On the expressibility of copyless cost register automata

Filip Mazowiecki\textsuperscript{2} and Cristian Riveros\textsuperscript{1}

\textsuperscript{1} Pontificia Universidad Católica de Chile
\textsuperscript{2} University of Warsaw

Abstract. Cost register automata (CRA) were proposed by Alur et all as an alternative model for weighted automata. In hope of finding decidable subclasses of CRA, they proposed to restrict their model with the copyless restriction but nothing is really know about the structure or properties of this new computational model called copyless CRA. In this paper we study the properties and expressiveness of copyless CRA. We propose a normal form for copyless CRA and we study the properties of a special group of registers (called stable registers). Furthermore, we find that copyless CRA do not have good closure properties since we show that they are not closed under reverse operation. Finally, we propose a subclass of copyless CRA and we show that this subclass is closed under regular-lookahead.

1 Introduction

Weighted automata are an extension of finite state automata for computing functions over words [5]. The main idea is that every transition of the automaton is mapped to an element of a fixed semi-ring. A run of the automaton on a word is then mapped to the product of its transitions and the value of an automaton on a word is the sum over all runs. By choosing a suitable semiring such as the min-plus semiring this allows to compute the minimal costs of a run over a word or similar quantitative properties of word languages. Weighted automata have been extensibly studied since Schützenberger [14] and their expressiveness [4,9], decidability [10,1], extensions [4], and applications [11,3] are very well known.

Recently, Alur et all [2] introduce the computational model of cost register automata (CRA), an alternative model to weighted automata for computing functions. The main idea of this model is to enhance deterministic finite automata with registers that can be combined with semiring operations, but they cannot be used for taking decisions during a computation. The output of CRA over an input is then given by a final function over the registers content. In [2], Alur et all shows that CRA are equally expressive than weighted automata, that is, they define the same class of functions than weighted automata.

The main advantage of introducing a new but equivalent model is that it allows to study natural subclasses of functions that could not be proposed from the
classical perspective. This is the case for the class of copyless CRA that where proposed in [2] in hope of finding decidable subclasses of weighted automata. The idea of the so-called copyless restriction is to use each register at most once in every transition. Intuitively, the automaton model is register-deterministic in the sense that it cannot copy the content of each register similar than a deterministic finite automaton that cannot make a copy of its current state. Despite that this is a natural and interesting model for computing functions, research on this line has not been pursued further and nothing is known until yet.

In this paper we embark in the first study of the structure and expressiveness of copyless CRA. We start this research by proposing a normal form for copyless CRA that, given any total order on the set of registers, one can construct an equivalent copyless CRA such that during transitions the content of each register is transfered to lower registers with respect to the total order. The normal form of a copyless CRA shows how the registers content flows from higher to lower registers and how registers are used as “aggregators”. This normal form motivates to divide the set of registers into two types: stable and non-stable registers. Non-stable registers store and operate its content like any register but they are unstable in the sense that at some point on a run they pass its content to lower registers forgetting what they have computed so far. In contrast, stable registers never pass their content to lower registers and they are always receiving and aggregating the content of non-stable registers. We show that from any state in a copyless CRA in normal form, one can force the automaton to read a word and “reset” the content of non-stable registers. This implies that during a run the output of a function depends directly from the content of stable registers and non-stable registers are just acting as temporary reservoirs. Finally, we also study the behavior of stable register during an automaton cycle. We show that when an automaton cycle is repeated, stable registers are updated as (pseudo) geometric series.

We use the previous results to study the expressiveness of copyless CRA. In particular, we show that copyless CRA are not closed under reverse operation, namely, there exists a function $f$ defined by a copyless CRA whose reverse function $f^r$ (i.e. $f^r(w) = f(w^r)$ where $w^r$ is the reverse word of $w$) is not definable by any copyless CRA. This unpleasant property of copyless CRA shows that the computational model is asymmetric with respect to the orientation of the input. This implies that copyless CRA do not enjoy good closure properties and, moreover, a potential logical characterization of the class of function defined by copyless CRA seems unlikely.

The non-expressibility result of copyless CRA explained above motivates the introduction of subclass of copyless CRA that has good closure properties. This subclass is called bounded alternation copyless CRA (BAC-CRA). We show that the bounded alternation restriction of copyless-CRA imposed good closure properties. In particular, we show that BAC-CRA are closed under regular look-ahead, a desirable property for any one way computational model (i.e. it is symmetric with respect to the future). The proof and construction that shows that BAC-CRA is closed under regular look-ahead is non-trivial and it requires
the introduction of an internal data structure in order to keep parallel runs with finite memory and copyless registers.

The paper is organized as follows: in Section 2 and 3 we give the background and definitions of copyless CRA with some examples. In Section 4 we show our structural results regarding copyless CRA like its normal form and stable registers. Then in Section 5 we study the expressiveness of copyless CRA and show that they are not closed under reverse. In Section 6 we introduce the class of bounded alternation copyless CRA and show that they are close under regular look-ahead.

## 2 Preliminaries

In this section, we summarise the notation and definitions used for regular languages, finite automata, and weighted automata.

**Finite automata.** A finite automaton [7] over \( \Sigma^* \) is a tuple \( \mathcal{A} = (Q, \Sigma, \delta, q_0, F) \) where \( Q \) is a finite set of states, \( \delta \in Q \times \Sigma \times Q \) is a finite transition relation, \( q_0 \) is the initial state and \( F \) is the final set of states. A run \( \rho \) of \( \mathcal{A} \) is a sequence of transitions of the form:

\[
p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} p_n
\]

where \((p_i, a_{i+1}, p_{i+1}) \in \delta \) for every \( i \leq n \). We say that \( \rho \) (like above) is a run of \( \mathcal{A} \) over a word \( w = a_1 \ldots a_n \) if \( p_0 = q_0 \). Furthermore, we say that \( \rho \) is an accepting run if \( p_n \in F \). A word \( w \) is accepted by \( \mathcal{A} \) if there exists an accepting run of \( \mathcal{A} \) over \( w \). We denote \( L(\mathcal{A}) \) the language of all words accepted by \( \mathcal{A} \).

A finite automaton \( \mathcal{A} \) is called deterministic if \( \delta \) is a function \( \delta : Q \times \Sigma \to Q \). When \( \mathcal{A} \) is deterministic, we extend the function \( \delta \) from letters to words and we denote this extension by \( \delta^* : Q \times \Sigma^* \to Q \), i.e., \( \delta^*(q, \epsilon) = q \) and \( \delta^*(q, w \cdot a) = \delta(\delta^*(q, w), a) \).

**Semirings and functions.** A semiring is a structure \( S = (S, \oplus, \otimes, 0, 1) \) where \((S, \oplus, 0)\) is a commutative monoid, \((S, \otimes, 1)\) is a monoid, multiplication distributes over addition, and \(0 \otimes s = s \otimes 0 = 0\) for each \( s \in S\). If the multiplication is commutative, we say that \( S \) is commutative. In this paper, we always assume that \( S \) is commutative. For the sake of simplicity, we usually denote the set of elements \( S \) by the name of the semiring \( S \). As standard examples of semi-rings we will consider the **semi-ring of natural numbers** \( \mathbb{N}(+,\cdot) = (\mathbb{N},+,\cdot,0,1) \), the **min-plus semi-ring** \( \mathbb{N}_\infty(\min,+) = (\mathbb{N}_\infty,\min,\cdot,\infty,0) \) and the **max-plus semi-ring** \( \mathbb{N}_{-\infty}(\max,+) = (\mathbb{N}_{-\infty},\max,\cdot,-\infty,0) \) which are standard semi-rings in the field of weighted automata [5].

We denote by \( \mathcal{F} \) the class of all functions from \( \Sigma^* \) to \( S \). We say that a function \( f \in \mathcal{F} \) is definable by a computational system \( \mathcal{A} \) (i.e. weighted automata, CRA) if \( f(w) = [\mathcal{A}](w) \) for any \( w \in \Sigma^* \) where \([\mathcal{A}]\) is the semantics of \( \mathcal{A} \) over words.

For any word \( w \), let \( w^r \) denote the reverse string. We say that a class of functions \( F \subseteq \mathcal{F} \) is **closed under reverse** [2] if for every \( f \in F \) there exists a function \( f^r \in F \) such that \( f^r(w) = f(w^r) \) for all \( w \in \Sigma^* \).
**Weighted automata.** Fix now a semiring $\mathbb{S}$ and a finite alphabet $\Sigma$. A *weighted automaton* over $\Sigma$ [13,5] is a tuple $A = (Q, \Sigma, E, I, F)$ where $Q$ is a finite set of states, $E : Q \times \Sigma \times Q \rightarrow \mathbb{S}$ is a weighted transition relation, and $I, F : Q \rightarrow \mathbb{S}$ is the initial and final function, respectively. Usually, if $E(p, a, q) = s$, we denote this transition graphically by $p \xrightarrow{a,s} q$. A *run* $\rho$ of $A$ is a sequence of transitions:

$$\rho = q_0 \xrightarrow{w_1/s_1} q_1 \xrightarrow{w_2/s_2} \cdots \xrightarrow{w_n/s_n} q_n,$$

where $w_i \neq 0$ for all $i \leq n$. We say that $\rho$ is a run of $A$ over a word $w = a_1a_2 \ldots a_n$ if it also holds that $I(q_0) \neq 0$. Moreover, a run $\rho$ like above is *accepting* if $I(q_0) \neq 0$ and $F(q_n) \neq 0$. In this case, the *weight* of an accepting run $\rho$ of $A$ over $w$ is defined by $|\rho| = I(q_0) \odot \prod_{i=1}^{n} s_i \odot F(q_n)$. We define $\text{Run}_A(w)$ as the set of all accepting runs of $A$ over $w$. Finally, the weight of $A$ over a word $w$ is defined by

$$\mathcal{L}(A)(w) = \sum_{\rho \in \text{Run}_A(w)} |\rho|$$

where the sum is equal to 0 if $\text{Run}_A(w)$ is empty. The set $\mathcal{L}(A) = \{ w \in \Sigma^* | \text{Run}_A(w) \neq \emptyset \}$ is called the *language* of $A$.

A weighted automaton $A$ is called *unambiguous* if $|\text{Run}_A(w)| \leq 1$ for every $w \in \Sigma^*$ and is called *finitely ambiguous* if there exists a uniform bound $N$ such that $|\text{Run}_A(w)| \leq N$ for every $w \in \Sigma^*$ [15,8]. Furthermore, $A$ is called *polynomial ambiguous* if the function $|\text{Run}_A(.)|$ is bounded by a polynomial on the input length [15,8]. For the special case when the number of runs of $A$ are bounded by a linear function, we say that $A$ is *linear ambiguous*.

**Trim assumption.** For technical reasons, in this paper we assume our finite automata, weighted automata or cost register automata (see Section 3) are always *trimmed*, namely, all its states are reachable from some initial states (i.e., they are accessible) and they can reach some final states (i.e., they are co-accessible). It is worth noticing that, since verifying if a state is accessible or co-accessible reduced to checking reachability in the transition graph [12] and this can be done in NLOGSPACE, then we can assume without lost of generality that all our automata are trimmed.

### 3 Copyless cost register automata

In this paper we study the expressibility of copyless cost register automata. Cost register automata (CRA) were introduced in [2] as an alternative model for computing functions over words. In contrast with weighted automata, CRA is deterministic model based on register. An advantage of CRA with respect to weighted automata is that it allows a fine grained understanding on how functions are computed. For example, the copyless-restriction (defined below) constrains a CRA to only use a register once per transition. This kind of restrictions admit a better understanding of subclasses of function that are included in the hierarchy of weighted automata but they cannot be naturally defined.
In this section, we recall the definitions of cost register automata and the copyless restriction. For this, we need to also give the definitions of expressions and substitutions over a semiring that are standard in the area.

**Variables, expressions, and substitutions.** Fix a semiring \( S = (\mathbb{S}, \circ, \otimes, 0, 1) \) and a set of variables \( \mathcal{X} \) disjoint from \( S \). We denote by \( \text{Expr}(\mathcal{X}) \) the set of all syntactical expressions that can be defined from \( \mathcal{X} \), constants in \( S \), and the syntactical signature of \( S \). An expression in \( \text{Expr}(\mathcal{X}) \) is basically a formula in \( S \) where variables in \( \mathcal{X} \) are missing values. For any expression \( e \in \text{Expr}(\mathcal{X}) \) we denote by \( \text{Var}(e) \) the set of variables in \( e \). We call an expression \( e \in \text{Expr}(\mathcal{X}) \) without variables (i.e. \( \text{Var}(e) = \emptyset \)) a **ground** expression. For any ground expression we define \([e] \in \mathbb{S}\) to be the evaluation of \( e \) with respect to \( S \).

A **substitution** over \( \mathcal{X} \) is defined as a mapping \( \sigma : \mathcal{X} \to \text{Expr}(\mathcal{X}) \). We denote the set of all substitutions over \( \mathcal{X} \) by \( \text{Subs}(\mathcal{X}) \). A **ground substitution** \( \sigma \) is a substitution where each expression \( \sigma(x) \) is ground for each \( x \in \mathcal{X} \). Any substitution can be extended to a mapping \( \tilde{\sigma} : \text{Expr}(\mathcal{X}) \to \text{Expr}(\mathcal{X}) \) such that, for every \( e \in \text{Expr}(\mathcal{X}) \), \( \tilde{\sigma}(e) \) is the resulting expression \( e[\sigma] \) of substituting each \( x \in \text{Var}(e) \) by the expression \( \sigma(x) \). For example, if \( \sigma(x) \) := \( 2x \) and \( \sigma(y) = 3y \), and \( e = x + y \), then \( \tilde{\sigma}(e) = 2x + 3y \). By using the extension \( \tilde{\sigma} \), we can define the composition substitution \( \sigma_1 \circ \sigma_2 \) of two substitution \( \sigma_1 \) and \( \sigma_2 \) such that \( \sigma_1 \circ \sigma_2(x) = \tilde{\sigma}_1(\sigma_2(x)) \) for each \( x \in \mathcal{X} \).

A valuation is defined as a substitution of the form \( \nu : \mathcal{X} \to \mathbb{S} \). We denote the set of all valuations over \( \mathcal{X} \) by \( \text{Val}(\mathcal{X}) \). Clearly, any valuation \( \nu \) composed with a assignment \( \sigma \) defined an expression without variables that can be evaluated as \( [\nu \circ \sigma(x)] \) for any \( x \in \mathcal{X} \).

In this paper, we say that two expressions \( e_1 \) and \( e_2 \) are equal (denoted by \( e_1 = e_2 \)) if they are equal up to evaluation equivalence, that is, \( [\nu \circ e_1] = [\nu \circ e_2] \) for every valuation \( \nu \in \text{Val}(\mathcal{X}) \). Similarly, we say that two substitutions \( \sigma_1 \) and \( \sigma_2 \) are equal (denoted by \( \sigma_1 = \sigma_2 \)) if \( \sigma_1(x) = \sigma_2(x) \) for every \( x \in \mathcal{X} \).

**Copyless restriction.** We say that an expression \( e \in \text{Expr}(\mathcal{X}) \) is **copyless** if for all \( x \in \mathcal{X} \), \( \sigma(x) \) uses every variable from \( \mathcal{X} \) at most once. For example, \( x : (y + z) \) is copyless but \( x \cdot y + x \cdot z \) is not copyless (i.e. \( x \) is mentioned twice). Notice that the copyless restriction is a syntactical constraint over expressions. Furthermore, we say that a substitution \( \sigma \) is copyless if \( \sigma(x) \) is a copyless expression for every \( x \in \mathcal{X} \) and \( \text{Var}(\sigma(x)) \cap \text{Var}(\sigma(y)) = \emptyset \) for every different pair \( x, y \in \mathcal{X} \). Similar than for expressions, copyless substitutions are restricted in such a way that each variable is used at most once over the whole substitution.

The following technical result will be used in Section 5 and the reader can avoid the proof for a first read. It shows the form of a copyless expression when it is rewrite as a sum of monomials.

**Lemma 1.** For any copyless expression \( e \), there exist an equivalent expression \( e' \) of the form:

\[
e' = \bigoplus_{i=1}^{k} \left( c_i \circ x_i \right)
\]
where \( X_1, \ldots, X_k \) is a sequence of different sets over \( \mathcal{X} \) and \( c_1, \ldots, c_k \) is a sequence of values over \( S \) for \( k \geq 0 \).

