Better Complexity Bounds for Cost Register Automata*

Eric Allender, Department of Computer Science, Rutgers University
Andreas Krebs, WSI, Universität Tübingen
Pierre McKenzie Université de Montréal, Québec

May 9, 2018

Abstract

Cost register automata (CRAs) are one-way finite automata whose transitions have the side effect that a register is set to the result of applying a state-dependent semiring operation to a pair of registers. Here it is shown that CRAs over the tropical semiring $(\mathbb{N} \cup \{\infty\}, \min, +)$ can simulate polynomial time computation, proving along the way that a naturally defined width-k circuit value problem over the tropical semiring is P-complete. Then the copyless variant of the CRA, requiring that semiring operations be applied to distinct registers, is shown no more powerful than NC^1 when the semiring is $(\mathbb{Z}, +, \times)$ or $(\Gamma^* \cup \{\perp\}, \max, \text{concat})$. This relates questions left open in recent work on the complexity of CRA-computable functions to long-standing class separation conjectures in complexity theory, such as NC versus P and NC^1 versus GapNC^1.

1 Introduction

In this paper, we study *cost register automata*, the study of which is motivated in part by connections to formal verification [6] and by the relation between this model and the well-studied notion of weighted automata.

A *weighted finite automaton* on a given input computes the sum, over every computation path, of the product, over the transitions encountered along that path, of the semiring elements assigned to those transitions. Weighted automata have a long history and extensive theoretical support (see [19]) but their utility for the purpose of computer-aided verification is limited. This motivated Alur and his co-authors to introduce the streaming string transducer [4], soon followed by the cost register automaton (CRA) [6].

CRAs are deterministic and are yet strictly more expressive than weighted automata [6]. A CRA computes a so-called *regular function* from strings to a cost domain. (This should not be confused with Colcombet's regular cost functions, which are intended to capture asymptotic behavior [15].) A "copyless"

*Supported by NSF grants CCF-1555409 and CCF-1514164 (Allender), by the DFG Emmy-Noether program KR 4042/2 (Krebs), and by the Natural Sciences and Engineering Research Council of Canada (McKenzie). An earlier version of this work appeared as [3]. The proof of Lemma 5 in this expanded version corrects a construction in the proof of Theorem 4 of [3].

variant (CCRA) of the CRA has the expressiveness of single-valued weighted automata [6, Thm 4]. (See also Theorem 7 of [5].) Another variant of CRAs restricts the multiplicative operation, by only allowing multiplication by constants; this model has the full expressivity of weighted automata [6, Thm 9]. A theory of CRAs, largely concerned with expressivity and decidability properties, was developed in a series of papers, including [6, 8, 7].

None of the above work considered the computational complexity of the functions expressed by CRAs. Yet the CRA model is interesting from that viewpoint because it combines a parallelizable component (logarithmic depth boolean circuits indeed recognize regular languages) with a less structured component that builds and evaluates expressions over the cost domain. The three variants of the CRA discussed above (CRAs, CCRAs, and CRAs with restricted multiplication) in fact are reminiscent of three variants of algebraic circuits (general, tree-like, and skew). This raises the question of whether CRA variants over various domains capture interesting complexity classes, such as the (functional) class P and subclasses of NC. These considerations prompted Allender and Mertz to develop complexity bounds for the functions computable by CRAs and CCRAs [1]. This line of inquiry for various models was also pursued and extended in [24, 28, 27, 18, 17, 14]. ‘ The main results obtained by Allender and Mertz are depicted on Figure 1. Most results involve the (weaker) CCRA model with integer arithmetic, but also with "tropical" arithmetic, that is, over domains such as $(\mathbb{N}\cup\{\infty\}, \min, +)$ and $(\Gamma^*, \max, \circ)$. We note that tropical semirings arise frequently in the study of weighted automata (see for instance [16, Sect. 1.1] and [6, Thm 9]). Computationally, min and max are "forgetful" operations that should intuitively lend themselves to simpler simulations.

Our contribution here is to improve some of the bounds from [1]. In particular, the closing section of [1] listed the following four open questions:

- Are there any CCRA functions over $(\mathbb{Z}, +, \times)$ that are complete for GapNC^1?
- Are there any CCRA functions over the tropical semiring that are hard for $\#\mathsf{NC}^1_{\mathrm{trop}}$?
- The gap between the upper and lower bounds for CCRA functions over $(\Gamma^*, \max, \circ)$ is quite large (NC^1 versus $\mathsf{OptLogCFL} \subseteq \mathsf{AC}^1$). Can this be improved?
- Is there an NC upper bound for CRA functions (without the copyless restriction) over the tropical semiring?

We essentially answer all of these questions, modulo long-standing open questions in complexity theory. We show that CCRA functions over each of $(\mathbb{Z}, +, \times)$, $(\Gamma^*, \max, \circ)$, and the tropical semiring are all computable in NC^1. We thus give the improvement asked for in the third question, and we show that the answers to the first two questions are equivalent to $\mathsf{NC}^1 = \mathsf{GapNC}^1$ and $\mathsf{NC}^1 = \#\mathsf{NC}^1_{\mathrm{trop}}$, respectively. We also provide a negative answer to the fourth question (assuming $\mathsf{NC} \neq \mathsf{P}$), by reducing a P-complete problem to the computation of a CRA function over the tropical semiring. It follows from the latter that for any k larger than a small constant, the width-k circuit value problem over structures such as $(\mathbb{N}, \max, +)$ and $(\mathbb{N}, \min, +)$ is P-complete under AC^0-Turing reductions. (See Section 2 for the precise definition of the problem and then Corollary 9.) Figure 2 summarizes our results.

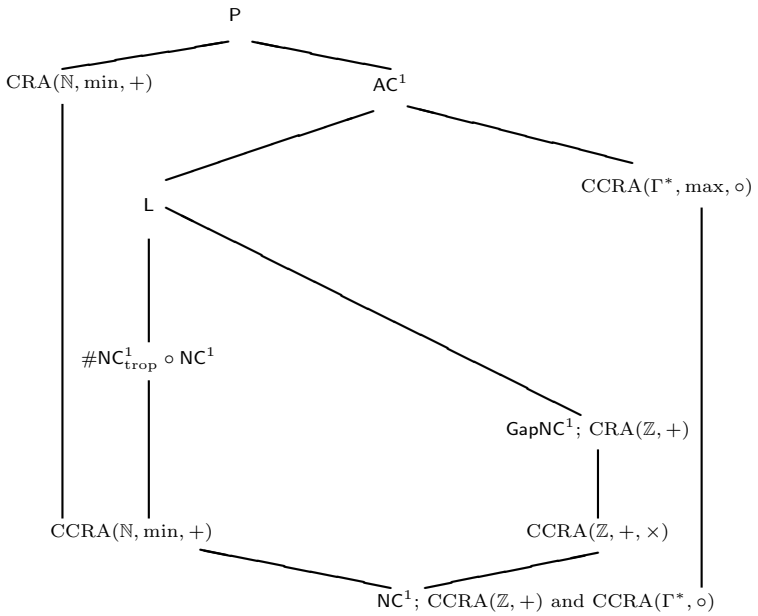

Figure 1: Prior state of knowledge, from [1]. When a class of CRA functions and a complexity class appear together, it means that containment of the CRA class in the complexity class is tight, since some of the CRA functions are complete for the complexity class. (Definitions of the complexity classes can be found in Section 2.)

