Better Complexity Bounds for Cost Register Machines

Eric Allender\textsuperscript{1}, Andreas Krebs\textsuperscript{2}, and Pierre McKenzie\textsuperscript{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 \(\text{NC}^1\) when the semiring is \((\mathbb{Z}, +, \times)\) or \((\Gamma^* \cup \{\bot\}, \text{max}, \text{concat})\). 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 \(\text{NC}^1\) versus \(P\) and \(\text{NC}^1\) versus \(\text{GapNC}^1\).

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases computational complexity, cost registers, weighted automata

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

The Cost Register Automaton (CRA) model of computation was defined by Alur \textit{et al.} and studied in a series of papers, including [4, 6, 5]. This model has application in the field of computer-aided verification. Much previous work has focused on the so-called “copyless” variant of the CRA model (CCRA). Complexity bounds for functions computable in this model were presented in [1], and this line of inquiry was extended in [18, 21, 20, 13, 12, 11]. The purpose of this paper is to improve some of the bounds that were presented in [1].

In particular, the closing section of [1] listed the following four open questions:

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

In this paper, we essentially answer all of these questions, modulo long-standing open questions in complexity theory. We show that CCRA functions over each of \((\mathbb{Z}, +, \times),\)
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.

(G*, max, o), and the tropical semiring are all computable in NC1. 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 NC1 = GapNC1 and NC1 = #NC1trop, respectively. We also provide a negative answer to the fourth question (assuming NC ≠ 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 (N, max, +) and (N, min, +) is P-complete under AC0-Turing reductions. (See Section 2 for the precise definition of the problem and then Corollary 5.)

Figures 1 and 2 diagram the state of our knowledge before this work, and after.

2 Preliminaries

A structure (A, +, ×) is a semiring if (A, +) is a commutative monoid with an additive identity element 0, and (A, ×) 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.

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 ⊆ \{0,1\}^* is accepted by a Boolean circuit family (C_n)_{n∈N} if for all x it holds that x ∈ A iff C_{|x|}(x) = 1. Circuit families encountered in this paper will be uniform. Uniformity is a somewhat messy issue because of subtleties encountered at low complexity levels. We will not be concerned with such subtleties, but for definiteness we will assume a circuit family (C_n)_{n≥0} to be Uε-uniform (see [25, Sect. 4.5]), which means, informally, 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.
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.

- $\text{NC}^1 = \{A : A$ 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 $O(\log n)$\}.
- $\text{AC}^1 = \{A : A$ 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 $O(\log n)$\}.
- $\text{TC}^1 = \{A : A$ is accepted by a $U_E$-uniform family of circuits of unbounded fan-in MAJORITY gates, having size $n^{O(1)}$ and depth $O(\log n)$\}.

We remark that, for constant-depth classes such as $\text{AC}^0$ and $\text{TC}^0$, $U_E$-uniformity coincides with $U_D$-uniformity, which is also frequently called DLOGTIME-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, the function $f$ is said to be in $\text{NC}^1$ 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 (say) $\text{NC}^1$ if and only if the language $\{(x, i, b) :$ the $i$-th symbol of $f(x)$ is $b\}$ is in $\text{NC}^1$.

Definition 1. An arithmetic circuit over a semiring $(S, +, \times)$ is a directed acyclic graph. Each vertex of the graph is called a “gate”; each gate is labeled with a “type” from the set $\{+, \times, \text{input, constant}\}$, 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 $S$. 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 [23]: 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 $(S,+,\times)$ has $n$ input gates, then $C_n$ computes a function $f : S^n \to S$ in the obvious way.

**Definition 2.** A **straight-line program** over a semiring $R$ 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 $r_k$ comes from a finite set of registers (or from $R$, or from the set of input variables). Straight-line programs have been studied at least as far back as [19], 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 $b$.

$\#\text{NC}^1_S$ is the class of functions $f : \bigcup_n S^n \to S$ for which there is a $U_E$-uniform family of arithmetic circuits $\{C_n\}$ of logarithmic depth, such that $C_n$ computes $f$ on $S^n$.

By convention, when there is no subscript, $\#\text{NC}^1$ denotes $\#\text{NC}^1_{\mathbb{N}}$, with the additional restriction that the functions in $\#\text{NC}^1$ 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.)

$\text{GapNC}^1$ is defined as $\#\text{NC}^1 - \#\text{NC}^1$; that is: the class of all functions that can be expressed as the difference of two $\#\text{NC}^1$ functions. It is the same as $\#\text{NC}^1_{\mathbb{Z}}$ restricted to the Boolean domain. See [25, 2] for more on $\#\text{NC}^1$ and $\text{GapNC}^1$.