**Proof.** The lemma is shown by induction on the size of \( e \). For the base case, when \( e \) is equal to a constant or a variable, the lemma trivially holds by taking \( e' = e \). For the inductive case, suppose that \( e = e_1 \otimes e_2 \) where \( \otimes \) is either \( \oplus \) or \( \odot \). By the inductive hypothesis we know that there exist expressions \( e'_1 \) and \( e'_2 \) equivalent to \( e_1 \) and \( e_2 \), respectively, such that for \( j \in \{1, 2\} \):

\[
e'_j \equiv \bigoplus_{i=1}^{k_i} \left( c'_i \otimes_{x \in X'_i} x \right)
\]

where \( X'_1, \ldots, X'_k \) is a sequence of different sets over \( \mathcal{X} \) and \( c'_1, \ldots, c'_k \) is a sequence of values over \( S \) for \( k_j \geq 0 \). Given that \( e \) is a copyless expression, then we know that \( \text{Var}(e_1) \cap \text{Var}(e_2) \). Without lost of generality, we can assume that \( X'_j \subseteq \text{Var}(e_j) \) for \( j \in \{1, 2\} \) and \( i \leq k_j \). If not and there exists \( X'_i \neq \text{Var}(e_j) \), then this implies that \( c'_i \) must be equal to \( \emptyset \) (i.e. \( X'_i \) is not contributing in \( e'_j \)) and we can omit \( X'_i \) in \( e'_j \). Then since \( X'_i \subseteq \text{Var}(e_j) \) and \( \text{Var}(e_1) \cap \text{Var}(e_2) \), this implies that:

\[
X'_{i_1} \cap X'_{i_2} = \emptyset \quad \text{for every } i_1 \leq k_1 \text{ and } i_2 \leq k_2.
\]  

(1)

Now we consider when \( \otimes \) is either \( \oplus \) or \( \odot \). If \( e = e_1 \oplus e_2 \), then by considering \( e' = e_1 \odot e_2 \) the lemma is proved given that by (1) all sets of the form \( X'_i \) are different for \( j \in \{1, 2\} \) and \( i \leq k_j \). Otherwise, \( e = e_1 \oplus e_2 \) and we get:

\[
e'_1 \odot e'_2 = \bigoplus_{i_1=1}^{k_1} \left( c'_{i_1} \odot_{x \in X'_{i_1}} x \right) \odot \bigoplus_{i_2=1}^{k_2} \left( c'_{i_2} \odot_{x \in X'_{i_2}} x \right)
\]

\[
= \bigoplus_{i_1=1}^{k_1} \bigoplus_{i_2=1}^{k_2} \left( c'_{i_1} \odot_{x \in X'_{i_1} \cup X'_{i_2}} x \right) = e'
\]

The last derivation holds by (1), i.e., we do not need to consider repetitions in the multiplication of two monomials since \( X'_{i_1} \cap X'_{i_2} = \emptyset \). One can also check that all sets of the form \( X'_{i_1} \cup X'_{i_2} \) are different. Indeed, all sets \( X'_{i_1} \) are pairwise different and the same holds for \( X'_{i_2} \). This means by (1) that all sets \( X'_{i_1} \cup X'_{i_2} \) must be different as well. Then \( e' \) is equivalent to \( e \) and it has the form stated in the lemma.

\( \square \)

**Cost register automata.** A cost register automaton (CRA) over a semiring \( S \) is a tuple \( A = (Q, \Sigma, \mathcal{X}, \delta, q_0, \nu_0, \mu) \) where \( Q \) is a set of states, \( \Sigma \) is the input alphabet, \( \mathcal{X} \) is a set of variables, \( \delta : Q \times \Sigma \to Q \times \text{Subs}(\mathcal{X}) \) is the transition function, \( q_0 \) is the initial state, \( \nu_0 : \mathcal{X} \to S \) is the initial valuation, and \( \mu : Q \to \text{Expr}(\mathcal{X}) \) is the final output function. A configuration of \( A \) is a tuple
where \( q \in Q \) and \( \nu \in \text{Val}(A) \) represents the current values in the variables of \( A \). Given a string \( w = a_1 \ldots a_n \in \Sigma^* \), the run of \( A \) over \( w \) is a sequence of configurations: 
\[
(q_0, \nu_0) \xrightarrow{a_1} (q_1, \nu_1) \xrightarrow{a_2} \ldots \xrightarrow{a_n} (q_n, \nu_n)
\]
such that, for every \( 1 \leq i \leq n \), 
\[\delta(q_{i-1}, a_i) = (q_i, \sigma_i) \text{ and } \nu_i(x) = \left[ \nu_{i-1} \circ \sigma_i(x) \right] \text{ for each } x \in X.\]
The output of \( A \) over \( w \), denoted by \( [A](w) = \left[ \hat{\nu}_n(\mu(q_n)) \right] \).

A CRA \( A \) is called copyless if for any transition \( \delta(q_1, a) = (q_2, \sigma) \) and for any state \( q \in Q \), it holds that \( \sigma \) is a copyless substitution and \( \mu(q) \) is a copyless expression. In other words, every time that registers from \( A \) are operated, they can be use just once. In the following, we give some examples of copyless CRA and compare their expressiveness with weighted automata.

Example 1. Let \( S \) be the max-plus semiring \( \mathbb{N}_{\infty}(\max, +) \) and \( \Sigma = \{a, b\} \). Consider the function \( f_1 \) that given a word \( w \in \Sigma^* \) it computes the longest substring of \( b \)'s. This can be easily defined by the following CRA \( A_1 \) with two register \( x \) and \( y \).

\[
\begin{align*}
    &a \quad x := 0 \quad y := \max\{x, y\} \\
    &b \quad x := x + 1 \quad y := y \\
    &\text{max}\{x, y\}
\end{align*}
\]

\( A_1 \) stores in the \( x \)-register the length of the last suffix of \( b \)'s and in the \( y \)-register the length of the longest substring of \( b \)'s seen so far. After reading a \( b \)-letter \( A_1 \) updates \( x \) by one (i.e. the \( b \)-suffix has increased by one) and it keeps \( y \) unchanged. Furthermore, after reading a \( a \)-letter it resets \( x \) to zero and updates \( y \) by comparing the substring of \( b \)'s that has just finished (i.e. previous \( x \)-content) with the length of the longest substring of \( b \)'s (i.e. previous \( y \)-content) that has been seen so far. Finally, it outputs the maximum between \( x \) and \( y \).

One can easily check that the previous CRA satisfies the copyless restriction and, therefore, it is a copyless CRA. Indeed, each substitution is copyless and the final output expression \( \max\{x, y\} \) is copyless as well.

In [2] it was shown that copyless CRA contains the class of all unambiguous weighted automata (UWA), a robust subclass of weighted automata that has many characterizations and good decidability properties [8,9]. One can easily show that the class of copyless CRA contains also the class of finitely ambiguous weighted automata (FAWA) [15,8], a subclass that is more expressive than UWA [8] and still has good decidability properties [6]. Interestingly, the previous example shows that copyless CRA strictly contains the class of FAWA. Indeed, it was shown in [8] that \( f_1 \) is a function that is not definable by any FAWA. This means that copyless CRA are strictly more expressive than FAWA.

What functions are definable by copyless CRA? The previous example suggest that copyless CRA could contain the class of linear ambiguous weighted au-
tomata given that \( f_1 \) is definable by a weighted automata whose non-determinism is linearly bounded with respect to the size of the input. Unfortunately, this does not hold in general as we will show in Section 5: there exists linear ambiguous weighted automata that are not definable by any copyless CRA.

The previous discussion suggests the following question: are copyless CRA contained in the class of linearly ambiguous weighted automata? The next example shows that the answer to this question is negative since there exists functions definable by copyless CRA that are not definable by linear ambiguous weighted automata.

**Example 2.** Again, let \( S \) be the max-plus semiring \( \mathbb{N}_{\infty}(\max, +) \) and \( \Sigma = \{a, b, \#\} \). Consider the function \( f_2 \) such that, for any \( w \in \Sigma^* \) of the form \( w_0\#w_1\#\ldots\#w_n \) with \( w_i \in \{a, b\}^* \), it computes the maximum number of \( a \)'s or \( b \)'s for each substring \( w_i \) (i.e. \( \max\{w_i|_a, w_i|_b\} \)) and then it sums these values over all substrings \( w_i \), that is, \( f_2(w) = \sum_{i=0}^n \max\{w_i|_a, w_i|_b\} \). One can check that the copyless CRA \( A_2 \) define below computes \( f_2 \):

\[
\begin{align*}
\# & \quad x, y := 0 \\
& \quad z := z + \max\{x, y\} \\

x, y, z := 0 & \quad a \quad x := x + 1 \\
& \quad b \quad y := y + 1 \\
& \quad z + \max\{x, y\}
\end{align*}
\]

In the above diagram of \( A_2 \), we omit an assignment if a register is not updated (i.e. it keeps its previous value). For example, for the \( a \)-transition we omit the assignments \( y := y \) and \( z := z \) for the sake of presentation of the CRA. Similarly, we also omit the assignment \( x := x \) and \( z := z \) for the \( b \)-transition.

The copyless CRA \( A_2 \) follows similar ideas than \( A_1 \): the \( x \)- and \( y \)-register counts the number of \( a \)'s and \( b \)'s, respectively, of the longest suffix without \( \# \) and \( z \)-register stores the partial output without considering the last suffix of \( a \)'s and \( b \)'s. When the last substring \( w_i \) over \( \{a, b\} \) finishes (i.e. it comes a \( \# \)-letter or the input ends), then \( A_2 \) sums the maximum number of \( a \)'s or \( b \)'s in \( w_i \) to \( z \) (i.e. \( z := z + \max\{x, y\} \)).

Function \( f_2 \) is a counterexample for showing that copyless CRA are contained in the class of linear ambiguous weighted automata. In \([8]\) it was shown that \( f_2 \) is definable by a weighted automata but it is not definable by any polynomial weighted ambiguous automata. Interestingly, the previous example shows that copyless CRA can defined \( f_2 \) and, therefore, they can defined functions that are beyond the scope of polynomial ambiguous automata.
4 Structural properties of copyless CRA

In this section we develop some machinery that will be useful in Section 5 when we show the non-expressibility of copyless CRA. In Section 4.1 we show that copyless CRA can be assumed to do not contain the 0-constants in any transition, initial valuation or final function. Then, in Section 4.2, we propose a normal form for copyless CRA and, in Section 4.3, we highlight some special subset of registers (called stable registers) that are fundamental to understand the run of a copyless CRA. Finally, in Section 4.4 we study the asymptotic growing of the register content of a copyless CRA when a cycle is iterated.

Most of the results in this section are technical and the reader can skip most of the proofs for a first read. However, we would like to highlight the normal form of copyless CRA (Section 4.2) and the results on stable registers (Section 4.3) which are of independent interest from the non-expressibility result.

4.1 Removing zeros from CRA

We say that an expression \( e \in \text{Expr}(X) \) is reduced if \( e = 0 \) or the 0-constant is not mentioned inside \( e \). It is straightforward to show that for any expression \( e \) there exists an equivalent expression \( e^* \) that is reduced. Indeed, one can construct inductively an equivalent expression by using the following reductions: \( e \oplus 0 = e \) and \( e \odot 0 = 0 \). Then by reducing each subexpression recursively, the resulting expression is either 0 or do not use the 0-constant at all. Further, note that if \( e \) is copyless, then its reduced expression \( e^* \) is copyless as well.

Let \( f : \Sigma^* \rightarrow S \) be a function definable by a copyless CRA. We say that \( f \) is a non-zero function if \( f(w) \neq 0 \) for all \( w \in \Sigma^* \). The following result shows that, without lost of generality, we can assume that all constants in a copyless CRA are different from 0 whenever the function defined by \( A \) is a non-zero function.

**Proposition 1.** Let \( A \) be a copyless CRA such that \( [A] \) is a non-zero function. Then there exists a copyless CRA \( A' \) such that its initialization function, substitutions and final output functions are reduced and different from 0.

**Proof.** In this proof, we show how to avoid keeping a 0 value in registers that where forced to be 0 (i.e. forced by a substitution \( \sigma \) such that \( \sigma(x) = 0 \) for some register \( x \)). The idea is to store in the state which registers are equal to 0. Let \( A = (Q, \Sigma, X, \delta, q_0, \nu_0, \mu) \) be a copyless CRA. Define a new copyless CRA \( A' = (Q', \Sigma, X', \delta, q'_0, \nu'_0, \mu') \) such that:

- \( Q' = Q \times 2^Q \) is the new set of states,
- \( q'_0 = (q_0, S_0) \) where \( S_0 \) is the set of registers such that \( x \in S \) iff \( \nu_0(x) = 0 \), and
- \( \nu'_0 \) is the same as \( \nu_0 \) for registers in \( X \setminus S_0 \) and for registers \( x \in S_0 \) we define \( \nu'_0(r) = 1 \) (or any other constant from \( S \)).
For the definition of $\delta'$ and $\mu'$ we need to first introduce some notation. Let $S \subseteq X$ and $e$ be an expression over $X$. We define the expression $e[S]$ that is a reduced expression $d^\ast$ where $d$ is the result of taking $e$ and replacing all registers $x \in S \cap \text{Var}(e)$ by $0$.

Now, we are ready to define $\delta'$ and $\mu'$. For every $(q, S) \in Q'$ and $a \in \Sigma$, we define $\delta'((q, S), a) = ((q', S'), \sigma')$, where $\delta(q, a) = (q', \sigma)$, $S'$ is the set of all registers $x$ such that $\sigma(x)[S]$ is equal to $0$, and $\sigma'$ is defined for every $x \in X$ as follows:

$$\sigma'(x) = \begin{cases} \sigma(x)[S] & \text{if } x \notin S \\ 1 & \text{if } x \in S \end{cases}$$

Finally, we define the output function $\mu'$ such that, for every $(q, S) \in Q'$, it holds that $\mu'((q, S)) = \mu(q)[S]$.

It is straightforward to show that $A'$ and $A$ define the same function and that $\nu'_0$, $\delta'$ and $\mu'$ do not use the $0$-constant. Note that there cannot exist a reachable state $(q, S) \in Q'$ with $\mu'((q, S)) = 0$, otherwise the output of the function defined by $A$ will be $0$ for an input in $\Sigma^*$ which contradict the fact that $\llbracket A \rrbracket$ is a non-zero function.

### 4.2 The normal form of a copyless CRA

In this subsection, we define a normal form for copyless CRA given a linear order on registers. Let $A = (Q, \Sigma, X, \delta, q_0, \nu_0, \mu)$ be a copyless CRA and let $\preceq$ be any predefined linear order over $X$. We say that $A$ is in normal form with respect to $\preceq$ if for every transition of the form $\delta(p, a) = (q, \sigma)$, it holds that $x \preceq y$ for all $y \in \text{Var}(\sigma(x))$. In other words, all the variables mentioned in $\sigma(x)$ are greater or equal than $x$ with respect to $\preceq$. For example, the copyless CRA in Example 1 is in normal form with respect to the order $y \preceq x$.

The idea behind this normal form is to prevent unexpected behaviors on copyless CRA. For example, consider the following copyless CRA that is not in normal form with respect to the order $x \preceq y$:

```plaintext
\begin{align*}
a & \quad x := x + 1 \\ y := y 
\end{align*}

\begin{align*}
b & \quad x := y + 1 \\ y := x 
\end{align*}
```

The initial state in the above copyless CRA is $q_1$ and the initial valuation $\nu_0$ is equal to $\nu_0(x) = \nu_0(y) = 0$. Here, registers $x$ and $y$ count the number of $a$’s and $b$’s, respectively. However, depending on the state both registers can have either the number of $a$’s or the number of $b$’s. It is clear that one would like to avoid
this type of behaviour since this cyclic information can be easily stored in finite memory. Intuitively, one register should always contain the number of $a$’s and the other register the number of $b$’s.

In the next result, we show that for every copyless CRA there always exists an equivalent copyless CRA in normal form. The idea behind the proof is to permute the substitutions for registers and keep in the state the reverse permutation to encode the substitutions. Comming back to our running example, the normal form of this copyless CRA is the following:

From now on we assume that copyless cost-register automata are given with a linear order on the registers so we can order the registers, i.e., write $X = \{x_1, \ldots, x_n\}$ with the order defined by $x_i \preceq x_j$ iff $i \leq j$. The next proposition shows that every copyless CRA can be transformed into an equivalent copyless CRA in normal form with respect to $\preceq$.

**Proposition 2.** For every copyless CRA $A$ there exists a copyless CRA in normal form $A'$ such that they recognize the same function. Moreover $A'$ and $A$ have the same set of registers and the same linear order on them. The number of states in $A'$ can be bounded exponentially in the size of the automaton $A$.

