Better Complexity Bounds for Cost Register Automata*

Eric Allender[1], Andreas Krebs[2], and Pierre McKenzie[3]

1 **Department of Computer Science, Rutgers University, Piscataway, NJ, USA**
allender@cs.rutgers.edu
2 **WSI, Universität Tübingen, Germany**
mail@krebs-net.de
3 **DIRO, Université de Montréal, Québec, Canada**
mckenzie@iro.umontreal.ca

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 $(\Gamma^* \cup \{\bot\}, \max, \text{concat})$. 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.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases computational complexity, cost registers

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.24

1 Introduction

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 [17]) 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 [3], soon followed by the cost register automaton (CRA) [5].

CRAs are deterministic and are yet strictly more expressive than weighted automata [5]. 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 [13].) A "copyless" variant (CCRA) of the CRA has the expressivity of single-valued weighted automata [5, Thm 4]. Another variant of CRAs restricts the multiplicative operation, by only allowing multiplication by constants; this model has the full expressivity of weighted automata [5, Thm 9]. A theory of CRAs, largely concerned with expressivity and decidability properties, was developed in a series of papers, including [5, 7, 6].

* Supported by NSF grant CCF-1555409 (Allender), by the DFG Emmy-Noether program KR 4042/2 (Krebs), and by the Natural Sciences and Engineering Research Council of Canada (McKenzie)

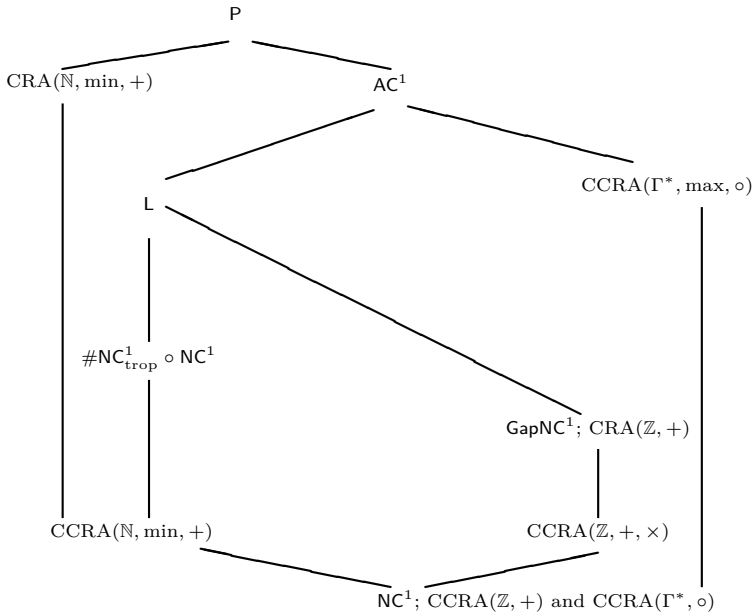

Figure 1 Prior state of knowledge, from [1]. When a class of CRA functions and a complexity class appear together, it means that containment of the CRA class in the complexity class is tight, since some of the CRA functions are complete for the complexity class. (Definitions of the complexity classes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

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 [22, 25, 24, 16, 15, 12].

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 [14, Sect. 1.1] and [5, 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 $\#\mathsf{NC}^1_{\text{trop}}$?
- 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 $\mathsf{NC}^1 = \#\mathsf{NC}^1_{\text{trop}}$, 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 5.) Figure 2 summarizes our results.

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.

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 [29, 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 $O(\log^i n)\}$.

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 [29] 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 $\{(x, i, b) :$ the i-th symbol of $f(x)$ is $b\}$ is in NC^1. Similar observations hold for other classes.

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.) The 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 [27]: We will consider *layered* circuits, which means that the set of gates is partitioned into layers, where wires connect only gates in adjacent layers. The width of a circuit is the largest number of gates that occurs in any 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 semiring R with registers $\{r_1, \ldots, r_k\}$ consists of a sequence of statements of the form $r_i \leftarrow r_j \odot r_k$ where $\odot$ is one of the semiring operations, and each r_ℓ is 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 [23], 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 *width-k circuit value problem* over a semiring R is that of determining, given a width-k arithmetic circuit C over R (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 [29, 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 $\mathsf{GapNC}^1 \subseteq \mathsf{L}$ [19].

