can be simulated by copyless-CRAs that maintain pairs with operations of $\underline{\oplus}$, $\underline{\otimes} c$, and $[\cdot]$. The correspondence between the terms constructed by SSTTs and their most evaluated forms maintained by CRAs as pairs leads to the equivalence of $\mathbb{F}^c(\mathbb{D} \times \mathbb{D}, \underline{\oplus}, \underline{\otimes} c, [\cdot])$ and $\mathbb{R}(\mathbb{D}, \oplus, \otimes d)$. Finally, we establish that this class is included in $\mathbb{F}(\mathbb{D}, \oplus, \otimes c)$. Consider a CRA M that maintain pairs using operations of $\underline{\oplus}$, $\underline{\otimes} c$, and $[\cdot]$. We perform a form of a subset construction over the registers to get a non-copyless CRA with $\oplus$ and $\otimes c$. Intuitively, the subset construction amounts to pre-emptively computing the result of substituting variable x into y. $\square$

B. Relation to Weighted Automata

In Section 4, we noted that single-valued weighted automata correspond exactly to CRAs with addition. 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 9 (Weighted Automata Expressiveness) *If* $(\mathbb{D}, \oplus, \otimes)$ *forms a semiring, then the class of functions* $\mathbb{F}(\mathbb{D}, \oplus, \otimes c)$ *exactly coincides with the class of functions computable by weighted automata.*

Proof Sketch: Consider a weighted automaton W with states P. The corresponding CRA M has state-set 2^P as in standard determinization. For each $p \in P$, the CRA M maintains a cost register x_p. The value of x_p after reading input w is the $\oplus$-sum over costs of all the paths from the initial state to the state p on the input w, where the cost of a path is the $\otimes$-sum of the weights of the transitions along the path. Given an input symbol a, the update of cost registers is given by: $x_q := \oplus\{x_p \otimes c \mid p \overset{a,c}{\to} q\}$. The correctness depends on the distributivity of $\otimes$ over $\oplus$. Note that the same register x_p contributes to all x_q's for all its a-successor states q's. Thus, the update is not necessarily copyless. In the reverse direction, consider a CRA M with states Q and registers X. The weighted automaton W has states $Q \times X$. Consider a transition from a state q to a state q' in M on an input symbol a with the update $x := y \otimes c$. Then, the automaton W has an edge from (q, y) to (q', x) labeled with the input symbol a and weight c. Consider a transition from a state q to a state q' in M on an input symbol a with the update $x := \oplus(y, z)$. Then, the automaton W has an edge from (q, y) to (q', x) labeled with the input a symbol and weight 0, and from (q, z) to (q', x) labeled with input a and weight 0. $\square$

C. Decision Problems for Min-Plus Models

Now we turn our attention to semirings in which the cost domain is a numerical domain such as $\mathbb{N}$, $\oplus$ is the minimum operation, and $\otimes$ is the addition. Given a CRA M over $(\mathbb{Q}, \min, +c)$, we can construct an equivalent weighted automaton linear in the size of M in linear time. The min-cost problem for weighted automata can be solved in polynomial-time using standard algorithms.

Theorem 10 (Min-cost for CRAs over min and $+c$) *Given a CRA M over the cost model $(\mathbb{Q}, \min, +c)$, 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 $(\mathbb{N}, \min, +c)$ 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 $(\mathbb{N}, \min, +c)$, and also for the class $\mathbb{R}(\mathbb{N}, \min, +c)$ is decidable.

VI. Discounted Costs

We now analyze the class of regular cost functions definable using $+c$ and $*d$.

A. 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 cost c. We call such machines *past-discount CRAs* (an example is M_4 of Figure 1). Thanks to the use of multiple registers this class of cost functions is closed under regular choice and regular look-ahead: the discount factors can depend conditionally upon future events. Our main result for past-discount CRAs is that the min-cost problem can be solved in polynomial-time.

Theorem 11 (Min-Cost 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.

B. Future Discounts

Symmetric to past discounts are future discounts: at every step, the machine wants to pay an additional cost c, and discount all future costs by a factor d. While processing an input $w_1 \dots w_n$, if the sequence of costs is $c_1, \dots c_n$ and the sequence of discount factors is $d_1, \dots d_n$, 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 discounted-sum operator $\underline{+}$: 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 $e := (c, d) \mid e \,\underline{+}\, (c, d) \mid x$. The interpretation for $e \,\underline{+}\, (c, d)$ is defined to be $(e.c + c * e.d, e.d * d)$ (that is, the discount factor $e.d$ is scaled by the new discount d, and the cost $e.c$ is updated by adding the new cost c, scaled by the 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 "reverse" paths, a single cost value updated using assignments of the form $x := d{*}x{+}c$ as in past-discount CRAs suffices.

Theorem 12 (Shortest Paths for Future Discounts) *For a future-discount CRA M over the cost model $(\mathbb{Q}, +c, *d)$, the problem of computing $\min\{M(w) \mid w \in \Sigma^*\}$ is solvable in* PTIME.