**Proof.** Let $A = (Q, \Sigma, X, \delta, q_0, \nu, \mu)$ be a copyless CRA. We define a copyless CRA $A'$ in normal form such that $A'$ computes the same function as $A$. The idea of $A'$ is to store a state $q \in Q$ and a permutation $\rho$ of the set $X$ such that, if $((q, \nu))$ is the current configuration of $A$ over an input $w$, then $((q, \rho), \nu')$ is the configuration of $A'$ over $w$ and $\nu'(x) = \nu(\rho(x))$. In other words, the content in $\nu$ is still in $\nu'$ but the value $\nu(x)$ of $x$ is now in the register $\rho(x)$ for every $x \in X$. The permutation of registers’ content will allow us to keep the normal form in $A'$.

Formally, let $A' = (Q', \Sigma, X, \delta', q'_0, \nu'_0, \mu')$ where:

$$Q' = Q \times \{\rho \mid \rho \text{ is a permutation of the set } X\}$$

is the set of states, $q'_0 = (q_0, \text{id})$ is the initial state where id is the identity permutation, and $\nu'_0 = \nu_0$ is the initial function. For the sake of presentation, let us show how the run of $A'$ will correspond to the run of $A$ before going into the definition of the transition function $\delta'$ and the output function $\mu'$. For an expression $e \in \text{Expr}(X)$ and a permutation $\rho$ over $X$, we define $\rho(e)$ to be
the expression $e$ where the registers are replaced according to $\rho$. Let $(q, \nu)$ and $(q', \nu')$ be the configuration of $A$ and $A'$, respectively, after reading $w \in \Sigma^*$. Then we will show that:

$$q' = (q, \rho) \text{ for some permutation } \rho \text{ and } \nu(x) = \nu'(\rho(x)) \text{ for every } x \in \mathcal{X}. \quad (2)$$

Note that, for the word $\epsilon$, this correspondence holds since $(q_0, \nu_0)$ and $((q_0, \text{id}), \nu'_0)$ are the initial configuration of $A$ and $A'$, respectively, and $\nu'_0(\text{id}(x)) = \nu'_0(x) = \nu_0(x)$. Furthermore, if we define $\mu'(((q, \rho)) = \rho(\mu(q))$ and show that (2) always holds, we will prove that $A$ and $A'$ computes the same function. Indeed, if the transition function $\delta'$ preserves (2), then

$$\nu'(\mu'((q, \rho))) = \nu'(\rho(\mu(q))) \quad (\text{by definition of } \mu')$$

$$= \nu(\mu(q)) \quad (\text{by Property (2)})$$

This will prove that the outputs of $A$ and $A'$ are the same. Therefore, in the rest of the proof we will show how to define $\delta'$ such that $A'$ is a copyless CRA in normal form and its definition satisfies (2).

Before defining $\delta'$, we need some additional definitions. For a copyless substitution $\sigma$ and a permutation $\rho$ both over $\mathcal{X}$, define the set $S_{\sigma, \rho} = \{ x \in \mathcal{X} \mid \text{Var}(\rho(\sigma(x))) = \emptyset \}$, that is, the set of all variables $x$ where $\rho(\sigma(x))$ is a non-constant expression. Further, define the set $S'_{\sigma, \rho} = \{ \text{min}\{\text{Var}(\rho(\sigma(x)))\} \mid x \in S_{\sigma, \rho} \}$ and the function $\tau^0_{\sigma, \rho} : S_{\sigma, \rho} \rightarrow S'_{\sigma, \rho}$ such that for all $x \in S_{\sigma, \rho}$:

$$\tau^0_{\sigma, \rho}(x) = \text{min}\{\text{Var}(\rho(\sigma(x)))\} \quad (3)$$

One can easily check that $\tau^0_{\sigma, \rho}$ is a bijective function from $S_{\sigma, \rho}$ to $S'_{\sigma, \rho}$. It is surjective by the definition of $S_{\sigma, \rho}$ and $S'_{\sigma, \rho}$, and injective by the copyless restriction over $\sigma$. To see the last claim, recall that any copyless substitution satisfies that $\text{Var}(\sigma(x)) \cap \text{Var}(\sigma(y)) = \emptyset$ and, in particular, $\text{Var}(\rho(\sigma(x))) \cap \text{Var}(\rho(\sigma(y))) = \emptyset$ for any permutation $\rho$. In other words, this means that:

$$\tau^0_{\sigma, \rho}(x) = \text{min}\{\text{Var}(\rho(\sigma(x)))\} \neq \text{min}\{\text{Var}(\rho(\sigma(y)))\} = \tau^0_{\sigma, \rho}(y)$$

for every pair $x, y \in S_{\sigma, \rho}$, that is, $\tau^0_{\sigma, \rho}$ is an injective function. Finally, given that $\tau^0_{\sigma, \rho}$ is a bijective function from $S_{\sigma, \rho} \subseteq \mathcal{X}$ to $S'_{\sigma, \rho} \subseteq \mathcal{X}$, we can extend $\tau^0_{\sigma, \rho}$ to a bijection $\tau_{\sigma, \rho} : \mathcal{X} \rightarrow \mathcal{X}$ such that $\tau_{\sigma, \rho}(x) = \tau^0_{\sigma, \rho}(x)$ for every $x \in S_{\sigma, \rho}$. Of course, there might be many alternatives for extending $\tau^0_{\sigma, \rho}$ into a bijective function $\tau_{\sigma, \rho}$, but we can choose any of these extensions (i.e. this decision is not important for the construction).

We have now all the ingredients to define the transition function $\delta'$. For every $p \in Q$, $a \in \Sigma$, and permutation $\rho$ over $\mathcal{X}$, if $\delta(p, a) = (q, \sigma)$ then we define $\delta'((p, \rho), a) = ((q, \tau_{\sigma, \rho}), \sigma')$ such that:

$$\sigma'(x) = \rho(\sigma(\tau^{-1}_{\sigma, \rho}(x)))$$

for every $x \in \mathcal{X}$. From the previous definition, we can easily check that $\sigma'$ is a copyless substitution. In fact, $\rho \circ \sigma$ is a copyless substitution for every copyless
This proves that the transition function \( \delta \) is just a renaming of variables and \( \tau_{\sigma,\rho}^{-1} \) is just permuting the variables. Therefore, we can conclude that \( \sigma' \) is copyless as well.

Our next step is to show that \( \sigma' \) is in normal form. Recall that \( \sigma' \) is in normal form if for every \( x \in \mathcal{X} \) it holds that \( x \not\leq y \) for every \( y \in \text{Var}(\sigma'(x)) \). We prove this by case analysis by considering whether \( x \) form if for every \( x \).

Therefore, we can conclude that \( \sigma' \) is copyless as well.

For the last part of the proof, we show by induction that \( \delta' \) satisfies the correspondence (2) between \( A \) and \( A' \). Let \((p, \nu_n)\) and \((\rho, \nu'_n)\) be the configuration of \( A \) and \( A' \), respectively, after reading \( w \in \Sigma^* \) and suppose that \( \nu_n(x) = \nu'_n(\rho(x)) \) for every \( x \in \mathcal{X} \) (i.e. inductive hypothesis). Furthermore, suppose that:

\[
(p, \nu_n) \xrightarrow{\mathcal{A}} (q, \nu_{n+1}) \quad \text{and} \quad ((\rho, \nu'_n) \xrightarrow{\mathcal{A}'}, ((q, \tau_{\sigma,\rho}), \nu'_{n+1})
\]

are the transitions for \( A \) and \( A' \), respectively, after reading a new letter \( a \in \Sigma \). We prove the correspondence (2) between \( \nu_{n+1} \) and \( \nu'_{n+1} \) as follows:

\[
\nu'_{n+1}(\tau_{\sigma,\rho}(x)) = \nu'_n(\sigma'(\tau_{\sigma,\rho}(x))) \quad \text{(by definition of } \nu'_{n+1})
\]
\[
= \nu'_n(\rho(\sigma(\tau_{\sigma,\rho}(x)))) \quad \text{(by definition of } \sigma')
\]
\[
= \nu'_n(\rho(\sigma(x))) \quad \text{(by composing } \tau_{\sigma,\rho} \text{ and } \tau_{\sigma,\rho}^{-1})
\]
\[
= \nu_n(\sigma(x)) \quad \text{(by inductive hypothesis)}
\]
\[
= \nu_{n+1}(x) \quad \text{(by definition of } \nu_{n+1})
\]

This proves that the transition function \( \delta' \) keeps the correspondence (2) between \( A \) and \( A' \). Since it also holds for the initial configuration then by induction this shows that it holds for all configurations which proves that the outputs of \( A \) and \( A' \) are the same.

Let \( A = (Q, \Sigma, \mathcal{X}, \delta, q_0, \nu_0, \mu) \) be a copyless CRA. As usual, we define the transitive closure of \( \delta \) as the function \( \delta^* : Q \times \Sigma^* \rightarrow Q \times \text{Subs}(\mathcal{X}) \) by induction over the word-length. Formally, we define \( \delta^*(q, \epsilon) = (q, \id) \) where \( \id \) is the identity substitution for all \( q \in Q \) and \( \delta^*(q_1, w \cdot a) = (q_1, \sigma \circ \sigma') \) whenever \( \delta^*(q_1, w) = (q_2, \sigma) \) and \( \delta(q_2, a) = (q_3, \sigma') \). For a CRA \( A \) we define the set \( \text{Subs}(A) \) of all substitutions in \( A \) as follows:

\[
\text{Subs}(A) = \{ \sigma \in \text{Subs}(\mathcal{X}) \mid \exists p, q \in Q. \exists w \in \Sigma^*. \delta^*(p, w) = (q, \sigma) \}
\]
It is easy to check that, if all substitutions in \( \delta \) are copyless, then all substitution in \( \text{Subs}(A) \) are also copyless. Furthermore, \( \delta^* \) and, in particular, \( \text{Subs}(A) \) also preserves the normal form, that is, for all \( \sigma \in \text{Subs}(A) \) it holds \( x \preceq y \) for all \( y \in \text{Var}(\sigma(x)) \). This can be easy proved by induction over the word-length. Assume that it holds for all words \( w \) of length at most \( n \) and let \( \delta^*(q, w) = (q', \sigma) \). Suppose we want to extend \( w \) with \( a \in \Sigma \) and \( \delta^*(q', a) = (q'', \sigma') \). By definition, we know that \( \delta^*(q, w\cdot a) = (q'', \sigma \circ \sigma'((x))) \). Then take \( y \in \text{Var}(\sigma \circ \sigma'(x)) \). By definition there exists a register \( z \) such that \( z \in \text{Var}(\sigma(x)) \) and \( y \in \text{Var}(\sigma(z)) \). Since \( \delta \) is in normal form, we conclude that \( x \preceq z \preceq y \). In other words, \( x \preceq y \) for every \( y \in \text{Var}(\sigma \circ \sigma'(x)) \) and, thus, \( \sigma \circ \sigma' \) is in normal form.

### 4.3 Stable registers and collapse substitutions

Let \( A = (Q, \Sigma, \mathcal{X}, \delta, q_0, \nu_0, \mu) \) be a copyless CRA in normal form with respect to a fix total order \( \preceq \). In a copyless CRA in normal form, the content of registers flows during a run from higher to lower register with respect to \( \preceq \). Unfortunately, this does not necessarily mean that the content of all registers will eventually reach the \( \preceq \)-minimum register. For example, if all substitutions in \( A \) are of the form \( \sigma(x) = x \oplus k \) for some \( k \in \mathbb{S} \), then each register will store just its own content during the whole run. Intuitively, in this example each register is “stable” with respect to the content flow of \( A \). We formalize this idea with the notion of stable registers. Let \( \sigma \) be a copyless substitution in normal form (i.e. for each \( x \in \mathcal{X} \), \( x \preceq y \) for all \( y \in \text{Var}(\sigma(x)) \)). We say that a register \( x \) is \( \sigma \)-stable (or stable on \( \sigma \)) if \( x \in \text{Var}(\sigma(x)) \).

The following lemma shows that the composition preserves stability between registers.

**Lemma 2.** Let \( \sigma, \sigma' \) be two copyless substitution in normal form. For any register \( x \in \mathcal{X} \) it holds that \( x \) is stable on \( \sigma \) and \( \sigma' \) if, and only if, \( x \) is \((\sigma \circ \sigma')\)-stable.

**Proof.** Suppose that \( x \) is stable on \( \sigma \) and \( \sigma' \). This means that \( x \in \text{Var}(\sigma(x)) \) and \( x \in \text{Var}(\sigma'(x)) \) and, thus, \( x \in \text{Var}(\sigma \circ \sigma'(x)) \) by composition. For the other direction, suppose that \( x \in \text{Var}(\sigma \circ \sigma') \). Then we know that there exists \( y \) such that \( x \in \text{Var}(\sigma(y)) \) and \( y \in \text{Var}(\sigma'(x)) \). Since \( \sigma \) and \( \sigma' \) are in normal form, then \( x \preceq y \preceq x \) which implies that \( x = y \) and, thus, \( x \) is stable on \( \sigma \) and \( \sigma' \). \( \square \)

We generalize the idea of stable register from substitutions to copyless CRA. We say that a register \( x \) is stable in \( A \) (or just stable if \( A \) is understood from the context) if \( x \) is \( \sigma \)-stable for all transitions \( \sigma \in \text{Subs}(A) \). A register is called non-stable if it is not stable. By Lemma 2, it is enough to check that a register \( x \) is \( \sigma \)-stable for all letter-transitions \( \delta(p, a) = (q, \sigma) \) to know whether \( x \) is stable in \( A \) or not.

For the next proposition, we need to recall some standard definitions of directed labeled graphs and bottom strongly connected components. For \( Q' \subseteq Q \) we say that \( Q' \) is a bottom strongly connected component (BSCC) of \( A \) if (1)
for every pair \( q_1, q_2 \in Q' \) there exists \( u \in \Sigma^* \) such that \( \delta^*(q_1, u) = (q_2, \sigma) \) for some substitution \( \sigma \) and (2) for every \( q \in Q' \) and \( w \in \Sigma^* \) if \( \delta^*(q_1, w) = (q_2, \sigma) \) then \( q_2 \in \bar{Q} \). Intuitively a BSCC \( Q' \) of \( A \) is a set of mutually reachable states such that there is no word that leaves \( Q' \). We say that \( A \) is strongly connected if the whole set \( Q \) is a BSCC of \( A \). In this section, we assume that \( A \) is strongly connected.

By the previous proposition, we know that for every pair of states \( q, q' \) there exists a word \( w^{q,q'} \) that "collapses" the content of non-stable register into stable register. This motivates the following definition: a substitution \( \sigma \in \text{Subs}(A) \) is said to be collapse substitution if \( \text{Var}(\sigma(x)) = \emptyset \) for all non-stable registers in \( \sigma \). With this definition, the previous lemma says that for every pair of states \( q, q' \) there exists a word \( w^{q,q'} \) such that \( \delta(q, w^{q,q'}) = (q', \sigma) \) and \( \sigma \) is a collapse substitution. From now on, if \( q = q' \), we write \( w^q \) and \( \sigma^q \) instead of \( w^{q,q} \) and \( \sigma^{q,q} \), respectively.

We say that a substitution \( \sigma \in \text{Subs}(A) \) is a collapse substitution in \( A \) if \( \text{Var}(\sigma(x)) = \emptyset \) for all non-stable registers in \( \sigma \). Intuitively, collapse substitution makes all non-stable registers to forget its content. In this sense, non-stable registers are unstable and they can loose its current content with a collapse substitution. Notice that by definition stable register can never loose its current content.

**Proposition 3.** Let \( A = (Q, \Sigma, \mathcal{X}, \delta, q_0, \nu_0, \mu) \) be a copyless and strongly connected CRA. Then for all \( q, q' \in Q \) there exists a word \( w^{q,q'} \) and a substitution \( \sigma^{q,q'} \) such that (1) \( \delta^*(q, w^{q,q'}) = (q', \sigma^{q,q'}) \), (2) \( w^{q,q'} \) contains each letter in \( \Sigma \), and (3) \( \sigma^{q,q'} \) is a collapse substitution in \( A \).

**Proof.** Let \( \mathcal{X} = \{x_1, \ldots, x_n\} \) be the registers of \( \mathcal{X} \) in increasing order with respect to \( \leq \) and \( q, q' \) two states in \( Q \). We construct the word \( w^{q,q'} \) by (inverse) induction starting from \( x_n \) and ending in \( x_1 \). Specifically, for every \( i \leq n \) we will define a word \( w_i^{q,q} = w_i^{q,q'} \) and a substitution \( \sigma_i^{q,q} = \sigma_i^{q,q'} \) such that the proposition holds for all non-stable register greater than \( x_i \). Clearly the proposition will be shown by considering \( w_i^{q,q'} = w_i^{q,q} \).