2 Preliminaries

We assume familiarity with some common complexity classes and with basic notions of circuit complexity, such as can be found in any textbook on complexity theory, such as [32, 9].

Recall that a language $A \subseteq \{0,1\}^*$ is accepted by a Boolean circuit family $(C_n)_{\in\mathbb{N}}$ if for all x it holds that $x \in A$ iff $C_{|x|}(x) = 1$. Circuit families encountered in this paper will be *uniform*. Uniformity is a somewhat technical issue because of subtleties encountered at low complexity levels. We will not be concerned with such subtleties, and thus we refer the reader to a standard text (such as [32, Sect. 4.5]) for a precise definition of what it means for a circuit family $(C_n)_{n\geq 0}$ to be U_E-uniform. (Informally, this notion of uniformity means that there is a linear-time machine that takes inputs of the form (n, g, h, p) and determines if p encodes a path from gate h to gate g in C_n, and also determines what type of gate g and h are.) We will encounter the following circuit complexity classes.

- $\mathsf{NC}^i = \{A : A$ is accepted by a U_E-uniform family of circuits of bounded fan-in AND, OR and NOT gates, having size $n^{O(1)}$ and depth $O(\log^i n)\}$.
- $\mathsf{AC}^i = \{A : A$ is accepted by a U_E-uniform family of circuits of unbounded fan-in AND, OR and NOT gates, having size $n^{O(1)}$ and depth $O(\log^i n)\}$.
- $\mathsf{TC}^i = \{A : A$ is accepted by a U_E-uniform family of circuits of unbounded fan-in MAJORITY gates, having size $n^{O(1)}$ and depth $O(\log^i n)\}$.

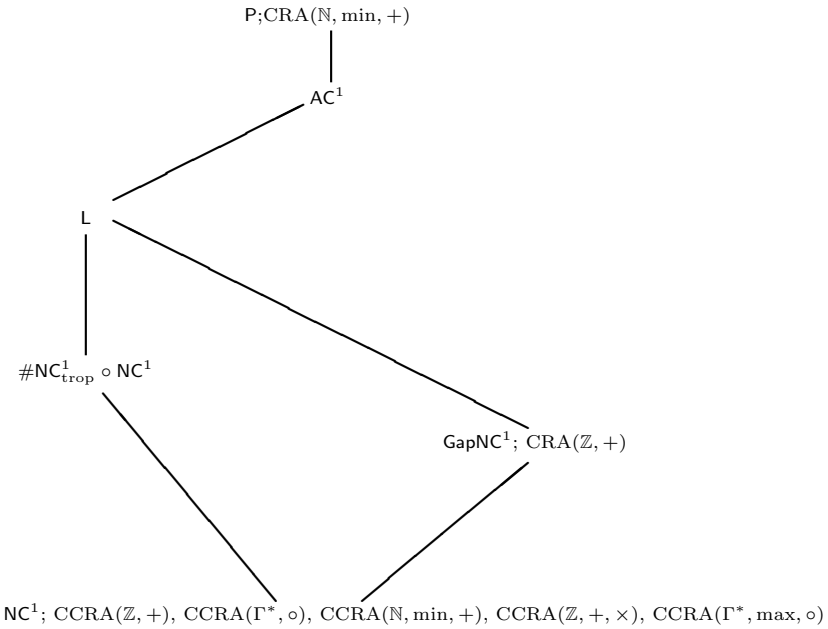

Figure 2: Update of the preceding figure, showing the improved state of our knowledge regarding $\mathrm{CRA}(\mathbb{N}, \min, +)$ and the copyless CRA classes $\mathrm{CCRA}(\mathbb{N}, \min, +)$, $\mathrm{CCRA}(\mathbb{Z}, +, \times)$, and $\mathrm{CCRA}(\Gamma^*, \max, \circ)$. All bounds listed are now tight.

We remark that, for constant-depth classes such as AC^0 and TC^0, U_E-uniformity coincides with U_D-uniformity, which is also frequently called DLOGTIME-uniformity. (Again, we refer the reader to [32] for more details on uniformity.)

Following the standard convention, we also use these same names to refer to the associated classes of *functions* computed by the corresponding classes of circuits. For instance, a function $f : \{0,1\}^* \to \{0,1\}^*$ is said to be in NC^1 if there is U_E-uniform family of circuits $\{C_n\}$ of bounded fan-in AND, OR and NOT gates, having size $n^{O(1)}$ and depth $O(\log n)$, where C_n has several output gates, and on input x of length n, C_n outputs an encoding of $f(x)$. (We say that an "encoding" of the output is produced, to allow the possibility that there are strings x and y of length n, such that $f(x)$ and $f(y)$ have different lengths.) It is easy to observe that, if the length of $f(x)$ is polynomial in $|x|$, then f is in NC^1 if and only if the language $L_f = \{(x, i, b) : \text{the } i\text{-th symbol of } f(x) \text{ is } b\}$ is in NC^1. Similar observations hold for other classes. A language L is complete for a complexity class $\mathcal{C}$ under AC^0 many-one reductions if $L \in \mathcal{C}$ and, for every language $B \in \mathcal{C}$, there is a function f in AC^0 such that, for all x, $x \in B$ if and only if $f(x) \in L$. We also need to make use of a more general notion of reducibility; A language L is complete for $\mathcal{C}$ under AC^0 Turing reductions if $L \in \mathcal{C}$ and, for every language $B \in \mathcal{C}$, there is a uniform family of polynomial-size, constant-depth circuits computing B, where the circuits consist of unbounded fan-in AND, OR and NOT gates, as well as *oracle gates* for the language L. An oracle gate has some number k of input wires, and outputs the value 1 iff the string y encoded on the input wires lies in L. In particular, if the language L_f is complete for P under AC^0 Turing reductions, then we say that f is complete for P under AC^0 Turing reductions. (Equivalently, every language

in P can be solved by AC^0 circuits with oracle gates for f.)

A structure $(\mathcal{A}, +, \times)$ is a semiring if $(\mathcal{A}, +)$ is a commutative monoid with an additive identity element 0, and $(\mathcal{A}, \times)$ is a (not necessarily commutative) monoid with a multiplicative identity element 1, such that, for all a, b, c, we have $a \times (b+c) = (a \times b) + (a \times c)$, $(b+c) \times a = (ba \times ca)$, and $0 \times a = a \times 0 = 0$.

Definition 1 *An* arithmetic circuit *over a semiring $(R, +, \times)$ is a directed acyclic graph. Each vertex of the graph is called a "gate"; each gate is labeled with a "type" from the set $\{+, \times$, input, constant$\}$, where each input gate is labeled by one of the inputs $x_1, \ldots, x_n$, and each constant gate is labeled with an element of R. (Input and constant gates have indegree zero.) There is a unique sink called the "output gate". The* size *of a circuit is the number of gates, and the* depth *of the circuit is the length of the longest path in the circuit. We shall also need to refer to the* width *of a circuit, and here we use the notion of circuit width that was provided by Pippenger [30]: which requires the notion of* layered *circuits. In layered circuits, the gates other than inputs and constants can be partitioned into layers $1, 2, \ldots$ in such a way that every wire out of a gate in a layer $i \geq 1$ connects to a gate in layer $i+1$. The width of a circuit is the largest number of gates that occurs in any such layer.*

If an arithmetic circuit C_n over $(R, +, \times)$ has n input gates, then C_n computes a function $f : R^n \to R$ in the obvious way.