The following inclusions are known:

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

All inclusions are straightforward, except for $\text{GapNC}^1 \subseteq L$ [15].

### 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 $\mathcal{A}$.

There is also an “output” function $\mu$ defined on the set of states; $\mu$ 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. [4].

A cost-register automaton $M$ is a tuple $(\Sigma, Q, q_0, X, \delta, p, \mu)$, where

- $\Sigma$ 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.
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 [22]. We will not need to make any use of $\infty$ 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, [14], 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 [10] showed that straight-line programs with $O(1)$ registers can simulate arithmetic formulae, and Koucky [17] 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 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 does not result in a decrease in computational complexity.

We will present a reduction from the \( P \)-complete problem Iterated Mod (problem A.8.5 in [14]), which was shown to be \( P \)-complete under logspace reductions by Karloff and Ruzzo [16]. The proof in [16] actually shows that the problem is complete under many-one reductions computable by \( \text{dlogtime-uniform } AC^0 \) circuits. (Incidentally, the proof sketch in [14] has a minor error, in that some indices are listed in the wrong order. The reader is advised to consult the original [16] 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 \( (\cdots ((v \mod m_1) \mod m_2) \cdots) \mod m_n = 0 \).

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 \( v \mod m \) can be seen to involve computing the following sequence:

\[
\begin{align*}
&v_0 = v \\
&v_1 = v_{i-1} - \max(0, v_{i-1} - m \cdot 2^{c-i})
\end{align*}
\]