We start by the base case \( i = n \) and consider whether \( x \) is stable or not. If \( x_n \) is stable, then take a word \( u \) and a substitution \( \sigma \) such that \( \delta^*(q, u) = (q', \sigma) \) and \( u \) contains each letter in \( \Sigma \). Given that \( A \) is strongly connected we know that \( u \) and \( \sigma \) always exists. Then by defining \( w_i^{q,q'} = u \) and \( \sigma_i^{q,q'} = \sigma \), the proposition holds for the stable register \( x_n \). Now, suppose that \( x_n \) is non-stable which means that there exist a pair \( p, p' \in Q \) and a word \( u \) such that \( \delta^*(p, u) = (p', \sigma) \) and \( x_n \) is non-stable in \( \sigma \). Given that \( A \) is in normal form and \( x_n \) is the maximum register with respect to \( \leq \), this implies that \( \text{Var}(\sigma(x_n)) = \emptyset \). Pick two words \( v_1, v_2 \in \Sigma^* \) such that \( \delta^*(q, v_1) = (p, \sigma_1) \) and \( \delta^*(p', v_2) = (q', \sigma_2) \) for some substitutions \( \sigma_1 \) and \( \sigma_2 \), and \( v_1 \) contains each letter in \( \Sigma \). Again, we know that \( v_1 \) and \( v_2 \) always exists since \( A \) is strongly connected. Then define \( w_n^{q,q'} = v_1 \cdot u \cdot v_2 \) and \( \sigma_n^{q,q'} = \sigma_1 \cdot \sigma \cdot \sigma_2 \). By construction, we know that \( \delta^*(q, w_n^{q,q'}) = (q', \sigma_n^{q,q'}) \) and \( w_n^{q,q'} \) contains all letter in \( \Sigma \). For proving that \( \text{Var}(\sigma_n^{q,q'}(x_n)) = \emptyset \), notice that \( x \) is non-stable on
\(\sigma\). By Lemma 2 this implies that \(x\) is non-stable on \(\sigma_1 \circ \sigma \circ \sigma_2\). Given that \(\mathcal{A}\) is in normal form and \(x_n\) is the maximum register, we get that \(\text{Var}(\sigma(x_n)) = \emptyset\).

For the inductive step, we suppose that \(w^{q,q}_{i+1}\) and \(\sigma^{q,q}_{i+1}\) exist and show how to construct \(w^{q,q}_i\) and \(\sigma^{q,q}_i\) that satisfy the proposition for registers greater or equal than \(x_i\). Again, we consider whether \(x_i\) is stable or not. If \(x_i\) is stable, then by defining \(w^{q,q}_i = w^{q,q}_{i+1}\) and \(\sigma^{q,q}_i = \sigma^{q,q}_{i+1}\) the proposition trivially holds. Indeed, property (1) and (2) are satisfied by the inductive hypothesis and (3) is satisfied since it holds for every register greater than \(x_i\) (again by inductive hypothesis) and also for \(x_i\) given that \(x_i\) is stable. Thus, the interesting case is when \(x_i\) is non-stable. Suppose that \(x_i\) is non-stable and, therefore, there exist a pair \(p, p' \in Q\) and a word \(u\) such that \(\delta^*(p, u) = (p', \sigma)\) and \(x_i\) is non-stable on \(\sigma\). Let \(v_1, v_2 \in \Sigma^*\) such that \(\delta^*(q, v_1) = (p, \sigma_1)\) and \(\delta^*(p', v_1) = (q', \sigma_2)\) for some substitutions \(\sigma_1\) and \(\sigma_2\). Recall that \(v_1\) and \(v_2\) exist by assuming that \(\mathcal{A}\) is strongly connected. Now, define:

\[
\begin{align*}
\quad w^{q,q}_i &= w^{q,q}_{i+1} \cdot v_1 \cdot u \cdot v_2 \\
\quad \sigma^{q,q}_i &= \sigma^{q,q}_{i+1} \circ \sigma_1 \circ \sigma \circ \sigma_2
\end{align*}
\]

It is clear by construction that \(\delta^*(q, w^{q,q}_i) = (q', \sigma^{q,q}_i)\) and \(w^{q,q}_i\) contains all letter in \(\Sigma\). The last fact holds because we know (by induction) that \(w^{q,q}_{i+1}\) contains all letters in \(\Sigma\). To conclude the proof, we must show that \(\text{Var}(\sigma^{q,q}_i(x)) = \emptyset\) for every non-stable register \(x \geq x_i\). Let \(x\) be any non-stable register \(x \geq x_i\) (possibly \(x_i\)) and let \(\sigma^* = \sigma_1 \circ \sigma \circ \sigma_2\). First, note that all \(y \in \text{Var}(\sigma^*(x))\) are non-stable. Otherwise, if \(y \notin \text{Var}(\sigma^*(x))\) is stable, then \(y \in \text{Var}(\sigma^*(y))\) but this is impossible since \(\text{Var}(\sigma^*(x)) \cap \text{Var}(\sigma^*(y)) = \emptyset\) by the definition of being copyless. Therefore, we have that every register in \(\text{Var}(\sigma^*(x))\) is non-stable. Note also that \(x_i \notin \text{Var}(\sigma^*(x))\). This is true when \(x \neq x_i\) (i.e. \(\mathcal{A}\) is in normal form) and also true when \(x = x_i\) since we know that \(x_i \notin \text{Var}(\sigma^*(x_i))\) (i.e. \(x_i\) is non-stable). Then we have that all registers in \(\text{Var}(\sigma^*(x))\) are non-stable and strictly greater than \(x_i\). This means that the inductive hypothesis holds for \(i + 1\) and \(\text{Var}(\sigma^{q,q}_{i+1}(y)) = \emptyset\) for all \(y \in \text{Var}(\sigma^*(x))\). By composing \(\sigma^{q,q}_{i+1}\) and \(\sigma^*\), we conclude that \(\text{Var}(\sigma^{q,q}_i(x)) = \emptyset\). This concludes the proof.

### 4.4 Growing rate of stable registers in a loop

Fix a copyless and non-zero CRA \(\mathcal{A} = (Q, \Sigma, \mathcal{X}, q_0, q_0, \nu_0, \mu)\). Similar that in the previous section, assume also that \(\mathcal{A}\) is in normal form with respect to the total order \(\succeq\) and, furthermore, assume that \(\mathcal{A}\) is strongly connected. We start this section with a trivial fact about copyless expression that will be useful during this section.

**Lemma 3.** Let \(\sigma, \tau \in \text{Subs}(\mathcal{A})\) where \(\sigma\) is a collapse substitution and let \(x\) be a \(\sigma\)-stable and \(\tau\)-stable register. Then the expression \(\sigma \circ \tau(x)\) is equivalent to an expression of the form \((c \circ \sigma(x)) \oplus d\), where \(c, d \in \Sigma\) and \(c \neq 0\).
Lemma 4. Let $\sigma \in \text{Subs}(A)$ be any substitution. Then there exists $N \geq 0$ such that $\sigma^N$ is a collapse substitution in $A$.

Proof. Suppose $x$ is non-stable over $\sigma$, i.e. $x \notin \text{Var}(\sigma(x))$. Since $A$ is copyless and in normal form then $x < \min(\text{Var}(\sigma(x)))$. But since $\min(\text{Var}(\sigma(x))) \in \text{Var}(\sigma(x))$ and $A$ is copyless then $\min(\text{Var}(\sigma(x))) \notin \text{Var}(\sigma(\min(\text{Var}(\sigma(x)))))$ and thus $\min(\text{Var}(\sigma(x))) \notin \min(\text{Var}(\sigma^2(x)))$. Applying this argument consecutively we get a increasing sequence of registers:

$$x < \min(\text{Var}(\sigma(x))) < \min(\text{Var}(\sigma^2(x))) < \min(\text{Var}(\sigma^3(x))) < \ldots$$

The number of registers is finite and then this sequence cannot be infinite. Thus there exists an $N$ such that $\text{Var}(\sigma^N(x)) = \emptyset$. We conclude that it suffices to take $N = |\mathcal{X}|$.

Proposition 4. Let $\sigma \in \text{Subs}(A)$ be a collapsing substitution and $x$ be a $\sigma$-stable register. Then there exist $c, d \in S$ with $c \neq 0$ such that for every $i \geq 0$ we have:

$$\sigma^{i+1}(x) = (c^i \odot \sigma(x)) \oplus (d \odot \bigoplus_{j=0}^{i-1} c^j)$$
Proof. Since $\sigma$ is a copyless and collapsing substitution from $\mathcal{A}$, then for every $y \in \text{Var}(\sigma(x))$ either $y = x$ or $\text{Var}(\sigma(y)) = \emptyset$. Then consider the expression $e$ that is equal to $\sigma(x)$ where every register $y \neq x$ is replaced with $[\sigma(y)]$. This is a copyless expression with only one variable $x$ and $\mathcal{A}$ is non-zero. By Lemma 3 (i.e. by replacing $\tau$ by $\sigma$ and $\sigma$ by the identity substitution) it can be rewritten in the form $e^* = (c \odot x) \odot d$ for some $c, d \in S$ and $c \neq 0$.

We prove this claim by induction using the constants $c, d$. We start with the base step for $i = 1$ (for $i = 0$ is trivially true). The expression $\sigma \circ e$ can be rewritten as $\sigma \circ e^* = (c \odot \sigma(x)) \odot d$, because it takes the expression $\sigma$ and substitutes $x$ with $\sigma(x)$ and all other registers $x$ with $[[\tau(x)]$.

For the inductive step, we have that $\sigma^{i+1} = \sigma^{i+1} \circ e^*$ by the same argument used in the previous paragraph. Then by replacing $\sigma^{i+1}$ with the inductive hypothesis, we get:

$$
\begin{align*}
\sigma^{i+2}(x) &= (c \odot \sigma^{i+1}(x)) \odot d \\
&= (c \odot ((c^i \odot \sigma(x)) \odot (d \odot \bigoplus_{j=0}^{i-1} e^j))) \odot d \\
&= ((\sigma^{i+1} \odot \sigma(x)) \odot (d \odot \bigoplus_{j=0}^{i} e^j)) \odot d \\
&= (\sigma^{i+1} \odot \sigma(x)) \odot (d \odot \bigoplus_{j=0}^{i} e^j)
\end{align*}
$$

The previous proposition shows that stable registers grow exponentially with respect of the number of times than a loop is iterated. In particular, when $S = \mathbb{N}_{-\infty}(\max, +)$ a stable register grow linearly. The next result is a refinement of Proposition 4 but in terms of the $\mathbb{N}_{-\infty}(\max, +)$-semiring. The lemma is technical and it will be used in the next section. The reader can avoid the proof for a first read.

For the next lemma, recall that, by Proposition 3, for any $q \in Q$ there exists a word $w^q$ such that $\delta^*(q, w^q) = (q, \sigma^q)$ and $\sigma^q$ is a collapse substitution over every non-stable registers in $\mathcal{A}$.

**Lemma 5.** Let $\mathcal{A}$ be a copyless and non-zero CRA in normal form over the $\mathbb{N}_{-\infty}(\max, +)$ semiring. Furthermore, let $q \in Q$ and $v \neq \epsilon$ be a loop such that $\delta^*(q, v) = (q, \tau)$ for some collapsing substitution $\tau$. Then for $j \in \mathbb{N}$ big enough, there exists a substitution $\sigma_j$ such that $\delta^*(q, w^q \cdot v^j \cdot w^q) = (q, \sigma_j)$ where:

$$
\sigma_j \circ \lambda(x) = \begin{cases} 
O(1) & \text{if } x \text{ is non-stable} \\
\max \{ j \cdot c + \sigma^q(x) + O(1), j \cdot d + O(1) \} & \text{otherwise,}
\end{cases}
$$

where $\lambda \in \text{Subs}(\mathcal{A})$ such that its size does not depend on $j$; and $c, d \in \mathbb{N}$ are constants that depend on $x$ but not on $j$ or $\lambda$.

**Proof.** From now on we work with the $\mathbb{N}_{-\infty}(\max, +)$ semiring. Let $q \in Q$ and $v \neq \epsilon$ be a loop such that $\delta^*(q, v) = (q, \tau)$ for some collapsing substitution $\tau$. 

Recall that $\mathcal{A}$ is a non-zero CRA and in this semiring $0 = -\infty$, thus we can assume that $c \geq 0$. By Proposition 4, we have that for every $j \geq 1$ it holds that:

$$\tau^{j+1}(x) = \left(c_j \tau(x) \oplus \left( d \oplus \bigoplus_{k=0}^{j-1} c_x \right) \right)$$  \hspace{1cm} (by Prop. 4)

$$\quad = \max \left\{ j \cdot c_x + \tau(x), d_x + \max_{k=0}^{j-1} \{ k \cdot c_x \} \right\} \quad \text{(by definition of } \mathbb{N}_{-\infty}(\max,+))$$

$$\quad = \max \left\{ j \cdot c_x + \tau(x), \ max \{ d_x + (j-1) \cdot c_x \} \right\} \quad \text{(by definition of max)}$$

$$\quad = \max \left\{ j \cdot c_x + \tau(x) + \mathcal{O}(1), \ j \cdot c_x + \mathcal{O}(1) \right\} \quad \text{(by using the } \mathcal{O}-\text{notation)} \quad (4)$$

Fix any substitution $\lambda \in \text{Subs}(\mathcal{A})$. Recall that $w^q$ is the word and $\sigma^q$ is the substitution from Proposition 3 for $q$. Then for any $j \geq 1$, consider the substitution $\sigma_j$ such that $\delta^*(q, w^q \cdot \nu^{j+1}, w^q) = (q, \sigma_j)$. We show next that $\sigma_j \circ \lambda$ satisfies the above properties for any $j$. For a non-stable register $x$ in $\mathcal{A}$, the result is straightforward. Indeed, $\mathcal{A}$ is in normal form and, thus, $\text{Var}(\lambda(x))$ contains just non-stable registers. This implies that:

$$\sigma_j \circ \lambda(x) = \sigma^q \circ \tau^j \circ \sigma^q \circ \lambda(x) \quad \text{(by definition)}$$

$$\quad = \sigma^q \circ \lambda(x) \quad \text{(Var}(\lambda(x)) \text{ contains only non-stable registers)}$$

$$\quad = \mathcal{O}(1) \quad \text{(}\sigma^q \text{ and } \lambda \text{ do not depend on } j)$$

Suppose now that $x$ is a stable register in $\mathcal{A}$. We need to show that:

$$\sigma_j \circ \lambda(x) = \max \{ j \cdot c + \sigma^q(x) + \mathcal{O}(1), \ j \cdot d + \mathcal{O}(1) \} \quad (5)$$

for $j$ big enough and some constants $c, d \in \mathbb{N}$ such that $c, d$ do not depend on $\lambda$. The expression $\sigma_j \circ \lambda(x)$ is equivalent to $\sigma^q \circ \tau^{j+1} \circ \sigma^q \circ \lambda(x)$. In other words, it is equivalent to the expression $\sigma^q \circ \lambda(x)$ where all registers $y \in \text{Var}(\sigma^q \circ \lambda(x))$ are substituted with $\sigma^q \circ \tau^{j+1}(y)$. We start the proof by analyzing these last expressions.