C. Global Discounts

A *global-discount CRA* is capable of scaling the global cost (the cost of the entire path) by a discount factor. Like for 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 $e := (0, 1) \mid (0, d) \,\underline{+}\, e \,\underline{+}\, (c, 1) \mid x$ (that is, the current discount factor $e.d$ is scaled by the new discount d, and the current cost $e.c$ is updated by first scaling it by the new discount, and then adding the new cost c scaled by the 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 13 (Min-cost 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 if $\min\{M(w) \mid w \in \Sigma^*\} \leq K$ is solvable in* NP. *Computing the minimum is solvable in* PTIME *if increments are restricted to natural numbers in unary encoding.*

Proof Sketch: We can reduce the min-cost problem to finding a shortest path in a graph G whose edges are labeled with (c, d) pairs, and the cost of a path containing edges labeled with $(c_1, d_1) \cdots (c_n, d_n)$ is $(\Sigma_i c_i)(\Pi_j d_j)$. First, observe that if there is a reachable cycle that contains an edge with discount factor < 1, then repeating this cycle drives the global discount to 0, and thus, existence of such a cycle implies that the (limiting) min-cost is 0. Since the shortest path need not involve a cycle in which all discount factors are equal to 1, the NP-bound follows. Suppose all costs c_i's are small natural numbers. The pseudo-polynomial algorithm for this case relies on the following: given a 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 is bound by nk for a graph with n vertices if each increment is a number smaller than k. $\Box$

D. 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 easy to show that:

Theorem 14 (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 $\mathbb{R}(\mathbb{D}, \oplus, \otimes d) \subset \mathbb{F}(\mathbb{D}, \oplus, \otimes c)$ (see Theorem 8), we can represent cost functions in $\mathbb{R}(\mathbb{Q}, +c, *d)$ using CRAs that use $+$ and $*d$. Such CRAs have *linear* updates, and the algorithm for checking equivalence of CRAs with addition can be adopted.

Theorem 15 (Equivalence of Inc-Scale Models) *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 equivalence can be solved in time polynomial in the number of states and exponential in the number of registers.*

VII. Related Work

Weighted Automata and Logics: Finite-state weighted automata have been an active area of research, with numerous articles studying their algebraic and algorithmic properties. See [12] for a comprehensive exposition. An important problem for WA is that of determinization ([12], [19]): it is known that there are weigthed automata with no equivalent deterministic ones.

It has been shown that the equivalence problem for weighted automata over the Min-Plus semiring is undecidable using a reduction from Hilbert's tenth problem [20], 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 is that of finite-valued weighted automata [25]. 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 CRAs over *min* and $+c$ with the copyless restriction is decidable.

In [11], 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.

Discounted Shortest Paths: Generalized network flow problems extend flow problems on directed graphs by specifying multipliers on edges in addition to costs ([15], [23]). The problem of finding the minimum cost flow (which in some cases is equivalent to the shortest path) from a source to a target can be solved in polynomial time ([23], [5]). 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 [13], 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.

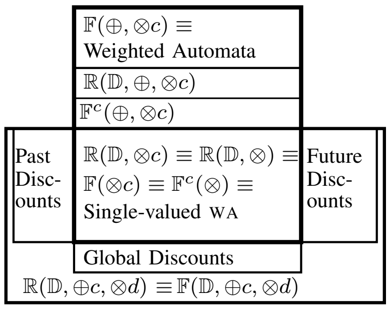
Transducer Models: A wide variety of different models have been proposed to model string and tree transductions. The models that are most relevant to this paper are MSO-definable transductions [10] and macro tree transducers [14]. An MSO-definable graph transduction specifies a function between sets 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 [14].

Streaming tree transducers [3] (STTs) 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 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 ([22], [17], [6]). 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. 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 [8], 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 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.

Affine Programs: In [18], and more recently in [21], 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


$\mathbb{F}(\oplus, \otimes c) \equiv$ Weighted Automata
$\mathbb{R}(\mathbb{D}, \oplus, \otimes c)$
$\mathbb{F}^c(\oplus, \otimes c)$
Past Discounts
$\mathbb{R}(\mathbb{D}, \otimes c) \equiv \mathbb{R}(\mathbb{D}, \otimes) \equiv \mathbb{F}(\otimes c) \equiv \mathbb{F}^c(\otimes) \equiv$ Single-valued WA
Future Discounts
Global Discounts
$\mathbb{R}(\mathbb{D}, \oplus c, \otimes d) \equiv \mathbb{F}(\mathbb{D}, \oplus c, \otimes d)$

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

Fig. 3. Summary of Results

$\mathbb{C}(\mathbb{Q}, +, *d)$, with the cost grammar $t := +(t,t) \mid *(t,d) \mid c$. While the cost functions defined by such CRAs do not have interesting regularity properties, we remark that the equivalence of such CRAs can be checked in polynomial time by using ideas similar to the ones in [21].

Quantitative Languages: A quantitative language [7], [1], [4] 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.

VIII. 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 3. 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 minimum 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. Streaming Transducers for Algorithmic Verification of Single-pass List-processing Programs. In *Proc. of Principles of Programming Languages*, pages 599–610, 2011.

[3] 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.

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

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

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

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

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

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

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

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

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

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

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

[15] 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.

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

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

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

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

[20] 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.

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

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

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

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

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