Definition 2 *A* straight-line program *over a structure $(R, \mathtt{op}_1, \ldots, \mathtt{op}_s)$, also called a $\{\mathtt{op}_1, \ldots, \mathtt{op}_s\}$-SLP over R, with registers $\{r_1, \ldots, r_k\}$ consists of a sequence of statements of the form $r_i \leftarrow r_j \odot r_k$ or $r_i \leftarrow r_j$ where $\odot \in \{\mathtt{op}_1, \ldots, \mathtt{op}_s\}$, r_i is a register, r_j and r_k are either a register, a value from R, or from the set of input variables. Straight-line programs have been studied at least as far back as [25], and they are frequently used as an alternative formulation of arithmetic circuits. Note that each line in a straight-line program can be viewed as a gate in an arithmetic circuit.*

Definition 3 *The k-*register SLP evaluation problem *over a semiring $(R, +, \times)$ takes as input an encoding of a SLP with k registers over the semiring and a pair (i, b), and determines whether the i-th bit of the binary representation of the output of the SLP is b.*

Definition 4 *The* width-k circuit value problem *over a semiring $(R, +, \times)$ is that of determining, given a width-k arithmetic circuit C over the semiring (where C has no input gates, and hence all gates with indegree zero are labeled by a constant in R), and given a pair (i, b) whether the i-th bit of the binary representation of the output of C is b.*

- $\#\mathsf{NC}^1{}_S$ is the class of functions $f : \bigcup_n R^n \to R$ for which there is a U_E-uniform family of arithmetic circuits $\{C_n\}$ of logarithmic depth, such that C_n computes f on R^n.
- By convention, when there is no subscript, $\#\mathsf{NC}^1$ denotes $\#\mathsf{NC}^1{}_{(\mathbb{N},+,\times)}$, with the additional restriction that the functions in $\#\mathsf{NC}^1$ are considered to have domain $\bigcup_n \{0,1\}^n$. That is, we restrict the inputs to the Boolean domain. (Boolean negation is also allowed at the input gates.)

- GapNC^1 is defined as $\#\mathsf{NC}^1 - \#\mathsf{NC}^1$; that is: the class of all functions that can be expressed as the difference of two $\#\mathsf{NC}^1$ functions. It is the same as $\#\mathsf{NC}^1{}_{\mathbb{Z}}$ restricted to the Boolean domain. See [32, 2] for more on $\#\mathsf{NC}^1$ and GapNC^1.

The following inclusions are known:

$$\mathsf{NC}^0 \subseteq \mathsf{AC}^0 \subseteq \mathsf{TC}^0 \subseteq \mathsf{NC}^1 \subseteq \#\mathsf{NC}^1 \subseteq \mathsf{GapNC}^1 \subseteq \mathsf{L} \subseteq \mathsf{AC}^1 \subseteq \mathsf{P}.$$

All inclusions are straightforward, except for $\mathsf{GapNC}^1 \subseteq \mathsf{L}$ [21].

2.1 Cost-register automata

A *cost-register automaton* (CRA) is a deterministic finite automaton (with a read-once input tape) augmented with a fixed finite set of *registers* that store elements of some algebraic domain $\mathcal{A}$. At each step in its computation, the machine

- consumes the next input symbol (call it a),
- based on a and the current state (call it q), moves to a new state,
- based on q and a, updates each register r_i using updates of the form $r_i \leftarrow f(r_1, r_2, \ldots, r_k)$, where f is an expression built using the registers $r_1, \ldots, r_k$ using the operations of the algebra $\mathcal{A}$.

There is also an "output" function μ defined on the set of states; μ is a partial function – it is possible for $\mu(q)$ to be undefined. Otherwise, if $\mu(q)$ is defined, then $\mu(q)$ is some expression of the form $f(r_1, r_2, \ldots, r_k)$, and the output of the CRA on input x is $\mu(q)$ if the computation ends with the machine in state q.

An example is provided in Figure 3.

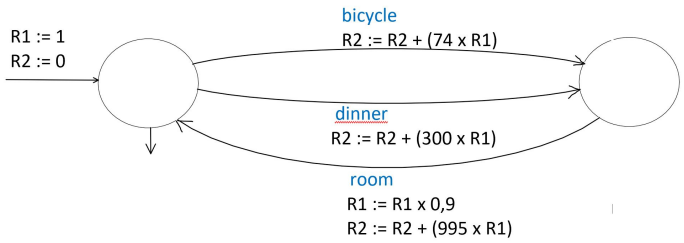

Figure 3: An example of a 2-state CRA over the 3-letter alphabet {room,dinner,bicycle}, modeling the price of transportation, food, and lodging, where the rental on a bicycle is 74 DKK for one day, dinner is 300 DKK, and lodging is 995 DKK, but the price for each successive day (for each of these items) decreases by 10% each day (with the further restriction that, on each day one must choose exactly one of (a) dinner, and (b) an all-day bicycle rental). Register 1 stores the current rate, and Register 2 stores the total expense thus far. For each state q, the output $\mu(q)$ is the value stored in Register 2.

More formally, here is the definition as presented by Alur *et al.* [6].

A cost-register automaton M 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*.
- $\delta : Q \times \Sigma \to Q$ is the state-transition function.
- $\rho : Q \times \Sigma \times X \to E$ is the register update function (where E is a set of algebraic expressions over the domain $\mathcal{A}$ and variable names for the registers in X).
- $\mu : Q \to E$ is a (partial) final cost function.

A *configuration* of a CRA is a pair (q, ν), where ν maps each element of X to an algebraic expression over $\mathcal{A}$. The *initial configuration* is (q_0, ν_0), where ν_0 assigns the value 0 to each register (or some other "default" element of the underlying algebra). Given a string $w = a_1 \dots a_n$, the *run* of M on w is the sequence of configurations $(q_0, \nu_0), \dots (q_n, \nu_n)$ such that, for each $i \in \{1, \dots, n\}$ $\delta(q_{i-1}, a_i) = q_i$ and, for each $x \in X$, $\nu_i(x)$ is the result of composing the expression $\rho(q_{i-1}, a_i, x)$ to the expressions in ν_{i-1} (by substituting in the expression $\nu_{i-1}(y)$ for each occurrence of the variable $y \in X$ in $\rho(q_{i-1}, a_i, x)$). The output of M on w is undefined if $\mu(q_n)$ is undefined. Otherwise, it is the result of evaluating the expression $\mu(q_n)$ (by substituting in the expression $\nu_n(y)$ for each occurrence of the variable $y \in X$ in $\mu(q_n)$).

We denote the class of functions computed by CRAs operating over algebra $\mathcal{A}$ by CRA($\mathcal{A}$).

It is frequently useful to restrict the algebraic expressions that are allowed to appear in the transition function $\rho : Q \times \Sigma \times X \to E$. One restriction that is important in previous work [6] is the "copyless" restriction.

A CRA is *copyless* if, for every register $r \in X$, for each $q \in Q$ and each $a \in \Sigma$, the variable "r" appears at most once in the multiset $\{\rho(q, a, s) : s \in X\}$. In other words, for a given transition, no register can be used more than once in computing the new values for the registers. Following [7], we refer to copyless CRAs as CCRAs. Over many algebras, unless the copyless restriction is imposed, CRAs compute functions that can not be computed in polynomial time. For instance, CRAs that can concatenate string-valued registers and CRAs that can multiply integer-valued registers can perform "repeated squaring" and thereby obtain results that require exponentially-many symbols to write down.