Let $y \in \text{Var}(\sigma^q \circ \lambda(x))$ be any variable. If $y$ is not a $\tau$-stable register, then $\text{Var}(\tau(z)) = \emptyset$ and thus $\sigma^q \circ \tau^{j+1}(y) = \mathcal{O}(1)$. Otherwise, by (4) we know that $\tau^{j+1}(y) = \max \{ j \cdot c_y + \tau(x) + \mathcal{O}(1), \ j \cdot c_y + \mathcal{O}(1) \}$ for some $c_y \in \mathbb{N}$ and, thus, by applying $\sigma^q$ on $\tau^{j+1}(y)$ we get:

$$\sigma^q \circ \tau^{j+1}(y) = \max \{ j \cdot c_y + \sigma^q \circ \tau(x) + \mathcal{O}(1), \ j \cdot c_y + \mathcal{O}(1) \} \quad (6)$$

If $y$ is a $\tau$-stable register, but non-stable on $\mathcal{A}$ (i.e. non-stable in general) then $\sigma^q \circ \tau(y)$ is equal to a constant and does not depend on $j$. Thus we can estimate $\sigma^q \circ \tau(y)$ by $\mathcal{O}(1)$ and then (6) becomes:

$$\quad = \max \{ j \cdot c_y + \mathcal{O}(1), \ j \cdot c_y + \mathcal{O}(1) \}$$

$$\quad = j \cdot c_y + \mathcal{O}(1)$$
we know that \( \sigma^q \circ \tau(y) \) can be represented as:

\[
\sigma^q \circ \tau(y) = \max\{ c' + \sigma^q(y), \ d' \} = \max\{ \sigma^q(y) + \mathcal{O}(1), \ \mathcal{O}(1) \}
\]

for some constants \( c', d' \in \mathbb{N} \) not depending on \( j \) and where \( c' \neq -\infty \). Thus, by combining (6) with the previous equation we get:

\[
\sigma^q \circ \tau^{j+1}(y) = \max\{ j \cdot c_y + \sigma^q \circ \tau(x) + \mathcal{O}(1), \ j \cdot c_y + \mathcal{O}(1) \}
\]

(by (6))

\[
= \max\{ j \cdot c_y + \max\{ \sigma^q(y) + \mathcal{O}(1), \ \mathcal{O}(1) \} + \mathcal{O}(1), \ j \cdot c_y + \mathcal{O}(1) \}
\]

(by (7))

\[
= \max\{ j \cdot c_y + \sigma^q(y) + \mathcal{O}(1), \ j \cdot c_y + \mathcal{O}(1), \ j \cdot c_y + \mathcal{O}(1) \}
\]

(by distribution)

\[
= \max\{ j \cdot c_y + \sigma^q(y) + \mathcal{O}(1), \ j \cdot c_y + \mathcal{O}(1) \}
\]

(8)

Observe that a variable \( y \in \text{Var}(\sigma^q \circ \lambda(x)) \) is table if, and only if, \( y = x \). We summary the three possible cases in the next equation:

\[
\sigma^q \circ \tau^{j+1}(y) = \begin{cases} 
\mathcal{O}(1) & \text{if } y \text{ is not } \tau\text{-stable} \\
 j \cdot c_y + \mathcal{O}(1) & \text{if } y \text{ is non-stable} \\
 \max\{ j \cdot c_y + \sigma^q(y) + \mathcal{O}(1), \ j \cdot c_y + \mathcal{O}(1) \} & \text{if } x = y
\end{cases} 
\]

(9)

Now we can prove (5). Recall that the expression \( \sigma_j \circ \lambda(x) \) is the expression \( \sigma^q \circ \lambda(x) \) where all registers \( y \in \text{Var}(\sigma^q \circ \lambda(x)) \) are substituted with \( \sigma^q \circ \tau^{j+1}(y) \). First notice that by Lemma 3 the expression \( \sigma^q \circ \lambda(x) \) is equivalent to \( \max\{ c'' + \sigma^q(x), d'' \} = \max\{ \sigma^q(x) + \mathcal{O}(1), \mathcal{O}(1) \} \). We can do these estimations because \( \lambda \) does not depend on \( j \). By combining this equation with the definition of \( \sigma_j \) we have:

\[
\sigma_j \circ \lambda(x) = \sigma^q \circ \tau^{j+1} \circ \sigma^q \circ \lambda(x) = \sigma^q \circ \tau^{j+1} \circ \max\{ \sigma^q(x) + \mathcal{O}(1), \ \mathcal{O}(1) \}
\]

Thus to finish the proof it suffices to show that:

\[
\sigma_j(x) = \max\{ j \cdot c + \sigma^q(x) + \mathcal{O}(1), \ j \cdot d + \mathcal{O}(1) \}
\]

(10)

for some constants \( c, d \in \mathbb{N} \). Indeed, if (10) holds, then \( \sigma_j(x) + \mathcal{O}(1) = \max\{ j \cdot c + \mathcal{O}(1) \} \). Similarly the value \( \mathcal{O}(1) \) can be estimated by the expression \( j \cdot d + \mathcal{O}(1) \) so the last max operation is not needed. Notice that this does not change the constants \( c \) and \( d \), which proves that they do not depend on \( \lambda \).

To conclude the proof, we show that (10) holds for a stable register \( x \). Recall that \( \sigma_j = \sigma^q \circ \tau^{j+1} \circ \sigma^q \). We will show that (10) holds even if we change \( \sigma_j \) by
\( \sigma^g \circ \tau^{j+1} \circ \sigma' \) where \( \sigma' \) is any copyless substitution and \( x \) is \( \sigma' \)-stable. The proof is by induction on the size of \( \sigma'(x) \). For the base step we must consider \( \sigma'(x) = x \) (i.e. \( x \) is stable). Then:

\[
\sigma_j(x) = \sigma^g \circ \tau^{j+1} \circ \sigma'(x) \\
= \sigma^g \circ \tau^{j+1}(x) \\
= \max\{ j \cdot c_x + \sigma^g(x) + O(1), j \cdot c_x + O(1) \} \quad \text{(by (8)).}
\]

Suppose that (10) holds for expressions \( \sigma' \) of length \( n \) and we want to show that (10) for an expression \( \sigma''(x) \) of the form \( \sigma''(x) = \sigma'(x) \oplus f \) where \( \oplus \) is either + or max, \( x \) is \( \sigma' \)-stable and \( f \) is an expression where all their registers are non-stable (recall that \( \sigma'' \) is copyless). By unraveling \( \sigma^g \circ \tau^{j+1} \circ \sigma''(x) \), we get:

\[
\sigma^g \circ \tau^{j+1} \circ \sigma''(x) = \sigma^g \circ \tau^{j+1} \circ (\sigma'(x) \oplus f) \\
= (\sigma^g \circ \tau^{j+1} \circ \sigma'(x)) \oplus (\sigma^g \circ \tau^{j+1} \circ f) \\
= \max\{ j \cdot c + \sigma^g(x) + O(1), j \cdot d + O(1) \} \oplus (\sigma^g \circ \tau^{j+1} \circ f) \quad \text{(by induction)}
\]

On the last expression, it is easy to check that \( \sigma^g \circ \tau^{j+1} \circ f \) is equal to a constant or to \( j \cdot c_f + O(1) \) for some constant \( c_f \neq -\infty \). Indeed, by (9) we know that \( \sigma^g \circ \tau^{j+1}(y) = j \cdot c_y + O(1) \) for non-stable registers. Since \( f \) is an expression over non-stable register, one can show by induction over the size of \( f \) that \( \sigma^g \circ \tau^{j+1} \circ f \) is either a constant or \( j \cdot c_f + O(1) \) for some constant \( c_f \neq -\infty \) and \( j \) big enough.

We start with the case when \( \sigma^g \circ \tau^{j+1} \circ f \in \mathbb{N} \). Then \( \sigma^g \circ \tau^{j+1} \circ f \in O(1) \) and for \( \oplus \) equal to + or max we have \( \sigma^g \circ \tau^{j+1} \circ \sigma''(x) = \max\{ j \cdot c + \sigma^g(x) + O(1), j \cdot d + O(1) \} \) for \( j \) big enough. Suppose now that \( \sigma^g \circ \tau^{j+1} \circ f = j \cdot c_f + O(1) \). Then we must consider when \( \oplus \) is + or max operation. For the former we have:

\[
\sigma^g \circ \tau^{j+1} \circ \sigma''(x) = \max\{ j \cdot c + \sigma^g(x) + O(1), d \cdot j + O(1) \} + j \cdot c_f + O(1) \\
= \max\{ j \cdot (c + c_f) + \sigma^g(x) + O(1), j \cdot (d + c_f) + O(1) \},
\]

and for the latter we have:

\[
\sigma^g \circ \tau^{j+1} \circ \sigma''(x) = \max\{ \max\{ j \cdot c + \sigma^g(x) + O(1), j \cdot d + O(1) \}, j \cdot c_f + O(1) \} \\
= \max\{ j \cdot c + \sigma^g(x) + O(1), j \cdot d + O(1), j \cdot c_f + O(1) \} \\
= \max\{ j \cdot c + \sigma^g(x) + O(1), j \cdot \max\{d, c_f \} + O(1) \}.
\]

The number of induction steps depends on the size of the expression \( \sigma''(x) \) which for \( \sigma''(x) = \sigma^g(x) \) does not depend on \( j \). This concludes the proof. \( \square \)

## 5 Non-expressibility of copyless CRA

In this section, we show two facts about the expressiveness of copyless CRA: (1) we show that the class of linear ambiguous weighted automata is not contained
in the class of copyless CRA and (2) copyless CRA are not closed under reverse. We derive this two conclusions by showing a function \( f \) that is definable by a copyless CRA but its reverse (i.e. \( f^r(w) = f(w^r) \)) is not definable by any copyless CRA over the same semiring. Interestingly, \( f^r \) is definable by a linear ambiguous weighted automata and, therefore, from this we also conclude (1).

Consider the function \( f_B \) given by the following copyless CRA \( B \) over \( \Sigma = \{a, b, \#\} \) and \( \mathbb{N}_{-\infty}(\max,+) \):

\[
\begin{align*}
    &a \quad x := x + 1 \\
    &\quad y := y \\
    &x, y := 0 \\
    &\quad \# \quad x := \max\{x, y\} \\
    &\quad y := 0 \\
    &b \quad x := x \quad \max\{x, y\} \\
    &\quad y := y + 1
\end{align*}
\]

To understand the output of \( B \), let us first define \( f_B \) formally. For any word \( w \in \Sigma^* \) let \( k \) be the number of \( \# \) symbols in \( w \). Furthermore, let \( n_i \) and \( m_i \) be the number of \( a \)'s and \( b \)'s, respectively, between the \( i \)-th and \( i+1 \) occurrence of \( \# \) in \( w \) for \( 0 < i < k \). Additionally, let \( n_0, m_0, n_k, m_k \) be the numbers of \( a \)'s and \( b \)'s before and after the first and last \( \# \) in \( w \). Then by the definition of \( B \), one can easily check that \( f_B \) is given by the following definition:

\[
f_B(w) = \max_j \left\{ m_j + \sum_{i=j+1}^{k} n_i \mid j \in \{0, \ldots, k\} \right\}
\]

for \( w \neq \epsilon \) and \( f_B(\epsilon) = 0 \). From this definition, one can also give a formal definition of \( f_B^r \) (i.e. the reverse function of \( f_B \)) which is given by reversing the roll of \( n_i \) and \( m_j \) in the definition of \( f_B \). Formally, one can easily check that the definition of \( f_B^r \) is defined as:

\[
f_B^r(w) = \max_j \left\{ \sum_{i=0}^{j-1} n_i + m_j \mid j \in \{0, \ldots, k\} \right\}
\]

In Theorem 1 below we show that \( f_B^r \) is not definable by any copyless CRA. Intuitively, a copyless CRA \( A \) that computes \( f_B^r \) will have to store the number of \( a \)'s in a register \( x_a \) and compare \( x_a \) with the last sequence of \( b \)'s between \( \# \)-letters. Since a copyless CRA can used the content of a register just once per transition, then \( A \) will lose the content of \( x_a \) and forget the number of \( a \)'s seen so far. The previous argument is formalized in the next result.

**Theorem 1.** The function \( f_B^r \) is not recognizable by any copyless CRA.
\textbf{Proof.} Suppose there exists a copyless CRA \( \mathcal{A} = (Q, \Sigma, \mathcal{A}, \delta, q_0, \nu_0, \mu) \) which computes the function \( f_B^R \). Since the function \( f_B^R \) is non-zero (i.e. \( f_B^R(w) = -\infty \) for every \( w \in \Sigma^* \)) then by Proposition 1 we can assume that the transitions, initial and final functions in \( \mathcal{A} \) do not use \(-\infty\). Without lost of generality, we can also assume that \( \mathcal{A} \) is a strongly connected automaton (see Section 4.3). If not, we can always change our analysis by adding a word \( w_0 \) from the initial state to a BSCC and then construct a counterexample word from there. Formally, let \( \delta^*(q_0, w_0) = (q_0', \sigma_0) \) where \( q_0' \) is a state inside a BSCC \( Q' \subseteq Q \) of \( \mathcal{A} \). We can always redefine \( \mathcal{A} \) as follows: \( q_0' \) is the new initial state, and the new initialization function \( \nu_0' \) is defined as \( \nu_0'(x) = \nu_0 \circ \sigma_0(x) \). It is straightforward to check that for every word \( w \) the new automaton constructed from \( \mathcal{A} \) will return the output value of \( \mathcal{A} \) over \( w_0.w \). Thus, for the rest of the proof we assume that \( \mathcal{A} \) is strongly connected and, therefore, all the results from Section 4 hold.

We start the proof by analyzing the behavior of \( \mathcal{A} \) on words containing loops of just one letter. Since \( Q \) is strongly connected, then there exists a state \( q_a \), a word \( \nu_a = a^{n_a} \) with \( n_a > 0 \) and a substitution \( \tau_a \) such that \( \delta^*(q_a, \nu_a) = (q_a, \tau_a) \). We can assume by Lemma 4 that \( \tau_a \) is a collapse substitution, that is, \( \operatorname{Var}(\tau_a(x)) = \emptyset \) for all non-stable registers \( x \) in \( \tau_a \). In addition, define a sequence of words:

\[
w_a(j) = w^{q_a} \cdot x^{j+1} \cdot w^{q_a}
\]

such that \( \delta^*(q_a, w_a(j)) = (q_a, \sigma_a^j) \) for some substitution \( \sigma_a^j \) (i.e. \( \sigma_a^j \) depends on \( j \)). Recall that \( w^{q_a} \) is the word defined in Proposition 3 for the state \( q_a \). By Lemma 5 we know that there exist constants \( c_a^x \) and \( d_a^x \) such that for \( j \) big enough:

\[
\sigma_a^j(x) = \begin{cases} 
\mathcal{O}(1) & \text{if } x \text{ is non-stable} \\
\max\{ j \cdot c_a^x + \sigma_a^q(x) + \mathcal{O}(1), j \cdot d_a^x + \mathcal{O}(1) \} & \text{otherwise.} 
\end{cases}
\] (11)

Analogously, by the definition of \( \mathcal{A} \) we can find a state \( q_b \) in \( \mathcal{A} \), a word \( \nu_b = b^{n_b} \) for some \( n_b \geq 0 \) and a collapsing substitution \( \tau_b \) such that \( \delta^*(q_b, \nu_b) = (q_b, \tau_b) \). Then similar to the sequence \( w_a(j) \), we can define the sequence of words \( w_b(j) = w^{q_b} \cdot x^{j+1} \cdot w^{q_b} \) such that \( \delta^*(q_b, w_b(j)) = (q_b, \sigma_b^j) \) for a substitution \( \sigma_b^j \) that satisfies (11) but for new constants \( c_b^x \) and \( d_b^x \).

The next step is to understand the growth of stable registers when we repeat \( w_a(j) \) \( s \)-times or, more important, when we compose \( (\sigma_a^j)^s \). For any \( j \geq 1 \) and \( s \geq 1 \), define the sequence of words:

\[
w_a(s, j) = (w^{q_a} \cdot x^{j+1})^s \cdot w^{q_a}.
\]
and \( \sigma_a^{s,j} = (\sigma_a \circ \tau^{j+1})^s \circ \sigma_a \). Let \( x \) be a stable register on \( \mathcal{A} \). For \( s = 2 \) one can easily check the following calculations:

\[
\begin{align*}
\sigma_a^{2,j}(x) &= \sigma_a^{q_a} \circ \tau^{j+1} \circ \sigma_a^{q_a} \circ \tau^{j+1} \circ \sigma_a^{q_a}(x) \\
&= \sigma_a^{q_a} \circ \tau^{j+1} \circ \sigma_a^{j}(x) \\
&= \sigma_a^{q_a} \circ \tau^{j+1} \circ \left( \max \left\{ j \cdot c_a^x + \sigma_a^{q_a}(x) + \mathcal{O}(1), \ j \cdot d_a^x + \mathcal{O}(1) \right\} \right) \\
&= \max \left\{ j \cdot c_a^x + \sigma_a^{q_a}(x) + \mathcal{O}(1), \ j \cdot d_a^x + \mathcal{O}(1) \right\} \\
&= \max \left\{ j \cdot c_a^x + \sigma_a^{q_a}(x) + \mathcal{O}(1), \ j \cdot d_a^x + \mathcal{O}(1) \right\} + \mathcal{O}(1), \ j \cdot d_a^x + \mathcal{O}(1) \right\} \\
&= \max \left\{ 2j \cdot c_a^x + \sigma_a^{q_a}(x) + 2 \cdot \mathcal{O}(1), \ j \cdot c_a^x + j \cdot d_a^x + 2 \cdot \mathcal{O}(1), \ j \cdot d_a^x + \mathcal{O}(1) \right\} \\
&= \max \left\{ 2j \cdot c_a^x + \sigma_a^{q_a}(x) + 2 \cdot \mathcal{O}(1), \ j \cdot c_a^x + j \cdot d_a^x + 2 \cdot \mathcal{O}(1) \right\} \\
&= \max \left\{ 2 \cdot j \cdot c_a^x + \sigma_a^{q_a}(x) + 2 \cdot \mathcal{O}(1), \ j \cdot c_a^x + j \cdot d_a^x + 2 \cdot \mathcal{O}(1) \right\}
\end{align*}
\]

(by definition)

(by (11))

(by dominance)

In general, for any natural number \( s > 1 \) if \( x \) is a stable register it is straightforward to check that:

\[
\sigma_a^{s,j}(x) = \max \left\{ s \cdot j \cdot c_a^x + \sigma_a^{q_a}(x) + \mathcal{O}(s), \ (s-1) \cdot j \cdot c_a^x + j \cdot d_a^x + \mathcal{O}(s) \right\}
\]

(12)

where \( \mathcal{O}(s) \) is a function that grows linearly with respect to \( s \) and do not depend on \( j \).