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),
- moves to a new state (based on a and the current state (call it q)),
- 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 $\mathcal{A}$.

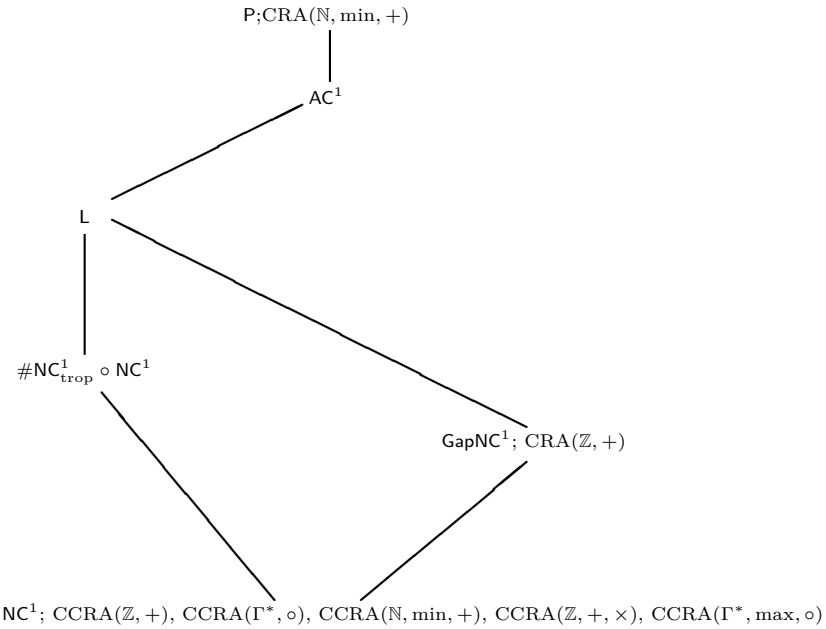

Figure 2 Update of the preceding figure, showing the improved state of our knowledge regarding CRA($\mathbb{N}, \min, +$) and the copyless CRA classes CCRA($\mathbb{N}, \min, +$), CCRA($\mathbb{Z}, +, \times$), and CCRA($\Gamma^*, \max, \circ$). All bounds listed are now tight.

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.

More formally, here is the definition as presented by Alur *et al.* [5].

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 $w = a_1 \ldots a_n$, the *run* of M on w is the sequence of configurations $(q_0, \nu_0), \ldots (q_n, \nu_n)$ such that, for each $i \in \{1, \ldots, n\}$ $\delta(q_{i-1}, a_i) = q_i$ 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 $\mu(q_n)$).

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 [5] 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 [6], 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.

3 CRAs over the Tropical Semiring

CRAs *without* the copyless restriction over the tropical semiring still yield only functions that are computable in polynomial time. The "repeated squaring" operation, when the "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 [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, [18], 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 they are allowed to have an unbounded number of registers.

Our focus will be more on straight-line programs with a *bounded* number of registers. Ben-Or and Cleve [11] showed that straight-line programs with $O(1)$ registers can simulate arithmetic formulae, and Koucky [21] 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. 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.

▶ **Theorem 4.** *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 [18]), which was shown to be P-complete under logspace reductions by Karloff and Ruzzo [20]. The proof in [20] actually shows that the problem is complete under many-one reductions computable by dlogtime-uniform AC^0 circuits. (Incidentally, the proof sketch in [18] has a minor error, in that some indices are listed in the wrong order. The reader is advised to consult the original [20] proof.)

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

Let c be chosen so that 2^c is greater than any of the numbers $v, m_1, m_2, \ldots, m_n$. Then the naïve division algorithm that one would use to compute $v \bmod m$ can be seen to involve computing the following sequence:

- $v_0 = v$
- $v_i = v_{i-1} - max(0, v_{i-1} - m \cdot 2^{c-i})$

By induction, one can see that each $v_i \equiv v \pmod m$ and $v_i < m2^{c-i}$, and hence v_c is the remainder when one divides v by m.