We denote the class of functions computed by CCRAs operating over algebra $\mathcal{A}$ by CCRA($\mathcal{A}$).

The CRA in Figure 3 is not copyless, since the transition on the symbol "room" uses R_1 in the updates of both R_1 and R_2. See Figure 4.

3 CRAs over the Tropical Semiring

CRAs *without* the copyless restriction over some algebras (such as the "tropical semiring" defined in the next paragraph) still yield only functions that are computable in polynomial time. The "repeated squaring" operation, when the

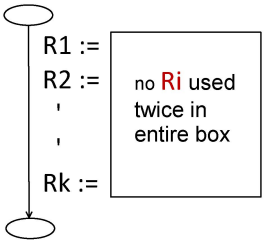

Figure 4: A visual aid for the notion of a Copyless CRA.

"multiplicative" operation is $+$, yields only numbers whose binary representation remains linear in the length of the input. In this section, we show that some CRA functions over the tropical semiring are hard for P.

The name "tropical semiring" is used to refer to several related algebras. Most often it refers to $(\mathbb{R} \cup \{\infty\}, \min, +)$ (that is, the "additive" operation is min, and the "multiplicative" operation is $+$. However, frequently $(\mathbb{R} \cup \{-\infty\}, \max, +)$ is used instead. In discrete applications, $\mathbb{R}$ is frequently replaced with $\mathbb{Q}$, $\mathbb{Z}$, or even $\mathbb{N}$. For more details, we refer the reader to [29]. For a discussion of the relationship between min and max in the tropical semiring, see [26]. We will not need to make any use of ∞ or $-\infty$ in our hardness argument, and we will prove P-hardness over $\mathbb{N}$, which thus implies hardness for the other settings as well. Our arguments will be slightly different for both the max and the min versions, and thus we will consider both.

The standard reference for P-completeness, [20], credits Venkateswaran with the proof that the Min-plus Circuit Value Problem is P-complete. This shows that evaluating straight-line programs over $(\mathbb{N}, \min, +)$ is a P-complete problem, as long as there is no restriction on the number of registers in the SLP.

Our focus will be more on straight-line programs with a *bounded* number of registers. That is, we focus on the k-register SLP evaluation problem (Definition 3). Ben-Or and Cleve [13] showed that straight-line programs with $O(1)$ registers can simulate arithmetic formulae, and Koucky [23] has shown that these models are in fact *equivalent*, if the straight-line programs are restricted to compute only formal polynomials whose degree is bounded by a polynomial in the number of variables. (More precisely, a family of multivariate polynomials $\{p_n(x_1, \ldots, x_n) : n \in \mathbb{N}\}$ where p_n has algebraic degree $n^{O(1)}$, is computed by a family of polynomial-size arithmetic formulae $\{F_n : n \in \mathbb{N}\}$ if and only if it is computed by a family of polynomial-size SLPs $\{P_n : n \in \mathbb{N}\}$ having $O(1)$ registers.) It is observed in [1] that arithmetic formulae (that is, straight-line programs with $O(1)$ registers *and a polynomial degree restriction*) over the tropical semiring can be evaluated in logspace. Our P-completeness result demonstrates that, in the *absence* of any degree restriction, restricting straight-line programs over the tropical semiring to have only $O(1)$ registers yields a model that is as powerful as having an unlimited number of registers.

We will need a strategy to compute the remainder of an integer division. Define

$$\mathit{modified\_subtraction}(a, b) = \begin{cases} a - b & \text{if } a \geq b, \\ a & \text{otherwise.} \end{cases}$$

Lemma 5 *Let $m \in \mathbb{N}^+, v \in \mathbb{N}, c \in \mathbb{N}$. If $v < m \cdot 2^c$, then the following algorithm computes $v \bmod m$, i.e., returns $x \in [0..m-1]$ such that $x \equiv v \pmod m$:*

$x \leftarrow v$
for $i = c-1, c-2, \ldots, 0$ ***do*** $x \leftarrow$ *modified_subtraction*$(x, m * 2^i)$ ***end for***
return x.

Proof: Fix $m \geq 1$. We show by induction on c that for any $v \in \mathbb{N}$ such that $v < m \cdot 2^c$, the algorithm returns $v \bmod m$. When $c = 0$, no iteration is performed so the algorithm correctly returns $x = v$. Now let $c > 0$. Pick $v \in \mathbb{N}$ such that $v < m \cdot 2^c$ and let $r \in [0..m-1]$ stand for $v \bmod m$. For some $d \in \{0, 1\}$ and $q \in [0..2^{c-1} - 1]$, we can write

$$\begin{aligned} v &= d \cdot 2^{c-1} \cdot m + q \cdot m + r \\ &\leq d \cdot 2^{c-1} \cdot m + (2^{c-1} - 1) \cdot m + r \\ &< d \cdot 2^{c-1} \cdot m + 2^{c-1} \cdot m. \end{aligned}$$

Suppose that $d = 0$. Then $v < 2^{c-1} \cdot m$, so the first iteration in the algorithm leaves x equal to v. At that point $x \bmod m$ has not changed and the induction hypothesis kicks in. Now suppose instead that $d = 1$. Then $v \geq 2^{c-1} \cdot m$, so the first iteration subtracts $m * 2^{c-1}$ from x, leaving the new x equal to $v - 2^{c-1} \cdot m \equiv r \pmod m$ and fulfilling $x < m \cdot 2^{c-1}$. The induction again kicks in, completing the proof. ∎

We will also need to implement *modified_subtraction*(a, b) without a positivity test:

Lemma 6 *Let $a \in \mathbb{N}, b \in \mathbb{N}, c \in \mathbb{N}$ such that $b < 2^c$. Then*

$$\textit{modified\_subtraction}(a, b) = a + \max(-b, -\max(0, a - b + 1) \cdot 2^c).$$

Proof: If $a < b$ then $\max(0, a - b + 1) = 0$, so the outer max is also 0 and the expression evaluates to a. If $a \geq b$ then $\max(0, a - b + 1) \cdot 2^c \geq 2^c$; since $-b > -2^c$, the outer max contributes $-b$ and the expression correctly evaluates to $a - b$. ∎

The following will serve as central building block in our construction of a CRA computing a P-complete function:

Lemma 7 *There is an AC^0 circuit family able to produce, given $v, m \in \mathbb{N}$ in binary, a $\{\max, +, -\}$-straight line program over $\mathbb{Z}$ that has no input variables, that has v loaded into register r_1 and m loaded into register r_4, that uses only 4 registers and only the constants 0 and 1, and that computes $v \bmod m$ into r_1.*

Proof: Let c equal the input length. The SLP is as follows, where for readability we write m for r_4:

$r_2 \leftarrow m * 2^{c-1}$ {stands for $r_2 \leftarrow m$ followed by $c-1$ occurrences of $r_2 \leftarrow r_2 + r_2$}
$r_1 \leftarrow$ *modified_subtraction*(r_1, r_2)
$r_2 \leftarrow m * 2^{c-2}$
$r_1 \leftarrow$ *modified_subtraction*(r_1, r_2)
$\vdots$

$r_2 \leftarrow m * 2^0$
$r_1 \leftarrow$ *modified_subtraction*(r_1, r_2).