We are ready to define the word for which we prove that \( \mathcal{A} \) give the wrong output. Let \( w_0 \) be a word such that \( \delta^*(q_0, w_0) = (q_a, \sigma_0) \). Let also \( w^{q_a,q_b} \) and \( w^{q_b,q_a} \) be the “collapsing words” between \( q_a \) and \( q_b \) from Proposition 3. For any \( j \geq 0 \) define the sequence of words:

\[
w(s,j) = w_0 \cdot w_a(s,j) \cdot w^{q_a,q_b} \cdot w_b(j^2) \cdot w^{q_b,q_a} \cdot w_a(j)
\]

It is straightforward to check that for any \( s, j \geq 0 \) the word \( w(s,j) \) has a run that goes to \( q_a \) with \( a \), then it cycles in \( q_a \) with \( w_a(s,j) \), next it goes from \( q_a \) to \( q_b \) with \( w^{q_a,q_b} \), cycles again \( j^2 \) times in \( q_b \) with \( w_b(j^2) \), then again back from \( q_b \) to \( q_a \) with \( w^{q_b,q_a} \) and finally it cycles \( j \)-times with the word \( w_a(j) \). In other words, \( w(s,j) \) behaves as expected by cycling and jumping between \( q_a \) and \( q_b \). In the sequel, we show that \( \mathcal{A} \) outputs wrong value over \( w(s,j) \) when we fix \( x \) and we take \( j \) big enough.

First we estimate the correct value \( f_B^R(w(s,j)) \) for any \( s, j \geq 0 \). By Proposition 3 we can assume that \( w_a(s,j), w_a(j) \) and \( w_b(j) \) contain the letter \( \# \). Thus the loops of \( a \)'s and \( b \)'s in the words \( w_a(s,j), w_a(j), \) and \( w_b(j) \) are all separated by \( \# \). The only fragments of \( w(s,j) \) that depend on \( j \) are these loops. From the definition of these loops, one can easily check that the number of \( a \)'s in \( w_a(j) \) is \( j \cdot n_a + \mathcal{O}(1) \) and the number of \( b \)'s in \( w_b(j^2) \) is \( j^2 \cdot n_b + \mathcal{O}(1) \). Furthermore, similar to \( w_a(j) \), the number of \( a \)'s in \( w_a(s,j) \) is equal to \( s \cdot j \cdot n_a + \mathcal{O}(1) \). Recall that \( f_B^R(w(s,j)) = \max \left\{ \sum_{i=0}^{t} n_i + m_i \right\} \), where \( n_i \) and \( m_i \) are the numbers of \( a \)'s and \( b \)'s, respectively, between the \( \# \)-letters and \( t \) is the number of maximal subsequences without \( \# \). It is easy to see that:

\[
f_B^R(w(s,j)) = n_b \cdot j^2 + n_a \cdot s \cdot j + \mathcal{O}(s)
\]

(13)
where $O(s)$ represents a fix number of $a$’s (i.e. do not depend on $j$) that are presented in $w_0$, $w^{q_a}$, or $w^{q_b}$. Notice that the last suffix $w_a(j)$ of $a$’s is not contributing in the sum. This happen because the sequence $w_b(j^2)$ is overshadowing the last sequence $w_a(j)$, i.e., the max-operator consider the number of $b$’s in $w_b(j^2)$ instead of the number of $a$’s in $w_a(j)$. The rest of the proof is to show that $A$ does not output the right value on $w(s,j)$. The reason of this misbehavior of $A$ with respect to $w(s,j)$ is because if $A$ is summing the sequence of $a$’s before $w_b(j^2)$ then it must also sum the sequence of $a$’s after $w_b(j^2)$ which, by the previous calculations, we know that this should not happen.

We estimate now the values in the registers of $A$ after reading the word $w(s,j)$ for any $j$ big enough. By the construction of $w(s,j)$ we know that $\delta^*(q_0,w(s,j)) = (q_0,\sigma_{w(s,j)})$ and, thus, the current value in the registers of $A$ after reading $w(s,j)$ is given by the assignment:

$$
\nu_0 \circ \sigma_{w(s,j)} = \nu_0 \circ \sigma_0 \circ \sigma_a^{s \cdot j} \circ \sigma_y^{q_b \cdot q_a \cdot q_b} \circ \sigma_b^{j \cdot q_b \cdot q_a} \circ \sigma_a^{j \cdot q_a}
$$

For all non-stable registers $y$ this expression is equivalent to $\sigma_a^j(y)$ (by Lemma 5) which is equal to a constant $O(1)$ that does not depend on $j$. For a stable register $x$, the story is much more complicated. We evaluate the expression $\sigma_{w(s,j)}(x)$ step-by-step to estimate this value. First, by (11) we get that:

$$
\sigma_a^j(x) = \max\{ j \cdot c_a^x + \sigma_y^{q_a}(x) + O(1), j \cdot d_a^x + O(1) \}
$$

Then by composing $\sigma_b^{j \cdot q_b \cdot q_a}$ with $\sigma_a^j(x)$ we get:

$$
\sigma_b^{j \cdot q_b \cdot q_a} \circ \sigma_a^j(x) = \sigma_b^j \circ \sigma_y^{q_b \cdot q_a} \circ \left( \max\{ j \cdot c_a^x + \sigma_y^{q_a}(x) + O(1), j \cdot d_a^x + O(1) \} \right)
$$

$$
= \max\{ j \cdot c_a^x + \sigma_b^j \circ \sigma_y^{q_b \cdot q_a} \circ \sigma_y^{q_a}(x) + O(1), \, j \cdot d_a^x + O(1) \}
$$

$$
= \max\{ j \cdot c_a^x + \sigma_b^j \circ \sigma_y^{q_b \cdot q_a} \circ \sigma_y^{q_a}(x) + O(1), \, j \cdot d_a^x + O(1) \}
$$

$$
= \max\{ j \cdot c_a^x + j^2 \cdot c_b^x + \sigma_y^{q_b}(x) + O(1), \, j \cdot d_a^x + O(1) \}
$$

The next-to-last substitution holds by Lemma 5 and by considering $\lambda = \sigma_y^{q_b \cdot q_a} \circ \sigma_y^{q_a}$. The next step is to compose $\sigma_a^{s \cdot j} \circ \sigma_y^{q_b \cdot q_a}$ with $\sigma_b^j \circ \sigma_y^{q_b \cdot q_a} \circ \sigma_a^j$. We denote
this composition by \( \sigma^{-w_0}_{w(s,j)} \):

\[
\sigma^{-w_0}_{w(s,j)} = \sigma^{s,j} \circ \sigma^{q_0, q_b} \circ \left( \sigma^2_b \circ \sigma^{q_0, q_a} \circ \sigma^j_a \right)
\]

\[
= \sigma^{s,j} \circ \sigma^{q_0, q_b} \circ \max \left\{ j \cdot c_a^x + j^2 \cdot c_b^x + \sigma^{q_b}(x) + \mathcal{O}(1),
\quad j \cdot c_a^x + j^2 \cdot d_b^x + \mathcal{O}(1),
\quad j \cdot d_a^x + \mathcal{O}(1) \right\}
\]

\[
= \max \left\{ j \cdot c_a^x + j^2 \cdot c_b^x + \sigma^{s,j} \circ \sigma^{q_0, q_b} \circ \sigma^{q_b}(x) + \mathcal{O}(1),
\quad j \cdot c_a^x + j^2 \cdot d_b^x + \mathcal{O}(1),
\quad j \cdot d_a^x + \mathcal{O}(1) \right\}
\]

\[
= \max \left\{ j \cdot c_a^x + j^2 \cdot c_b^x + \sigma^{q_b}(x) + \mathcal{O}(1),
\quad j \cdot c_a^x + j^2 \cdot d_b^x + \mathcal{O}(1),
\quad j \cdot d_a^x + \mathcal{O}(1) \right\}
\]

\[
= \max \left\{ j \cdot c_a^x + j^2 \cdot c_b^x + \sigma^{q_b}(x) + \mathcal{O}(1),
\quad j \cdot c_a^x + j^2 \cdot d_b^x + \mathcal{O}(1),
\quad j \cdot d_a^x + \mathcal{O}(1) \right\}
\]

\[
\nu_0 \circ \sigma_{w(s,j)}(x) = \max \left\{ (s+1) \cdot j \cdot c_a^x + j^2 \cdot c_b^x + \sigma^{q_b}(x) + \mathcal{O}(1),
\quad j \cdot c_a^x + j^2 \cdot d_b^x + \mathcal{O}(1),
\quad j \cdot d_a^x + \mathcal{O}(1) \right\}
\]

For the rest of the proof, we fix a value for \( s \) that does not depend on \( j \) (i.e. the exact value for \( s \) will be defined towards the end of the proof). We study now the behavior of the function \( \mathbb{A}(w(s,j)) \) for \( j \) big enough and we start with the form of \( \nu_0 \circ \sigma_{w(s,j)}(x) \) for a stable register \( x \). For a fix \( s \) we denote the quadratic functions on \( j \) by:

\[
g_1^x(j) = c_b^x \cdot j^2 + c_a^x \cdot (s+1) \cdot j + \mathcal{O}(1)
\]

\[
g_2^x(j) = c_b^x \cdot j^2 + (c_a^x \cdot s + d_b^x) \cdot j + \mathcal{O}(1)
\]

\[
g_3^x(j) = d_b^x \cdot j^2 + c_a^x \cdot j + \mathcal{O}(1)
\]

\[
g_4^x(j) = d_b^x \cdot j + \mathcal{O}(1)
\]

where \( \mathcal{O}(1) \) denotes some constant that do not depend on \( j \) (it probably depends on \( s \) but recall that \( s \) is now fixed). Then for every stable register \( x \) we have \( \nu_0 \circ \sigma_{w(s,j)}(x) = \max_{i=1}^4 \{ g_i^x(j) \} \). Notice that if \( d_b^x < c_a^x \) then for \( j \) big enough it
Lemma for the new definition of $g$

By the previous discussion, it is easy to see that $\nu$ is constant for non-stable registers and we can regroup the sets $\{s, j\}$ big enough there exists a constant $\nu_0 \circ \sigma_{w(s,j)}(x) = \max_{i=1}^4 \{ g_i^\sigma(j) \}$ for the new definition of $g_i^\sigma(j)$.

We study now the output function $[A](w(s,j)) = \nu_0 \circ \sigma_{w(s,j)} \circ \mu(q_a)$. By Lemma 1, we know that the copyless expression $\mu(q_a)$ can be presented as an expression of the form:

$$
\mu(q_a) = \max_{i=1}^k \left\{ l_i + \sum_{x \in X_i} x \right\}
$$

where $X_1, \ldots, X_k$ is a sequence of different sets over $\mathcal{X}$ and $l_1, \ldots, l_k$ is a sequence of values over $\mathbb{N}$ for $k \geq 0$. Recall that by the previous analysis we know that $\nu_0 \circ \sigma_{w(s,j)}(x)$ is equal to a constant whenever $x$ is non-stable or equal to $\max_{i=1}^4 \{ g_i^\sigma(j) \}$ whenever $x$ is a stable register for $j$ big enough. Then by composing $\nu_0 \circ \sigma_{w(s,j)}$ with $\mu(q_a)$ we get:

$$
[A](w(s,j)) = \nu_0 \circ \sigma_{w(s,j)} \circ \mu(q_a)
= \nu_0 \circ \sigma_{w(s,j)} \circ \max_{i=1}^k \left\{ l_i + \sum_{x \in X_i} x \right\}
= \max_{i=1}^k \left\{ l_i + \sum_{x \in X_i} \nu_0 \circ \sigma_{w(s,j)}(x) \right\}
= \max_{i=1}^h \left\{ m_i + \sum_{x \in Y_i} \max_{i=1}^4 \{ g_i^\sigma(j) \} \right\}
$$

where $Y_1, \ldots, Y_h$ is a new sequence of subsets of stable registers and $m_1, \ldots, m_h$ is a sequence of values over $\mathbb{N}$ for $k \geq 0$. The last equality holds since $\nu_0 \circ \sigma_{w(s,j)}(x)$ is constant for non-stable registers and we can regroup the sets $X_i$ and sum the constants $l_i$ to form new sets of stable registers $Y_i$ and new constants $m_i$.

The next step is to further simplify the output $[A](w(s,j))$. For this, note that $[A](w(s,j))$ is the sum and maximization of quadratic functions over $j$. Then for $j$ big enough there exists a constant $i^* \leq h$ and a partition of $Y_i, = Z_1 \cup Z_2 \cup Z_3 \cup Z_4$ such that:

$$
[A](w(s,j)) = m_i^* + \sum_{x \in Z_1} g_1^\sigma(j) + \sum_{x \in Z_2} g_2^\sigma(j) + \sum_{x \in Z_3} g_3^\sigma(j) + \sum_{x \in Z_4} g_4^\sigma(j)
$$

In the previous claim, $i^*$ is the maximum argument where (15) is maximize and the partition $Y_{i^*} = Z_1 \cup Z_2 \cup Z_3 \cup Z_4$ is the division of $Y_{i^*}$ where each function $g_i^\sigma$ dominates for $j$ big enough. Thus, $[A](w(s,j))$ is a sum of quadratic functions and by summing common $j$-terms we can reduce $[A](w(s,j))$ to a polynomial of the form $A \cdot j^2 + B \cdot j + C$. Intuitively the value $A \cdot j^2$ should correspond to the number of $b$'s in $w(s,j)$ and $B \cdot j$ to the number of $a$'s in $w(s,j)$.
of this proof, we analyze the $B$-coefficient and compare it with the corresponding coefficient in $f_B^R(w(s,j))$.

Recall from (13) that the output of $f_B^R$ over $w(s,j)$ is equal to $n_b \cdot j^2 + n_a \cdot s \cdot j + O(s)$. This suggests that if the output $[A](w(s,j))$ is correct then we should have that $B = n_a \cdot s$. By adding the linear coefficients of $g_i^R(j)$ for $i \in \{1, 2, 3, 4\}$ and $x \in Z_i$ we get that:

$$B = \sum_{x \in Z_i} c_x^a \cdot (s + 1) + \sum_{x \in Z_2} e_x^a \cdot (s + 1) + \sum_{x \in Z_3} c_x^e + \sum_{x \in Z_4} d_x^e$$

$$= \sum_{x \in Z_1 \cup Z_2} c_x^a \cdot (s + 1) + \sum_{x \in Z_2} e_x^a + \sum_{x \in Z_3} c_x^e + \sum_{x \in Z_4} d_x^e.$$  

Given that $B$ should be equal to $n_a \cdot s$ then this implies that $\sum_{x \in Z_1 \cup Z_2} c_x^a$ must be strictly less than $n_a$ (i.e. $\sum_{x \in Z_1 \cup Z_2} c_x^a < n_a$). Otherwise, $B$ will be too big with respect to $n_a \cdot s$. This implies that $\sum_{x \in Z_1 \cup Z_2} c_x^a \leq n_a - 1$ and then:

$$(s + 1) \cdot \sum_{x \in Z_1 \cup Z_2} c_x^a \leq (s + 1) \cdot (n_a - 1)$$

By replacing this fact in the definition of $B$, we can over-approximate the $B$-coefficient as follows:

$$B \leq (s + 1) \cdot (n_a - 1) + \sum_{x \in Z_2} e_x^a + \sum_{x \in Z_3} c_x^e + \sum_{x \in Z_4} d_x^e$$

$$\leq s \cdot n_a - s + n_a - 1 + \sum_{x \in X} (e_x^a + c_x^e + d_x^e).$$ (16)

For the last bound, recall that all constants $e_x^a$, $c_x^e$ and $d_x^e$ are non-negative and, thus, $\sum_{x \in X} (e_x^a + c_x^e + d_x^e)$ is greater or equal than the sum of a subset of $X$. It is important to notice that inequality (16) is independent on how we choose $s$, namely, for any value of $s$ we will have that $B$ must satisfy (16).