Thus $v \bmod m$ can be seen to be computed by the following straight-line program over $\mathbb{Z}$ with operations $+, -, \max$:

```
1:  for i ≤ c do
2:     shift_of_m ← m
3:     for k ≤ c − i do
4:        shift_of_m ← shift_of_m + shift_of_m
5:     end for
6:     {At end of this loop, shift_of_m = m2^{c−i}}
7:     temp ← v−shift_of_m
8:     temp ← max(0, temp)
9:     v ← v − temp
10: end for
```

Of course, by definition, straight-line programs contain no loop statements, but the algorithm can be computed by a program described by an AC^0-computable sequence of symbols from the 5-letter alphabet $\{$*shift_of_m* $\leftarrow m$, *temp* $\leftarrow v-$ *shift_of_m*, $v \leftarrow v-$*temp*, *shift_of_m* $\leftarrow$ *shift_of_m* + *shift_of_m*, *temp* $\leftarrow \max(0,$*temp*$)\}$.

A number v, presented as a sequence of b binary digits v_i, can be loaded into a register r by initially setting r to 0, and then executing b instructions of the form $r \leftarrow r + r + v_i$. Thus, the naïve polynomial-time algorithm for computing Iterated Mod can be implemented via a polynomial-size straight-line program over $\mathbb{Z}$ with operations $+, -, \max$, by first inputting the numbers v and m_1, executing the algorithm above to compute $v \bmod m_1$, then inputting m_2, repeating the procedure to compute $((v \bmod m_1) \bmod m_2)$, etc.

We observe next that $\max(a, b) = (-1) \cdot \min(-a, -b)$. Thus there is a polynomial-size straight-line program over $\mathbb{Z}$ with operations $+, -, \min$ that outputs 0 if and only if $(v, m_1, \ldots, m_n)$ is a positive instance of Iterated Mod, where the process to input a number in binary has each instruction $r \leftarrow r + r + v_i$ replaced by $r \leftarrow r + r - v_i$. Similarly, in the code to compute $v \bmod m$, each occurrence of the instruction *temp* $\leftarrow \max(0,$*temp*$)$ is replaced by *temp* $\leftarrow \min(0,$*temp*$)$, and each occurrence of $r \leftarrow v - s$ for $\{r, s\} \subseteq \{v,$ *temp*, *shift_of_m*$\}$ is replaced by $r \leftarrow s - v$.

The next observation is that, given any straight-line program Q over $\mathbb{Z}$, it is easy to build a straight-line program Q' over $\mathbb{Z}$, 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 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 $r \neq r_0$, 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 $r \leftarrow r_0 + b$. Replace each operation $r \leftarrow s - u$ by $r \leftarrow s - u + r_0$, and replace each operation $r \leftarrow s + u$ by the operations: $r \leftarrow s + u; r' \leftarrow s + r_0$ (for every $r' \neq r$); $r_0 \leftarrow r_0 + r_0$.

The final step is to replace every straight-line program Q over $(\mathbb{Z}, \max, +, -)$ or ($\mathbb{Z}$,min,+, -) where every register holds only nonnegative values by a new program Q' over $(\mathbb{N}, \max, +)$ or $(\mathbb{N}, \min, +)$, 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 $r \leftarrow s + u; r' \leftarrow s + r_{-1}$ (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 Q' has the operations: $r \leftarrow s + r_{-1}; r_{-1} \leftarrow r_{-1} + u; r' \leftarrow r' + u$ for every $r' \notin \{r, u\}$ (including $r' = r_{-1}$); and then $u \leftarrow u + u$. A straightforward induction shows that the invariant is maintained, that each register r of Q has the value $r - r_{-1}$ of Q'.

Thus, given an instance y of Iterated Mod, an AC^0 reduction can produce a straight-line program Q over $(\mathbb{N}, \min, +)$ or $(\mathbb{N}, \max, +)$, such that $y \in$ Iterated Mod iff the output register of Q has a value equal to the value of r_0.

Note that there is a CRA that takes as input strings over an alphabet whose symbols encode straight-line program instructions with $O(1)$ registers, and simulates the operation of the straight-line program. The function f that is computed by this CRA is the function whose existence is asserted in the statement of the theorem. ◀

▶ **Corollary 5.** *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 4. 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 in 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 $(\mathbb{Z} \cup \{\infty\}, \min, +)$.

▶ **Theorem 6.** *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 [29, 19] 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 $1 \le \ell \le k$, 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 [8, 9].

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.

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 $v_i \leftarrow v_{i-1} \times c_{i-1}$ for some sequence of constants $c_0, \ldots, c_n$.