Each line $r_1 \leftarrow$ *modified_subtraction*(r_1, r_2) stands for the sequence

$r_3 \leftarrow r_1 - r_2$; $r_3 \leftarrow r_3 + 1$; $r_3 \leftarrow \max(0, r_3)$; $r_3 \leftarrow 0 - r_3$
$r_3 \leftarrow r_3 * 2^{2c}$ {stands for $2c$ occurrences of $r_3 \leftarrow r_3 + r_3$; note that $r_2 < m \cdot 2^c < 2^{2c}$}
$r_2 \leftarrow 0 - r_2$; $r_2 \leftarrow \max(r_2, r_3)$; $r_1 \leftarrow r_1 + r_2$

implementing $r_1 \leftarrow r_1 + \max(-r_2, -\max(0, r_1 - r_2 + 1) \cdot 2^{2c})$. The complete construction can be done in AC^0, since the first list of instructions involving r_1 and r_2 are easily generated by substituting the numbers $c-1, c-2, \ldots, 0$ into the template, and then the final result is a simple syntactic substitution (replacing *modified_subtraction*). The result is a $\{\max, +, -\}$-SLP over $\mathbb{Z}$ having the desired properties and, by Lemma 5 and Lemma 6, leaving $v \bmod m$ in its register r_1. ∎

Theorem 8 *There is a function f computable by a CRA operating over the tropical semiring (either $(\mathbb{N} \cup \{\infty\}, \min, +)$ or $(\mathbb{N} \cup \{-\infty\}, \max, +)$) such that computing f is hard for* P *under* AC^0*-Turing reductions.*

Proof: We will present a reduction from the P-complete problem Iterated Mod (problem A.8.5 in [20]), which was shown to be P-complete under logspace reductions by Karloff and Ruzzo [22]. The proof in [22] actually shows that the problem is complete under many-one reductions computable by dlogtime-uniform AC^0 circuits. (Incidentally, the proof sketch in [20] has a minor error, in that some indices are listed in the wrong order. The reader is advised to consult the original [22] proof.)

The input to the Iterated Mod problem is a list of natural numbers $v, m_1, m_2, \ldots, m_n$ (encoded in binary), and the question is to determine if $((\cdots((v \bmod m_1) \bmod m_2)\cdots) \bmod m_n) = 0$.

Our reduction will first show how to construct (in AC^0) an encoding of a SLP with 6 registers such that $y \in$ Iterated Mod if and only if two designated registers contain the same value.

Given an instance y of Iterated Mod, our AC^0 reduction will first construct straight-line program Q with 6 registers (one of which is a special register denoted r_0), having no input variables, such that $y \in$ Iterated Mod iff the output register of Q has a value equal to the value of register r_0. (We will be able to carry out this construction over either $(\mathbb{N}, \min, +)$ or $(\mathbb{N}, \max, +)$.) This SLP Q has the further property the number of constants it uses is independent of y. Hence Q is built from a constant size set Σ of distinct instructions.

Now observe that there is a CRA that takes as input any string over Σ and simulates the operation of the encoded straight-line program (where the length of the string is equal to the number of lines in the SLP). The function f that is computed by this CRA is the function whose existence is asserted in the statement of the theorem. That is, Iterated Mod can be solved by making two oracle queries to f, one of which simulates Q and gives the value of the output register, and one of which simulates Q and gives the value of r_0.

We now show how to construct the SLP Q.

First note that an AC^0 circuit family on input $v, m_1, m_2, \ldots, m_n$ can output a $\{\max, +, -\}$-SLP that computes $((\cdots((v \bmod m_1) \bmod m_2)\cdots) \bmod m_n)$ as follows:

{load r_1 with v and r_4 with m_1}
$r_1 \leftarrow r_1 \bmod r_4$ {stands for the SLP provided by Lemma 7}
{load r_4 with m_2}
$r_1 \leftarrow r_1 \bmod r_4$
$\vdots$
{load r_4 with m_n}
$r_1 \leftarrow r_1 \bmod r_4$.

To achieve the above loading of a register r with a binary number such as $v = v_t v_{t-1} \cdots v_0$, the AC^0 circuit actually emits the instruction $r \leftarrow v_0$ followed for $i = 1, \ldots, t$ by the pair $r \leftarrow r + r$; $r \leftarrow r + v_i$. The final $\{\max, +, -\}$-SLP over $\mathbb{Z}$ constructed further satisfies the properties that it uses only 4 registers and only the constants 0 and 1.

The SLP produced is not yet over the desired tropical semirings. We will reach this goal in 3 steps: max will be shown replaceable with min, negative numbers will be shown avoidable and subtractions will be shown unnecessary.

Trading max *for* min. We observe that $\max(a, b) = (-1) \cdot \min(-a, -b)$. Thus a further AC^0 transformation can convert our $\{\max, +, -\}$-SLP into a $\{\min, +, -\}$-SLP over $\mathbb{Z}$ that outputs 0 if and only if $(v, m_1, \ldots, m_n)$ is a positive instance of Iterated Mod: the new SLP is just the old one with every occurrence of max replaced with min and every occurrence of the constant 1 in any instruction replaced with the constant -1. An induction shows that the resulting $\{\min, +, -\}$-SLP over $\mathbb{Z}$ computes the negative of the value computed by the old SLP. The new SLP uses only the constants -1 and 0, and only 4 registers.

Getting rid of negatives. Using in each case two extra registers assigned to constants from $\{-1, 0, 1\}$, we first ensure that every $+$, $-$, max or min operation in our $\{\max, +, -\}$-SLP or $\{\min, +, -\}$-SLP Q is performed on operands that are registers. In AC^0 we will further transform Q into Q' such that each register of Q' always holds a nonnegative integer, and such that the value of each register r of Q at the end of the computation is equal to the value of the difference $r - r_0$ of Q' at the end, where r_0 is a new special register of Q'. We accomplish this by initially setting r_0 to 2^c (using $r_0 \leftarrow 1$ and repeated addition), where 2^c is larger than any value that is stored by any register of Q during its computation. (This is possible by taking c to be larger than the length of Q.) Then for every other register $r \neq r_0$, we perform the operation $r \leftarrow r_0$. Now we will maintain the invariant that the value of register r of Q is obtained by subtracting r_0 from the value of register r of Q'. This is accomplished as follows: Replace any assignment $r \leftarrow b$ where b is a constant, with $r \leftarrow r_0 + b$. Leave any instruction involving max or min untouched, insert $r \leftarrow r + r_0$ after each operation $r \leftarrow s - u$, and replace each operation $r \leftarrow s + u$ by the operations: $r \leftarrow s + u; r' \leftarrow r' + r_0$ (for every $r' \neq r$); $r_0 \leftarrow r_0 + r_0$.

The SLP constructed in this way has five registers.