We are ready to show that, by suitably choosing a fix value for $s$, $B$ will not be equal to $n_a \cdot s$. Formally, choose $s > n_a - 1 + \sum_{x \in X} (e_x^a + c_x^e + d_x^e)$. This with (16) implies that $B < n_a \cdot s$ which is a contradiction. In other words, $A$ is not counting all the $a$’s in $w(s,j)$ for $j$ big enough. This proves that $A$ does not output the right function.

In the previous proof we have shown that $f_B^R$ is not definable by any copyless CRA. Interestingly, the reverse of this function $f_B$ is definable by a copyless CRA as it was shown at the beginning of this section. From this, we can conclude that the class of copyless CRA is not closed under reverse.

It is also interesting that $f_B^R$ is definable by a linear ambiguous weighted automata. To see this, consider the following weighted automata $A'$:
In the previous diagram, the initial and final states are marked with an incoming and outgoing arrow, respectively, and they start and finish with value 0. States that are not marked start or finish with $-\infty$. One can easily check that the weighted automaton $A'$ computes the function $f_R^B$ and, moreover, $A'$ is linear ambiguous. This shows that there exist functions that are definable by linear ambiguous weighted automata, but they cannot be defined by a copyless CRA.

We summarize the conclusions of Theorem 1 in the next corollary.

**Corollary 1.**
1. The class of linear ambiguous weighted automata is not contained in the class of copyless CRA.
2. The class of functions recognizable by copyless register automata is not closed under reverse.

6 Bounded alternation copyless CRA

We define the *alternation* of an expression $e \in \text{Expr}_S(X)$ as maximum number of shifts between $\oplus$ and $\odot$ in all branches of the parse tree of expression $e$. Formally, for any expression $e$ we define $f_\oplus(e) = 1$ if $e = e_1 \odot e_2$, $f_\odot(e) = 1$ if $e = e_1 \oplus e_2$ and $f(e) = 0$ otherwise. The alternation $\text{Alt}(e)$ of an expression $e$ is recursively defined as $\text{Alt}(e) = 0$ for every $e \in S \cup X$ and $\text{Alt}(e_1 \odot e_2) = \max_{i \in \{1,2\}} \{\text{Alt}(e_i) + f_\oplus(e_i)\} + 1$ for $\oplus \in \{\oplus, \odot\}$ and any expressions $e_1, e_2 \in \text{Expr}_S(X)$. We add 1 to distinguish expressions not using any binary operators from the expressions using exactly one type of binary operator. We denote by $\text{Expr}_N(X)$ all expressions with alternation bounded by $N$.

Theorem 1 and Corollary 1 show that Copyless CRA are not closed under reverse, that is, the execution of Copyless CRA is asymmetric with respect of an input. A close look to the copyless CRA $B$ define in Section 5 shows that the unbounded alternation of the final expression is the reason why the function is not closed under reverse. This fact motivates the definition of *bounded alternating* copyless CRA (BAC-CRA), a subclass of copyless CRA. It turns out that the class BAC-CRA have good properties for a computational model. In particular, BAC-CRA is closed under regular-lookahead (see the definition below) which is the main result of this section (see Theorem 2).

Formally, we say that a Copyless CRA $A$ has bounded alternation if there exists $N \in \mathbb{N}$ such that, for every $w = a_1 \ldots a_n \in \Sigma^*$, if $(q_0, \nu_0) \xrightarrow{a_1} \ldots \xrightarrow{a_n} (q_n, \nu_n)$ is the run of $A$ over $w$ and $\delta(q_{i-1}, a_i) = (q_i, \sigma_i)$ for every $1 \leq i \leq n$, then it holds
that $\nu_0 \circ \sigma_1 \circ \ldots \circ \sigma_n \circ \mu(q_n) \in \text{Expr}_N(\mathcal{X})$, that is, the number of alternations of all ground expressions that are outputs of $\mathcal{A}$ is uniformly bounded by a constant. A CRA $\mathcal{A}$ is called a bounded alternating copyless CRA (BAC-CRA) if $\mathcal{A}$ is copyless and has bounded alternation. Most of the Copyless CRA examples presented in this paper have bounded alternation. For instance, Examples 1 and 2 are part of the class of BAC-CRA.

Interestingly, BAC-CRA are closed under regular lookahead A CRA with regular look-ahead (CRA-RLA) [2] is an extension of CRA where the machine can take decisions by reading the remaining suffix of the string in a regular manner. Formally, let $\text{REG}_\Sigma$ be the set of all regular languages over $\Sigma$. A CRA-RLA is a tuple $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \delta, q_0, \nu, \mu)$ where $Q$, $\Sigma$, $\mathcal{X}$, $q_0$, $\nu$, and $\mu$ are defined as before and $\delta : Q \times \text{REG}_\Sigma \rightarrow Q \times \text{Subs}(\mathcal{X})$ is a partial transition function with finite domain restricted as follows: for a fixed state $q$ let $\delta(q, L_1) = (q_1, \sigma_1), \delta(q, L_2) = (q_2, \sigma_2), \ldots, \delta(q, L_k) = (q_k, \sigma_k)$ be all transitions with $q$ in the first coordinate and $L_1, \ldots, L_k \in \text{REG}_\Sigma$. Then the languages $L_1, \ldots, L_k$ are pairwise disjoint (i.e. $L_1 \cap L_2 = \emptyset$). Note that the last requirement forces that on a given word the automaton $\mathcal{A}$ is deterministic. Given a string $w = a_1 \ldots a_n \in \Sigma^*$, the run of $\mathcal{A}$ over $w$ is a sequence of configurations:

$$(q_0, \nu_0) \xrightarrow{L_1} (q_1, \nu_1) \xrightarrow{L_2} \ldots \xrightarrow{L_n} (L, \nu)$$

such that for $1 \leq i \leq n$, $\delta(q_{i-1}, L_i) = (q_i, \sigma_i), w[i, \cdot) \in L_i$ and $\nu_i(x) = [\nu_{i-1} \circ \sigma_i(x)]$ for each $x \in \mathcal{X}$. The output of $\mathcal{A}$ over $w$ is defined as usual, i.e. $[[\mathcal{A}](w) = [\nu_n(\mu(q_n))]$. The extension of the classes of copyless and bounded alternation CRA-RLA are defined similarly.

The main result of this section is to show that BAC-CRA with regular look-ahead defined the same class of functions as normal BAC-CRA.

**Theorem 2.** For every BAC-CRA $\mathcal{A}$ with regular look-ahead there exists a BAC-CRA $\mathcal{A}'$ without regular look-ahead that computes the same function, that is, $[[\mathcal{A}](w) = [[\mathcal{A}'](w)$ for every $w \in \Sigma^*$.

The proof of Theorem 2 goes in two steps: (1) we show that for every BAC-CRA $\mathcal{A}$ with regular look-ahead there exists an unambiguous BAC-CRA $\mathcal{A}'$ equivalent to $\mathcal{A}$ (Lemma 7) and (2) for every unambiguous BAC-CRA $\mathcal{A}'$ there exists a BAC-CRA $\mathcal{A}''$ equivalent to $\mathcal{A}'$ (Lemma 8). Clearly, if we show (1) and (2), then Theorem 2 will be proved.

An unambiguous CRA is a non-deterministic CRA such that for every word $w$ there exists exactly one run that is accepting. Formally, a non-deterministic CRA is a tuple $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \Delta, q_0, \nu, \mu)$ where $Q$, $\Sigma$, $\mathcal{X}$, $q_0$, $\nu$, and $\mu$ are defined as before, $\Delta \subseteq Q \times \Sigma \times Q \times \text{Subs}(\mathcal{X})$ is a finite transition relation and $F$ is the set of final states. Additionally, we assume that for every $q, q' \in Q$ and $a \in \Sigma$ there exists at most one $\sigma \in \text{Subs}(\mathcal{X})$ such that $(q, a, q', \sigma) \in \Delta$. Given a string $w = a_1 \ldots a_n \in \Sigma^*$, a run of $\mathcal{A}$ over $w$ is a sequence of configurations:

$$(q_0, \nu_0) \xrightarrow{a_1} (q_1, \nu_1) \xrightarrow{a_2} \ldots \xrightarrow{a_n} (q_n, \nu_n)$$
such that for $1 \leq i \leq n$, $(q_{i-1}, a_i, q_i, \sigma_i) \in \Delta$ and $\nu_i(x) = [\nu_{i-1} \circ \sigma_i(x)]$ for each $x \in \mathcal{X}$. Furthermore, a run of $A$ over $w$ (like the one above) is called accepting if $q_n \in F$. Then, we say that a non-deterministic $A$ is unambiguous if for every $w \in \Sigma^*$ there exists exactly one accepting run of $A$ over $w$. The output of an unambiguous CRA $A$ over $w$ is defined as $[A](w) = [\nu_n(\mu(q_n))]$ where $(q_n, \nu_n)$ is the final configuration of the only accepting run of $A$ over $w$. The definition of unambiguous copyless CRA or unambiguous BAC-CRA are a straightforward extension of the definition above.

The following lemma is a well-known property of unambiguous finite automata and, in particular, of unambiguous CRA that will be useful during the proof Lemma 7.

**Lemma 6.** [13, 16] If $\rho$ and $\rho'$ are two runs of an unambiguous finite automaton $A$ over $w \in \Sigma^*$, then the last states of $\rho$ and $\rho'$ are different, that is, $\rho(|w|) \neq \rho'(|w|)$.

The previous lemma implies that the number of active runs of an unambiguous automaton until any position of the input is always bounded by the number of states. This observation will be useful in the proof of the second step. First we show next that BAC-CRA extended with regular look-ahead can be expressed with unambiguous BAC-CRA.

**Lemma 7.** For every BAC-CRA $A$ with regular look-ahead there exists an unambiguous BAC-CRA $A'$ that computes the same function, that is, $[A](w) = [A'](w)$ for every $w \in \Sigma^*$.

**Proof.** Let $A = (Q, \Sigma, \mathcal{X}, \Delta, \mathcal{R}, q_0, \nu_0, \mu)$ be a copyless CRA-RLA. To represent the transition function $\Delta$ finitely, let $L_1, \ldots, L_N$ be all the regular languages in the finite domain of $\Delta$ and, for each $i \leq N$, let $A_i = (P_i, \Sigma, \delta_i, p_i^0, F_i)$ be a finite state automaton such that $L_i = \mathcal{L}(A_i)$. To unify the automata structure of all $A_i$, define the set of states $P = \mathcal{U}_{i=1}^N P_i$, the transition function $\delta = \mathcal{U}_{i=1}^N \delta_i$, and the set of final state $F = \mathcal{U}_{i=1}^N F_i$, that is, $P, \delta$ and $F$ are the disjoint union of states, transitions and final states, respectively, of the automata $A_1, \ldots, A_N$. Then for each $i \leq k$, define the automaton $R_i = (P, \Sigma, q_i^0, F)$. By the definition of $P$, $\delta$ and $F$, one can easily see that $L_i = \mathcal{L}(R_i)$ for every $i \leq k$.

We are ready to define the unambiguous BAC-CRA equivalent to $A$. Let $A' = (Q', \Sigma, \mathcal{X}, \Delta', q'_0, \nu_0, F', \mu')$ be a BAC-CRA such that:

- $Q' = Q \times 2^P$ is the set of states,
- $\Delta' \subseteq Q' \times \Sigma \times Q' \times \text{Subs}(\mathcal{X})$ where $((q, S), a, (q', S'), \sigma) \in \Delta'$ iff there exist $(q, L_i, q', \sigma) \in \Delta$ for some $i \leq N$, and a surjective function $f: S \cup \{p_i^0\} \rightarrow S'$ such that $\delta(s, a) = f(s)$ for every $s \in S \cup \{p_i^0\}$,
- $q'_0 = (q_0, \varnothing)$,
- $F' = \{(q, S) \mid S \subseteq F\}$, and
- $\mu': Q' \rightarrow \text{Expr}(\mathcal{X})$ where $\mu'(q, S) = \mu(q)$ for every $(q, S) \in Q'$. 


The main idea of $\mathcal{A}'$ is to guess and choose, at each letter, the right transitions in $\mathcal{A}$ that is satisfied by the current suffix. In each state $(q, S)$ of $\mathcal{A}'$ we keep the current state $q$ of a run and a subset $S$ of $P$ that includes all regular transitions that has been taken so far by the run on $q$. Then we have a transition $((q, S), a, (q', S'), \sigma) \in \Delta'$ if there exist a transition $(q, L_i, q', \sigma) \in \Delta$ (i.e. a transition from $q$ to $q'$) such that we can extend each state in $S \cup \{p_i^0\}$ to a state in $S'$. Note that, in order to start simulating the finite automaton $R_i$ over the suffix, $p_i^0$ is also included on the set of states that are updated. Finally, if the last state $(q, S)$ of a run satisfies $S \subseteq F$, then we know that all suffixes during the run satisfy the regular look-ahead of the transitions and, then, the state $(q, S)$ is final.

We show first that $\mathcal{A}'$ is unambiguous. By contradiction, suppose that $\mathcal{A}'$ is not unambiguous, that is, there exist $w = a_1 \ldots a_n \in \Sigma^*$ and two different accepting runs $\rho$ and $\rho'$ of $\mathcal{A}'$ over $w$. Let $i \leq n$ be the least position such that $\rho(i) = \rho'(i) = (q, S)$ but $\rho(i + 1) = (q_1, S_1) \neq (q_2, S_2) = \rho'(i + 1)$. We know that this position exists since, by construction, it holds that $\rho(0) = \rho'(0)$. Let $(q, L, q_1, \sigma_1) \in \Delta$ and $(q', L', q_2, \sigma_2) \in \Delta$ be the transitions that witness the transitions $((q, S), a_{i+1}, (q_1, S_1), \sigma_1) \in \Delta'$ and $((q, S), a_{i+1}, (q_2, S_2), \sigma_2) \in \Delta'$. Since both runs are accepting, then it is straightforward to show by induction that $w[i, \cdot] \in L$ and $w[i, \cdot] \in L'$ for the suffix $w[i, \cdot]$ of $w$ at position $i$. Then we have a contradiction since, by definition of CRA-RLA, we know that $L \cap L' = \emptyset$. We conclude that $\mathcal{A}'$ must be unambiguous. Note that during the construction we did not change the assignments $\sigma$ in the transitions. For this reason, we can also conclude that $\mathcal{A}'$ is a copyless CRA with bounded alternation.

For the last part of the proof, we have to show that $[\mathcal{A}](w) = [\mathcal{A}'](w)$ for every $w \in \Sigma^*$. It is easy to verify that for every run $(q_0, \nu_0) \xrightarrow{\Delta \nu_0} \cdots \xrightarrow{\Delta \nu_n} (q_n, \nu_n)$ of $\mathcal{A}$ over $w \in \Sigma^*$, there exist sets $S_i$ such that the sequence $((q_0, S_0), \nu_0) \xrightarrow{\Delta \nu} \cdots \xrightarrow{\Delta \nu} ((q_n, S_n), \nu_n)$ is an accepting run of $\mathcal{A}'$ over $w$. For a given word $w$ the sets $S_i$ are uniquely determined by the transitions $\Delta$ and $\nu$. Thus, $[\mathcal{A}](w) = \hat{\nu}_n(\mu(q_n))] = [\mathcal{A}'](w)$ for every $w \in \Sigma^*$.

Next we prove the second step of the proof.

**Lemma 8.** For every unambiguous BAC-CRA $\mathcal{A}$ there exists an BAC-CRA $\mathcal{A}'$ that computes the same function, that is, $[\mathcal{A}](w) = [\mathcal{A}'](w)$ for every $w \in \Sigma^*$.

**Proof.** Before going into the main details, we need to introduce standard notation for trees and parse trees that will be useful during this proof.