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 $(\Gamma^* \cup \{\bot\}, \max, \circ)$. 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). $\bot$ 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 can 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 $\mathsf{TC}^0 \subseteq \mathsf{NC}^1$.

▶ **Theorem 7.** *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 $\{i \in P_j : \text{the multiplication operation at } v_i \text{ is of the form } v_i \leftarrow c_{i-1} \times v_{i-1}\}$, 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 $\{i \in P_j : \text{the multiplication operation at } v_i \text{ is of the form } v_i \leftarrow v_{i-1} \times c_{i-1}\}$.

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 this value can be computed in NC^1.

Thus the output computed by M on x is

$$l_1 \times ((l_2 \times (\ldots (l_2 \times ((l_1 \times s_1 \times r_1) + s_2) \times r_2) \ldots) \times r_2) + s_1) \times r_1$$

which is equal to

$$\sum_j (\prod_{\ell \geq j} l_j) s_j (\prod_{\ell \geq j} r_\ell).$$

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 polynomial time lower bound, conditional on $\mathsf{NC} \neq \mathsf{P}$, 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 [10], 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 [28] 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. doi:10.1016/j.jcss.2016.10.005.

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 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. doi:10.1145/1926385.1926454.

4 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. URL: https://arxiv.org/abs/1111.0670.

5 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, [4]. doi:10.1109/LICS.2013.65.

6 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. doi:10.1145/2603088.2603151.

7 Rajeev Alur and Mukund Raghothaman. Decision problems for additive regular functions. In *Proc. 40th International Colloquium on Automata, Languages, and Programming (ICALP)*, number 7966 in Lecture Notes in Computer Science, pages 37–48. Springer, 2013. doi:10.1007/978-3-642-39212-2_7.

8 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. doi:10.1016/0022-0000(89)90037-8.

9 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. doi:10.1007/BFb0028542.

10 M. Beaudry, P. McKenzie, P. Péladeau, and D. Thérien. Finite monoids: from word to circuit evaluation. *SIAM Journal on Computing*, 26:138–152, 1997. doi:10.1137/S0097539793249530.

11 Michael Ben-Or and Richard Cleve. Computing algebraic formulas using a constant number of registers. *SIAM Journal on Computing*, 21(1):54–58, 1992. doi:10.1137/0221006.

12 Michaël Cadilhac, Andreas Krebs, and Nutan Limaye. Value automata with filters. *CoRR*, abs/1510.02393, 2015. URL: http://arxiv.org/abs/1510.02393.

13 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. doi:10.1007/978-3-642-02930-1_12.

14 Thomas Colcombet. Regular cost functions, part I: logic and algebra over words. *Logical Methods in Computer Science*, 9(3), 2013. doi:10.2168/LMCS-9(3:3)2013.

15 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. doi:10.4230/LIPIcs.STACS.2016.29.

16 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. doi:10.1145/2933575.2934549.

17 Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata*. Springer-Verlag New York Inc., 2009. doi:10.1007/978-3-642-01492-5.

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

19 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. doi:10.1016/S0022-0000(02)00025-9.

20 Howard J Karloff and Walter L Ruzzo. The iterated mod problem. *Information and Computation*, 80(3):193–204, 1989. doi:10.1016/0890-5401(89)90008-4.

21 Michal Koucký. Unpublished manuscript, 2003.

22 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 LNCS, pages 587–598. Springer, 2016. doi:10.1007/978-3-319-42634-1_47.

23 Nancy A Lynch. Straight-line program length as a parameter for complexity analysis. *Journal of Computer and System Sciences*, 21(3):251–280, 1980. doi:10.1016/0022-0000(80)90024-0.

24 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. doi:10.4230/LIPIcs.CSL.2015.144.

25 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. doi:10.4230/LIPIcs.STACS.2016.53.

26 Jean-Eric Pin. *Tropical semirings*. Cambridge Univ. Press, Cambridge, 1998. doi:10.1017/CBO9780511662508.004.

27 Nicholas Pippenger. On simultaneous resource bounds. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 307–311, 1979. doi:10.1109/SFCS.1979.29.

28 H. Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser, Boston, 1994. doi:10.1007/978-1-4612-0289-9.

29 H. Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer-Verlag New York Inc., 1999. doi:10.1007/978-3-662-03927-4.