Getting rid of subtractions. The final step is to replace the $(\max, +, -)$-SLP or $(\min, +, -)$-SLP over $\mathbb{Z}$ obtained so far, relabeled Q and fulfilling the condition that every register only ever holds nonnegative values, by a new $(\max, +)$-SLP or $(\min, +)$-SLP Q' over $\mathbb{N}$, where the value of every register r of Q at the end is equal to the value of the difference $r - r_{-1}$ of registers of Q', where r_{-1} is a new register of Q'. Initially, $r_{-1} \leftarrow 0$. Operations that involve min or max need no modification. If Q has the operation $r \leftarrow s + u$, then Q' has the operations

$r \leftarrow s+u; r' \leftarrow r'+r_{-1}$ (for every $r' \neq r$); $r_{-1} \leftarrow r_{-1}+r_{-1}$. (This is exactly the same replacement as was used in the preceding paragraph.) Finally, if Q has the operation $r \leftarrow s-u$, then (assuming with no loss of generality that r, s, u are distinct registers) Q' has the operations: $r \leftarrow s + r_{-1}; r_{-1} \leftarrow r_{-1} + u; r' \leftarrow r' + u$ for every $r' \notin \{r, u\}$; and then $u \leftarrow u + u$. An induction shows that the invariant is maintained, that each register r of Q has the value $r - r_{-1}$ of Q'. For instance, if Q has the instruction $r \leftarrow \min(r_i, r_j)$, then Q' also has the instruction $r \leftarrow \min(r_i, r_j)$, and the induction step would observe that $\min(r_i - r_{-1}, r_j - r_{-1}) = \min(r_i, r_j) - r_{-1}$. The other cases are equally trivial. The SLP constructed in this way has six registers.

This completes the discussion of how to build the desired SLP, and this also completes the proof. ∎

Corollary 9 *Let R be the semiring $(\mathbb{N} \cup \{\infty\}, \min, +)$ or $(\mathbb{N} \cup \{-\infty\}, \max, +)$. There is a constant c such that for every $k \geq c$, the width-k circuit value problem over R is* P*-complete under* AC^0*-Turing reductions.*

Proof: The P upper bounds are clear since each semiring operation is polynomial-time computable. Hardness follows by appealing to the straight-line programs with a bounded number of registers that are constructed in the proof of Theorem 8. A further AC^0-Turing reduction can transform a straight-line program that uses k registers into an arithmetic circuit of width $O(k)$. (Each layer in the arithmetic circuit contains a gate for each register, as well as gates for each constant that is used in the next time step. If a register r is not changed at time t, then the gate for register r in layer t is simply set to 0 + the value of register r at layer $t-1$.) ∎

Completeness under AC^0-many-one reductions (or even logspace many-one reductions) is still open.

4 CCRAs over Commutative Semirings

In this section, we study two classes of functions defined by CCRAs operating over commutative algebras with two operations satisfying the semiring axioms:

- CRAs operating over the commutative ring $(\mathbb{Z}, +, \times)$
- CRAs operating over the tropical semiring, that is, over the commutative semiring $(\mathbb{Z} \cup \{\infty\}, \min, +)$.

(Above, we use "+" to denote addition over the integers, although + is the "multiplicative" operation in the tropical semiring. In the following theorem, we use "+" to denote the "additive" operation in the semiring under consideration.)

Theorem 10 *Let $(\mathcal{A}, +, \times)$ be a commutative semiring such that the functions*

$$(x_1, x_2, \ldots, x_n) \mapsto \sum_i x_i \text{ and } (x_1, x_2, \ldots, x_n) \mapsto \prod_i x_i$$

can be computed in NC^1*. Then CCRA($\mathcal{A}$)* $\subseteq \mathsf{NC}^1$*.*

We remark that both the tropical semiring and the integers satisfy this hypothesis. We refer the reader to [32, 21] for more details about the inclusions:

- unbounded-fan-in min $\in \mathsf{AC}^0$.
- unbounded-fan-in $+ \in \mathsf{TC}^0$.
- unbounded-fan-in $\times \in \mathsf{TC}^0$.

Proof: Let $M = (Q, \Sigma, \delta, q_0, X, \rho, \mu)$ be a copyless CRA operating over $\mathcal{A}$. Let M have k registers $r_1, \ldots, r_k$.

As in the proof of [1, Theorem 1], it is straightforward to see that the following functions are computable in NC^1:

- $(x, i) \mapsto q$, such that M is in state q after reading the prefix of x of length i. Note that this also allows us to determine the state q that M is in while scanning the final symbol of x, and thus we can determine whether the output $\mu(q)$ is defined.
- $(x, i) \mapsto G_i$, where G_i is a labeled directed bipartite graph on $[2k] \times [k]$, with the property that there is an edge from j on the left-hand side to ℓ on the right hand side, if the register update operation that takes place when M consumes the i-th input symbol includes the update $r_\ell \leftarrow \alpha \otimes \beta$ where $r_j \in \{\alpha, \beta\}$ and $\otimes \in \{+, \times\}$. In addition, vertex ℓ is labeled with the operation $\otimes$. If one of $\{\alpha, \beta\}$ is a constant c (rather than being a register), then label vertex $k+\ell$ in the left-hand column with the constant c, and add an edge from vertex $k + \ell$ in the left-hand column to ℓ in the right-hand column. (To see that this is computable in NC^1, note that by the previous item, in NC^1 we can determine the state q that M is in as it consumes the i-th input symbol. Thus G_i is merely a graphical representation of the register update function corresponding to state q.) Note that the outdegree of each vertex in G_i is at most one, because M is copyless. (The indegree is at most two.) To simplify the subsequent discussion, define G_{n+1} to be the graph resulting from the "register update function" $r_\ell \leftarrow \mu(q)$ for $1 \leq \ell \leq k$, where q is the state that M is in after scanning the final symbol x_n.

Now consider the graph G that is obtained by concatenating the graphs G_i (by identifying the left-hand side of G_{i+1} with the first k vertices of the right-hand side of G_i for each i). This graph shows how the registers at time $i + 1$ depend on the registers at time i. G is a constant-width graph, and it is known that reachability in constant-width graphs is computable in NC^1 [10, 11].

The proof of the theorem proceeds by induction on the number of registers $k = |X|$. When $k = 1$, note that the graph G consists of a path of length $n + 1$, where each vertex v_i on the path is connected to two vertices on the preceding level, one of which is a leaf. (Here, we are ignoring degenerate cases, where the path back from the output node does not extend all the way back to the start, but instead stops at some vertex v_i where the corresponding register assignment function sets the register to a constant. An NC^1 computation can find where the path actually does start.) That is, when $k = 1$, the graph G has width two. We will thus really do our induction on the width of the graph G, starting with width two.

Along this path, some of the gates are + gates, and some are × gates. Assume for convenience that the first operation on the path is + and the last one is ×; otherwise add dummy initial and final operations that add 0 and multiply by 1, respectively. Thus, in $\mathsf{TC}^0 \subseteq \mathsf{NC}^1$, we can partition the index set $I = \{0, \ldots, n{+}1\}$ into consecutive nonempty subsequences $S_1, P_1, S_2, P_2, \ldots, S_m, P_m$, where $i \in S_j$ implies that vertex v_i on the path is labeled with +, and $i \in P_j$ implies that vertex v_i on the path is labeled with ×. That is, $i \in S_j$ implies that the i-th operation is of the form $v_i \leftarrow v_{i-1} + c_{i-1}$, and $i \in P_j$ implies that the i-th operation is of the form $v_i \leftarrow v_{i-1} \times c_{i-1}$ for some sequence of constants $c_0, \ldots, c_n$. (See Figure 5.) The number m is determined by the number of alternations between + and × along this path.

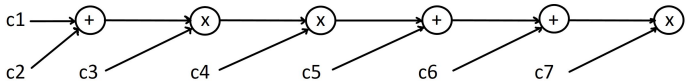

Figure 5: Diagram showing how to partition a path of + and × gates. Here S_1 is the leftmost + gate, P_1 consists of the next two × gates, S_2 consists of the next two + gates, and P_2 consists of the final × gate.