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

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

```plaintext
1: for \( i \leq c \) do
2:   shift_of_m \leftarrow m
3:   for \( k \leq c - i \) do
4:     shift_of_m \leftarrow shift_of_m + shift_of_m
5:   end for
6:   \{At end of this loop, \( shift_of_m = m2^{c-i} \}\}
7:   temp \leftarrow v - shift_of_m
8:   temp \leftarrow \max(0, temp)
9:   v \leftarrow 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 \( \{\text{shift}_of_m \leftarrow m, \text{temp} \leftarrow v \leftarrow \text{shift}_of_m, v \leftarrow v - \text{temp}, \text{shift}_of_m \leftarrow \text{shift}_of_m + \text{shift}_of_m, \text{temp} \leftarrow \max(0, \text{temp})\} \).

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 \( v \mod m_1 \), then inputting \( m_2 \), repeating the procedure to compute \( (v \mod m_1 \mod m_2) \), 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 $v \mod m$, 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 $(\mathbb{Z}, \max, +, -)$ by $(\mathbb{Z}, \min, +, -)$ where each 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 $\text{Iterated Mod}$, an $\text{AC}^0$ reduction can produce a straight-line program $Q$ over $(\mathbb{N}, \min, +)$ or $(\mathbb{N}, \max, +)$, such that $y \in \text{Iterated Mod}$ 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. We have thus proved the following theorem:

**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 $\mathsf{P}$ under $\text{AC}^0$-Turing reductions.

**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 $\mathsf{P}$-complete under $\text{AC}^0$-Turing reductions.

**Proof.** The $\mathsf{P}$ 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 $\text{AC}^0$-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 $\text{AC}^0$-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 $\langle \mathbb{Z}, +, \times \rangle$
- CRAs operating over the tropical semiring, that is, over the commutative semiring $\langle \mathbb{Z} \cup \{\infty\}, \min, + \rangle$.

\textbf{Theorem 6.} Let $\langle A, +, \times \rangle$ be a commutative 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 $\text{NC}^1$. Then $\text{CCRA}(A) \subseteq \text{NC}^1$.

We remark that both the tropical semiring and the integers satisfy this hypothesis. We refer the reader to [25, 15] for more details about the inclusions:

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

\textbf{Proof.} Let $M = (Q, \Sigma, \delta, q_0, X, \rho, \mu)$ be a copyless CRA operating over $A$. 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 $\text{NC}^1$:

- $(x, i) \mapsto q$, 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 $\ell$ on the right-hand side, if the register update operation that takes place when $M$ consumes the $i$-th input symbol includes the update $r_\ell \leftarrow \alpha \oplus \beta$ where $r_j \in \{\alpha, \beta\}$ and $\oplus \in \{+, \times\}$. In addition, vertex $\ell$ is labeled with the operation $\oplus$. 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 $\ell$ in the right-hand column. (To see that this is computable in $\text{NC}^1$, note that by the previous item, in $\text{NC}^1$ 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 $\text{NC}^1$ [7, 8].

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 $\text{NC}^1$ 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 $\text{TC}^0 \subseteq \text{NC}^1$, 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 $\text{NC}^1$ 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 $\text{NC}^1$. 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 $\text{NC}^1$. 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 $\text{NC}^1$ 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 operations $v_i \leftarrow v_{i-1} \otimes_i w_i$ 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 $\text{NC}^1$. (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 $\text{NC}^1$-computable sequence of values $w_0, \ldots, w_n$.

Thus, as above, in $\text{NC}^1$ 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 $\text{NC}^1$.

## 5 CCRAs over Noncommutative Semirings

In this section, 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 $\Gamma^*$ as input, and produces as output the lexicographically-larger of the two. (Lexicographic order on $\Gamma^*$ 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 $\Gamma^*$ will be represented in an $\text{NC}^1$ 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 $\log m + \log |\Gamma|$ 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 $y$. It is easy to verify that, in $\text{TC}^0$, given a sequence of strings $l_1, r_1, l_2, r_2, \ldots, l_s, r_s$ represented in this way, one can compute the representation of the string $l_1l_2l_3\ldots l_{s-1}r_1r_2r_3\ldots r_{s-1}r_s$, since the $i$-th symbol of the concatenated string is equal to the $j$-th symbol of the $\ell$-th string in this list, where $j$ and $\ell$ are easy to compute by performing iterated addition on the lengths of the various strings, and comparing the result with $i$. In the max, o semiring, where concatenation is the “multiplicative” operation, this corresponds to iterated product, and it is computable in $\text{TC}^0 \subseteq \text{NC}^1$.

**Theorem 7.** Let $(A, +, \times)$ 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 $\text{NC}^1$. Then $\text{CCRA}(A) \subseteq \text{NC}^1$.

**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 $\text{TC}^0 \subseteq \text{NC}^1$, 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 $v_i \leftarrow v_{i-1} + v_{i-1}$, and $i \in P_j$ implies that the $i$-th operation is of the form $v_i \leftarrow v_{i-1} \times v_{i-1}$ for some sequence of constants $c_0, \ldots, c_n$.

In $\text{NC}^1$ 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 $l_{j,1}, l_{j,2}, \ldots, l_{j,m_j}$ 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 $v_i$ is of the form $v_i \leftarrow v_{i-1} \times v_{i-1}$}\}$, and similarly let $r_{j,1}, r_{j,2}, \ldots, r_{j,m_j}$ 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 $v_i$ is of the form $v_i \leftarrow v_{i-1} \times v_{i-1}$}\}$.

Let

$$l_j = c_{j,m_j} \ldots c_{j,3} \times c_{j,2} \times c_{j,1}$$

and let

$$r_j = c_{j+1} \ldots c_{j,3} \times c_{j,2} \times c_{j,1} \times c_{j,0}.$$ 

Then if the value of the path when it enters segment $P_j$ is $y_j$, it follows that the value computed when the path leaves segment $P_j$ is $l_j y_j r_j$. Note that this value can be computed in $\text{NC}^1$.

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

$$\sum_j (\prod_{\ell \geq j} l_j s_j (\prod_{\ell \geq j} r_j)).$$
This expression can be evaluated in \( \text{NC}^1 \). 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 \( \text{NC} \neq \text{P} \), 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 \( \text{P} \)-complete problem. It followed that for some small \( k \), the “width-\( k \) circuit value problem” over \( (\mathbb{N}, \min, +) \) is \( \text{P} \)-complete. We have also shown that any function computed by a copyless CRA over such semirings belongs to (functional) \( \text{NC}^1 \).

An open question of interest would be to characterize the semirings \( (S, +, \times) \) over which the width-\( k \) circuit value problem is \( \text{P} \)-complete. Given the \( \text{P} \)-completeness of the circuit value problem over the group \( A_5 \) [9], one possible approach would be to try to map \( S \) onto \( A_5 \) in such a way that iterating the evaluation of a fixed semiring expression over \( S \) 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 \( \text{NC}^1 \) analysis by restricting the algebraic properties of the underlying finite automaton, along the lines described in the context of ordinary finite automata (see Straubing [24] 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

5. Rajeev Alur, Adam Freilich, and Mukund Raghothaman. Regular combinators for string transformations. In Joint Meeting of the Twenty-Third EACSL Annual Conference on
Better Complexity Bounds for Cost Register Machines