By hypothesis, in NC^1 we can compute the values $s_j = \sum_{i \in S_j} c_{i-1}$ and $p_j = \prod_{i \in P_j} c_{i-1}$. Thus the output computed by M on x is

$$(\ldots(((s_1 \times p_1) + s_2) \times p_2) \ldots \times p_m) = \sum_j s_j \prod_{\ell \geq j} p_\ell.$$

This expression can also be evaluated in NC^1. This completes the proof of the basis case, when $k = 1$.

Now assume that functions expressible in this way when the width of the graph G is at most k can be evaluated in NC^1. Consider the case when G has width $k{+}1$, and assume that vertex 1 in the final level is the vertex that evaluates to the value of the function. In NC^1 we can identify a path of longest length leading to the output. Let this path start in level i_0. Since there is no path from a vertex in any level $i < i_0$ to the output, we can ignore everything before level i_0 and just deal with the part of G starting at level i_0. Thus, for simplicity, assume that $i_0 = 0$. Let the vertices appearing on this path be $v_1, v_2, \ldots, v_{n+1}$, where each vertex v_i is labeled with the operation $v_i \leftarrow v_{i-1} \otimes_i w_i$ for some operation $\otimes_i$ and some vertex w_i. Let H_i be the subgraph consisting of all vertices that have a path to vertex w_i. Since the outdegree of each vertex in G is one, and since no w_i appears on the path, it follows that each H_i has width at most k, and thus the value computed by w_i (which we will also denote by w_i) can be computed in NC^1. (This is the only place where we use the restriction that M is a *copyless* CRA.)

Now, as before partition this path into subsequences $S_1, P_1, S_2, P_2, \ldots, S_m, P_m$, where $i \in S_j$ implies that the i-th operation is of the form $v_i \leftarrow v_{i-1} + w_{i-1}$, and $i \in P_j$ implies that the i-th operation is of the form $v_i \leftarrow v_{i-1} \times w_{i-1}$ for some NC^1-computable sequence of values $w_0, \ldots, w_n$.

Thus, as above, in NC^1 we can compute the values $s_j = \sum_{i \in S_j} w_{i-1}$ and

$p_j = \prod_{i \in P_j} w_{i-1}$. Thus the output computed by M on x is

$$(\ldots(((s_1 \times p_1) + s_2) \times p_2) \ldots \times p_m) = \sum_j s_j \prod_{\ell \geq j} p_\ell.$$

This expression can also be evaluated in NC^1. ∎

5 CCRAs over Noncommutative Semirings

In this section, we show that the techniques of the preceding section can easily be adapted to work for noncommutative semirings.

The canonical example of such a semiring is $(\Gamma^* \cup \{\perp\}, \max, \circ)$. Here, the max operation takes two strings x, y in Γ^* as input, and produces as output the lexicographically-larger of the two. (Lexicographic order on Γ^* is defined as usual, where $x < y$ if $|x| < |y|$ or ($|x| = |y|$ and x precedes y, viewed as the representation of a number in $|\Gamma|$-ary notation). $\perp$ is the additive identity element. (One obtains a similar example of a noncommutative semiring, by using min in place of max.)

It is useful to describe how elements of Γ^* will be represented in an NC^1 circuit, in a way that allows efficient computation. For an input length n, let $m = n^{O(1)}$ be the maximum number of symbols in any string that will need to be manipulated while processing inputs of length n. Then a string y of length j will be represented as a sequence of $\log m + m \log |\Gamma|$ bits, where the first $\log m$ bits store the number j, followed by m blocks of length $\log |\Gamma|$, where the first j blocks store the symbols of y. Given a sequence of $l_1, r_1, l_2, r_2, \ldots, l_s, r_s$ represented in this way, we need to compute the representation of the string $l_s l_{s-1} \ldots l_2 l_1 r_1 r_2 \ldots r_{s-1} r_s$. It is easy to verify that this computation is in TC^0, since the i-th symbol of the concatenated string is equal to the j-th symbol of the ℓ-th string in this list, where j and ℓ are easy to compute by performing iterated addition on the lengths of the various strings, and comparing the result with i. In the $\max, \circ$ semiring, where concatenation is the "multiplicative" operation, this corresponds to iterated product, and it is computable in $\mathsf{TC}^0 \subseteq \mathsf{NC}^1$. Thus the hypothesis of the following theorem is satisfied. (We state the theorem assuming that iterated sum and iterated product are in NC^1, rather than in TC^0, because the weaker hypothesis suffices.)

Theorem 11 *Let $(\mathcal{A}, +, \times)$ be a (possibly noncommutative) semiring such that the functions $(x_1, x_2, \ldots, x_n) \mapsto \sum_i x_i$ and $(x_1, x_2, \ldots, x_n) \mapsto \prod_i x_i$ can be computed in NC^1. Then CCRA($\mathcal{A}$) $\subseteq \mathsf{NC}^1$.*

Proof: The proof is a slight modification of the proof in the commutative case.

Given a CCRA M, we build the same graph G. Again, the proof proceeds by induction on the width of G (related to the number of registers in M).

Let us consider the basis case, where G has width two.

In $\mathsf{TC}^0 \subseteq \mathsf{NC}^1$, we can partition the index set $I = \{0, \ldots, n+1\}$ into consecutive subsequences $S_1, P_1, S_2, P_2, \ldots, S_m, P_m$, where $i \in S_j$ implies that vertex v_i on the path is labeled with $+$, and $i \in P_j$ implies that vertex v_i on the path is labeled with $\times$. (Assume for convenience that the first operation on the path is $+$ and the last one is $\times$; otherwise add dummy initial and final

operations that add 0 and multiply by 1, respectively.) That is, $i \in S_j$ implies that the i-th operation is of the form $v_i \leftarrow v_{i-1} + c_{i-1}$, and $i \in P_j$ implies that the i-th operation is of the form $v_i \leftarrow v_{i-1} \times c_{i-1}$ or $v_i \leftarrow c_{i-1} \times v_{i-1}$ for some sequence of constants $c_0, \ldots, c_n$.

In NC^1 we can compute the value $s_j = \sum_{i \in S_j} c_{i-1}$. The product segments P_j require just a bit more work. Let $l_{j,1}, l_{j,2}, \ldots, l_{j,m_{j_l}}$ be the list of indices, such that $l_{j,s}$ is the s-th element of $\{i \in P_j :$ the multiplication operation at v_i is of the form $v_i \leftarrow c_{i-1} \times v_{i-1}\}$, and similarly let $r_{j,1}, r_{j,2}, \ldots, r_{j,m_{j_r}}$ be the list of indices, such that $r_{j,s}$ is the s-th element of $\{i \in P_j :$ the multiplication operation at v_i is of the form $v_i \leftarrow v_{i-1} \times c_{i-1}\}$.

Let

$$l_j = c_{l_{j,m_{j_l}}-1} \times c_{l_{j,m_{j_l}-1}-1} \times \ldots c_{l_{j,2}-1} \times c_{l_{j,1}-1}$$

and let

$$r_j = c_{r_{j,1}-1} \times c_{r_{j,2}-1} \times c_{r_{j,m_{j_r}-1}-1} \times c_{r_{j,m_{j_r}}-1}.$$

Then if the value of the path when it enters segment P_j is y, it follows that the value computed when the path leaves segment P_j is $l_j y r_j$. Note that, by hypothesis, this value can be computed in NC^1.

Thus the output computed by M on x is

$$l_m \times ((l_{m-1} \times (\ldots (l_2 \times ((l_1 \times s_1 \times r_1) + s_2) \times r_2) \ldots) \times r_{m-1}) + s_m) \times r_m$$

which is equal to

$$\sum_{j=1}^{m} (\prod_{\ell \geq j} l_j) s_j (\prod_{\ell \geq j} r_\ell).$$

This expression can be evaluated in NC^1. This completes the proof of the basis case, when G has width two.

The proof for the inductive step is similar to the commutative case, combined with the algorithm for the basis case. ∎

6 Conclusion

We have obtained a P-completeness theorem for some functions computed by CRAs over $(\mathbb{N}, \min, +)$ and other tropical semirings. This was done by proving that a straight-line program over such semirings using $O(1)$ registers can solve a P-complete problem. It followed that for some small k, the "width-k circuit value problem" over $(\mathbb{N}, \min, +)$ is P-complete. We have also shown that any function computed by a copyless CRA over such semirings belongs to (functional) NC^1.

An open question of interest would be to characterize the semirings $(R, +, \times)$ over which the width-k circuit value problem is P-complete. Given the P-completeness of the circuit value problem over the group A_5 [12], one possible approach would be to try to map R onto A_5 in such a way that iterating the evaluation of a fixed semiring expression over R would allow retrieving the result of a linear number of compositions of permutations from A_5.

A future direction in the study of copyless CRAs might be to refine our NC^1 analysis by restricting the algebraic properties of the underlying finite automaton, along the lines described in the context of ordinary finite automata (see

Straubing [31] for a broader perspective). The way to proceed is not immediately clear however since merely restricting the finite automaton (say to an aperiodic automaton) would not reduce the strength of the model unless the interplay between the registers is also restricted.

Acknowledgments Some of this research was performed at the 29th McGill Invitational Workshop on Computational Complexity, held at the Bellairs Research Institute of McGill University, in February, 2017.

References

[1] E. Allender and I. Mertz. Complexity of regular functions. *Journal of Computer and System Sciences*, 2017. To appear; LATA 2015 Special Issue. Earlier version appeared in Proc. 9th International Conference on Language and Automata Theory and Applications (LATA '15), Springer Lecture Notes in Computer Science vol. 8977, pp. 449-460.

[2] Eric Allender. Arithmetic circuits and counting complexity classes. In J. Krajíček, editor, *Complexity of Computations and Proofs*, volume 13 of *Quaderni di Matematica*, pages 33–72. Seconda Università di Napoli, 2004.

[3] Eric Allender, Andreas Krebs, and Pierre McKenzie. Better complexity bounds for cost register automata. In *42nd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 83 of *LIPIcs*, pages 24:1–24:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

[4] Rajeev Alur and Pavol Cerný. Streaming transducers for algorithmic verification of single-pass list-processing programs. In *38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL*, pages 599–610, 2011.

[5] Rajeev Alur, Loris D'Antoni, Jyotirmoy V. Deshmukh, Mukund Raghothaman, and Yifei Yuan. Regular functions, cost register automata, and generalized min-cost problems. *CoRR*, abs/1111.0670, 2011.

[6] Rajeev Alur, Loris D'Antoni, Jyotirmoy V. Deshmukh, Mukund Raghothaman, and Yifei Yuan. Regular functions and cost register automata. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 13–22, 2013. See also the expanded version, [5].

[7] Rajeev Alur, Adam Freilich, and Mukund Raghothaman. Regular combinators for string transformations. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science, (CSL-LICS)*, page 9. ACM, 2014.

[8] Rajeev Alur and Mukund Raghothaman. Decision problems for additive regular functions. In *Proc. 40th International Colloquium on Automata, Languages, and Programming (ICALP)*, Lecture Notes in Computer Science, pages 37–48. Springer, 2013.

[9] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*, volume 1. Cambridge University Press, 2009.

[10] D. A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38:150–164, 1989.

[11] D. A. M. Barrington, C.-J. Lu, P. B. Miltersen, and S. Skyum. Searching constant width mazes captures the AC^0 hierarchy. In *Proc. 15th International Symposium on Theoretical Aspects of Computer Science (STACS)*, number 1373 in Lecture Notes in Computer Science, pages 73–83. Springer, 1998.

[12] M. Beaudry, P. McKenzie, P. Péladeau, and D. Thérien. Finite monoids: from word to circuit evaluation. *SIAM Journal on Computing*, 26:138–152, 1997.

[13] Michael Ben-Or and Richard Cleve. Computing algebraic formulas using a constant number of registers. *SIAM Journal on Computing*, 21(1):54–58, 1992.

[14] Michaël Cadilhac, Andreas Krebs, and Nutan Limaye. Value automata with filters. *CoRR*, abs/1510.02393, 2015.

[15] Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *Automata, Languages and Programming, 36th Internatilonal Colloquium, ICALP 2009, Proceedings, Part II*, pages 139–150, 2009.

[16] Thomas Colcombet. Regular cost functions, part I: logic and algebra over words. *Logical Methods in Computer Science*, 9(3), 2013.

[17] Thomas Colcombet, Denis Kuperberg, Amaldev Manuel, and Szymon Toruńczyk. Cost functions definable by min/max automata. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, pages 29:1–29:13, 2016.

[18] Laure Daviaud, Pierre-Alain Reynier, and Jean-Marc Talbot. A generalised twinning property for minimisation of cost register automata. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 857–866, 2016.

[19] Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata*. Springer-Verlag New York Inc., 2009.

[20] Raymond Greenlaw, H James Hoover, and Walter L Ruzzo. *Limits to parallel computation: P-completeness theory*. Oxford University Press, 1995.

[21] William Hesse, Eric Allender, and David A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65:695–716, 2002.

[22] Howard J Karloff and Walter L Ruzzo. The iterated mod problem. *Information and Computation*, 80(3):193–204, 1989.

[23] Michal Koucký. Unpublished Manuscript., 2003.

[24] Andreas Krebs, Nutan Limaye, and Michael Ludwig. Cost register automata for nested words. In *Proc. 22nd International Computing and Combinatorics Conference - (COCOON)*, number 9797 in Lecture Notes in Computer Science, pages 587–598. Springer, 2016.

[25] Nancy A Lynch. Straight-line program length as a parameter for complexity analysis. *Journal of Computer and System Sciences*, 21(3):251–280, 1980.

[26] Meena Mahajan, Prajakta Nimbhorkar, and Anuj Tawari. Computing the maximum using (min, +) formulas. In *42nd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 83 of *LIPIcs*, pages 74:1–74:11. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

[27] Filip Mazowiecki and Cristian Riveros. Maximal partition logic: Towards a logical characterization of copyless cost register automata. In *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*, pages 144–159, 2015.

[28] Filip Mazowiecki and Cristian Riveros. Copyless cost-register automata: Structure, expressiveness, and closure properties. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, pages 53:1–53:13, 2016.

[29] Jean-Eric Pin. *Tropical semirings.* Cambridge Univ. Press, Cambridge, 1998.

[30] Nicholas Pippenger. On simultaneous resource bounds. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 307–311, 1979.

[31] H. Straubing. *Finite Automata, Formal Logic, and Circuit Complexity.* Birkhäuser, Boston, 1994.

[32] H. Vollmer. *Introduction to Circuit Complexity: A Uniform Approach.* Springer-Verlag New York Inc., 1999.