**Trees.** Let $\Sigma$ be a set of labels. An (unordered) labeled $\Sigma$-tree $t$ is a finite function $t : \text{nodes}(t) \rightarrow \Sigma$ such that nodes$(t)$ is a finite prefix-closed subset of $\mathbb{N}^*$, that is, $w \in \text{nodes}(t)$ whenever $w \cdot i \in \text{nodes}(t)$ for some $i \in \mathbb{N}$. Additionally we assume that $w \cdot i$ implies $w \cdot (i - 1) \in \text{nodes}(t)$ for $i > 0$. We say that the $i$-word is the root of $t$ and that $w \cdot i \in \text{nodes}(t)$ is a child of $w$. Further, we write labels$(t)$ to denote the set labels of a tree $t$. For any node $w \in \text{nodes}(t)$, we denote by $t[w]$ the subtree rooted at $w$, i.e. the labeled tree $t[w] : \text{nodes}(t[w]) \rightarrow \Sigma$ such
that $t[w](i) = t(w \cdot i)$ for every $i \in \text{nodes}(t[w])$. We usually write $a(t_1, \ldots, t_k)$ to denote a tree whose root is labeled by $a$ and $t_1, \ldots, t_k$ are the subtrees hanging from the root (i.e., for every $i \leq k$ there exists $j \in \mathbb{N}$ such that $t[j] = t_i$). We say that $w \in \text{nodes}(t)$ is an internal node of $t$ if $w \cdot i \in \text{nodes}(t)$ for some $i \in \mathbb{N}$. Otherwise, $w$ is called a leaf of $t$. The set of all leaves of $t$ is denoted by $\text{leaves}(t)$. We say that a tree is complete if any internal nodes has at least two children. One can easily check that if $t$ is a complete tree, then $|\text{nodes}(t)| \leq 2 \cdot |\text{leaves}(t)|$. Finally, we denote by $\text{Trees}(\Sigma)$ the set of all $\Sigma$-trees.

**Parse trees.** Let $\text{labs} = S \cup \{\oplus, \otimes\} \cup \mathcal{X}$ be the a set of labels in $S$ and variables $\mathcal{X}$. A parse tree is a complete labeled $\text{labs}$-tree $p$ whose leaves are labeled by $S \cup \mathcal{X}$ and internal nodes by $\{\oplus, \otimes\}$. We denote by $\text{Parse}(\mathcal{X})$ the set of all $\text{labs}$-parse trees. For any expression $e \in \text{Expr}(\mathcal{X})$, the parse tree of $e$ is a tree $p_e$ recursively defined as $p_e = e$ whenever $e$ is equal to a constant or variable, and $p_e = \oplus\{p_{e_1}, p_{e_2}\}$ whenever $e = e_1 \otimes e_2$ where $\otimes \in \{\oplus, \otimes\}$.

Conversely, any parse tree $p$ can be converted into an equivalent expression $\text{exp}(p)$. If $p$ is a single node then $e = p$ otherwise if $p = \oplus\{p_1, \ldots, p_k\}$ for $\otimes \in \{\oplus, \otimes\}$ then $\text{exp}(p) = \oplus_{1 \leq i \leq k} \text{exp}(p_i)$. So far we used $\oplus$ and $\otimes$ as binary operations thus the equivalent expressions are not defined uniquely. We allow for this because this definition is unique up to evaluation equivalence since $\oplus$ and $\otimes$ are commutative and associative. We define the size of an expression $e$ as the number of nodes in the parse tree $p_e$.

**Example 3.** Consider the expression $e = ((x_0 \otimes (y \otimes 2)) \oplus (z \otimes 4))$ where $x, y, z \in \mathcal{X}$ and $2, 3, 4 \in S$. One can easily check that the parse tree $p_e$ of $e$ is the following:

![Parse Tree Diagram]

Internal nodes of a parse tree can be merged by applying associativity and commutativity of $\oplus$ and $\otimes$, respectively. Formally, for any label $\otimes \in \{\oplus, \otimes\}$ define the flattening function $\text{flat}_\otimes$ such that $\text{flat}_\otimes(\oplus\{p_1, \ldots, p_k\})$ is equal to $\{p_1, \ldots, p_k\}$ whenever $\otimes = \oplus$ and $\oplus\{p_1, \ldots, p_k\}$ otherwise, for any label $\otimes \in \{\oplus, \otimes\}$ and trees $p_1, \ldots, p_k$. Then, given a parse tree $p$ we denote by $p^*$ the reduced parse tree constructed recursively as follows: $p^* = \oplus\{\text{flat}_\otimes(p_1^*), \ldots, \text{flat}_\otimes(p_k^*)\}$ whenever $p = \oplus\{p_1, \ldots, p_k\}$ and $p^* = p$ whenever $p^*$ is equal to a variable or constant. By the construction of $p^*$, the label of any node of $p^*$ is different to all root labels of its children, i.e., $p^*(e) \neq p^*(i)$ for any $i \in \text{nodes}(p^*) \cap \mathbb{N}$. Or equivalently if $\otimes \in \{\oplus, \otimes\}$ is a label of a internal node then it does not have a child labeled with $\otimes$. 
Proposition 5. Let \( p \) be a parse tree. The expressions \( \exp(p) \) and \( \exp(p^*) \) are equivalent and the alternation of these expressions is the same.

Proof. We prove this by induction on the depth of \( p \). If the depth is 0 then \( p = p^* \) by definition. Let \( p = \otimes\{p_1, \ldots, p_k\} \) and \( p^* = \otimes\{\flat_\otimes(p_1^*), \ldots, \flat_\otimes(p_k^*)\} \). Let \( p' = \otimes\{p_1', \ldots, p_k'\} \). By the induction assumption we know that \( \exp(p) \) and \( \exp(p') \) are equivalent and the alternation of these expressions is the same. To prove the proposition we need to show that this also holds between \( \exp(p') \) and \( \exp(p^*) \). Let \( p_i^* = \otimes\{p_i, \ldots, p_i, k_i\} \). It suffices to prove that for every \( i \) the parse trees \( p_i' \) and \( p_i'^* = \otimes\{p_i^*, \ldots, \flat_\otimes(p_i^*), \ldots, p_i^*\} \) define equivalent expressions with the same alternation. If \( \otimes \neq \otimes \) then these parse trees are the same. Otherwise \( p_i'^* = \otimes\{p_i^*, \ldots, p_i, 1, \ldots, p_i, k_i, \ldots, p_i^*\} \). Then by definition

\[
\exp(p') = \bigotimes_j \exp(p_j^*) \approx \bigotimes_{j \neq i} \exp(p_j^*) \otimes \exp(p_i^*) = \bigotimes_j \exp(p_j^*) \otimes \bigotimes_{j \neq i} \exp(p_i, j) \approx p_i'^*
\]

where \( \approx \) is the relation of equivalent expressions. These equalities follow from the commutativity and associativity of \( \otimes \).

We show that \( p_i' \) and \( p_i'^* \) have the same alternation. Formally the definition of alternation was for binary operators \( \oplus \) and \( \otimes \). It is easy to generalize it for the \( \otimes \) notation. We define \( f_\otimes(e) = 1 \) if \( e = \bigotimes_j e_j \), \( f_\otimes(e) = 1 \) if \( e = \bigoplus_j e_j \) and \( f(e) = 0 \) otherwise. The alternation \( \Alt \) is defined as \( \Alt(c) = 0 \) for every \( c \in S \) and \( \Alt(\bigotimes_j e_j) = \max_j \{\Alt(e_j) + f_\otimes(e_j)\} + 1 \) for \( \otimes \in \{\oplus, \otimes\} \) and any expressions \( e_j \) over \( \mathbb{S} \).

\[
\Alt(p_i') = \max_j \{\Alt(\exp(p_j^*)) + f_\otimes(\exp(p_i^*))\} + 1 = \\
\max_j \{\max_{j \neq i} \{\Alt(\exp(p_j^*)) + f_\otimes(\exp(p_i^*))\}, \Alt(\exp(p_i^*)) + f_\otimes(\exp(p_i^*))\} + 1
\]

Since \( f_\otimes(\exp(p_i^*)) = 0 \) then this is equivalent to

\[
\max_j \{\max_{j \neq i} \{\Alt(\exp(p_j^*)) + f_\otimes(\exp(p_j^*))\}, \Alt(\exp(p_i^*))\} + 1 = \\
\max_j \{\max_{j \neq i} \{\Alt(\exp(p_j^*)) + f_\otimes(\exp(e_j))\}, \max_j \{\Alt(p_i, j) + f_\otimes(p_i, j)\}\} + 1 = \Alt(p_i'^*)
\]

By Proposition 5 the alternation of an expression \( e \) is equivalent to the depth of its reduced parse tree \( p_e^* \). From now, we will implicitly assume that parse trees are always in their reduced form (i.e. \( p = p^* \) for every \( p \in \text{Parse}(X \cup Y) \)).

Example 4. Recall the parse tree \( p_e \) of the expression \( e \). The following tree is the reduced parse tree \( p_e^* \):

```
``
One of the main ingredients of the proof are trees labeled by substitutions. More precisely, for two disjoint set of variables $\mathcal{X}_1$ and $\mathcal{X}_2$, define $\text{Subs}(\mathcal{X}_1, \mathcal{X}_2)$ to be the set of all copyless substitutions $\sigma : \mathcal{X}_1 \rightarrow \text{Expr}(\mathcal{X}_1 \cup \mathcal{X}_2)$, that is, copyless substitutions where the domain contains only variables in $\mathcal{X}_1$. Here, composition between substitutions in $\text{Subs}(\mathcal{X}_1, \mathcal{X}_2)$ is defined in a straightforward way where $\mathcal{X}_2$-variables are treated as constants. Notice that the composition of two substitutions is copyless for the registers $\mathcal{X}_1$ but not necessarily for the registers $\mathcal{X}_2$. This is because the registers $\mathcal{X}_2$ are not in the domain of these substitutions.

The set $\text{Subs}(\mathcal{X}_1, \mathcal{X}_2)$ is used to label trees. Formally, we denote by $\text{Trees}(\mathcal{X}_1, \mathcal{X}_2)$ the set of all trees labeled by $\text{Subs}(\mathcal{X}_1, \mathcal{X}_2)$. We usually called a tree $t \in \text{Trees}(\mathcal{X}_1, \mathcal{X}_2)$ a substitution tree. Furthermore, we say that $t$ is copyless if $t(u)$ is a copyless substitution for every $u \in \text{nodes}(t)$ and:

$$\text{Var}(t(u)) \cap \text{Var}(t(v)) \subseteq \mathcal{X}_1$$

for every different $u, v \in \text{nodes}(t)$ where $\text{Var}(t(u))$ and $\text{Var}(t(v))$ are the set of all variables in $t(u)$ and $t(v)$, respectively. In other words, in addition that each substitution is copyless, each $\mathcal{X}_2$-variable is used at most once in a substitution tree $t$ (note that there is no restriction in $\mathcal{X}_1$). Notice that with this restriction the composition of substitutions that are labels of different nodes is copyless also for the registers $\mathcal{X}_2$.

We say that $t$ is constants-free if, for every $u \in \text{nodes}(t)$ the substitution $t(u)$ does not use constants from the semiring. Finally, for any node $u \in \text{nodes}(t)$ we define the collapse operation $t^i(u)$ to be the expression:

$$t^i(u) = t(\epsilon) \circ t(u[\cdot, 1]) \circ \ldots \circ t(u[\cdot, k])$$

where $k = |u|$ and $u[\cdot, i]$ is the prefix of $u$ until position $i$. In other words, $t^i(u)$ is the composition of all substitution along the branch from the root until $u$. Notice that this composition is a copyless substitution because of the condition (17).

Let $A = (Q, \Sigma, \mathcal{X}, \delta, q_0, \nu_0, F, \mu)$ be an unambiguous BAC-CRA whose alternation is bounded by $N$. To construct a deterministic BAC-CRA $A'$ from $A$, the idea is to keep in memory a tree of runs that encode how runs are branching when the word is read. Of course, if we keep the branching of all runs in memory, the tree would be unbounded. A characteristic property of unambiguous CRA is that, for each prefix, there is at most $|Q|$ active runs and, furthermore, the last state of all of them are different (Lemma 6). These two facts suggest a tree structure of the runs where each branch is a run and where the number of branches is bounded by $|Q|$. Although we can keep in finite memory the branching structure of the actives runs of $A$, we cannot do the same trick over the registers $A$ and naively keep copies of registers for each run (recall that $A'$ must be copyless). To overcome this problem, $A'$ will postpone the substitutions of each run by labeling internal tree structure of runs with substitutions. Clearly, $A'$ cannot postpone these substitutions forever and store a long sequence of these objects with finite memory. The key idea here is to prune and reduce the tree structure.
by doing partial evaluation of the substitutions whenever is possible. We will show that by exploiting the bounded alternation of the output and the copyless restriction of $\mathcal{A}$, we need only finite amount of memory to remember the tree structure and the substitutions of all runs.

Recall that $\mathcal{X}$ is the set of registers in $\mathcal{A}$. Let $\hat{\mathcal{X}}$ be a disjoint copy of $\mathcal{X}$. Formally, we define $\hat{\mathcal{X}} = \{ \hat{x} \mid x \in \mathcal{X} \}$ to have a natural correspondence between the registers in $\mathcal{X}$ and $\hat{\mathcal{X}}$. We construct a deterministic BAC-CRA $\mathcal{A}' = (Q', \Sigma, \mathcal{Y}, \delta', q'_0, \nu'_0, \mu')$ as follows.

- $Q'$ is the set of all pairs $(t, B)$ where $t \in \text{Trees}(\mathcal{X}, \mathcal{Y})$ is a complete, copyless, and constants-free substitution tree and $B : \text{leaves}(t) \rightarrow Q$ is an injective function. Additionally we assume that $|\text{range}(B) \cap F| = 1$. In the end we restrict $Q'$ to its subset of reachable states $Q'_r$.
- $\mathcal{Y}$ is a set of registers of size $|\mathcal{Y}| = 2 \cdot |Q| \cdot |\mathcal{X}| \cdot (|\mathcal{X}| \cdot N + 1)$ satisfying $\hat{\mathcal{X}} \subseteq \mathcal{Y}$ and $\mathcal{X} \cap \mathcal{Y} = \emptyset$ (recall that $\hat{\mathcal{X}}$ is a copy of $\mathcal{X}$).
- $q'_0 = (t_0, B_0)$ is the initial state of $\mathcal{A}'$ where $t_0$ is a single-node tree labeled with $\sigma_0 \in \text{Subs}(\hat{\mathcal{X}}, \mathcal{Y})$ and $B_0(\epsilon) = q_0$ such that $\sigma_0(x) = \hat{x}$ for every $x \in \mathcal{X}$, where $\hat{x}$ is the copy of $x$ in $\hat{\mathcal{X}}$.
- $\nu'_0 : \mathcal{Y} \rightarrow S$ is the initial substitution defined as $\nu'_0(\hat{x}) = \nu_0(x)$ for every $x \in \mathcal{X}$ and $\nu'_0(y) = \emptyset$ for every $y \in \mathcal{Y} \setminus \mathcal{X}$.
- $\mu'$ is the final substitution defined for every $(t, B) \in Q'$ by:

$$
\mu'(t, B) = t^i(u) \circ \mu(q)
$$

where $u \in \text{leaves}(t)$ is the only leaf satisfying $B(u) \in F$ and $q = B(u)$.

- $\delta'$ is the transition function that will be conveniently defined in the sequel.

We start explaining the relation between the set of registers $\hat{\mathcal{X}}$, $\mathcal{X}$ and $\mathcal{Y}$. The $\mathcal{X}$-variables in this setting represent $\mathcal{X}$-variables from $\mathcal{A}$ inside the internal structure of $\mathcal{A}'$ (i.e. states). In fact, $\mathcal{X}$-variables will never be used as real variables by $\mathcal{A}'$. They will just be used to keep track of temporary substitutions. Regarding $\mathcal{Y}$, the decision of taking the size of $\mathcal{Y}$ equal to $2 \cdot |Q| \cdot |\mathcal{X}| \cdot (|\mathcal{X}| \cdot N + 1)$ is technical and will be clear later in the proof. The property $\hat{\mathcal{X}} \subseteq \mathcal{Y}$ is needed for the definition of the initial substitution. Besides that we will not use the fact that $\hat{\mathcal{X}} \subseteq \mathcal{Y}$.

Each state in $Q'$ is composed by a complete, copyless, and constants free substitution tree $t$ and an injective function $B : \text{leaves}(t) \rightarrow Q$. As was suggested before, $t$ keeps track of the branching history of the active runs of $\mathcal{A}$ and $B$ labels leaves of $t$ with states in $Q$. The plan here is that each active run of $\mathcal{A}$ is represented by a different branch of $t$ and the state assigned by $B$ to the leaf of the branch represents the current state of the run.

Recall that $Q'_r \subseteq Q'$ is the set of reachable states in $\mathcal{A}'$. For a given word $w$ consider the set of active runs in $\mathcal{A}$. By Lemma 6 every active run on $w$ is
determined by its last state. We define \( Q(w) \subseteq Q \) as the set of last states in the active runs in \( A \). To understand the main purpose of \((t, B)\), we state the following Lemma. Its proof is postponed to the end of this section.

**Lemma 9.** Consider one of the runs of \( A \) on \( w \) and let \((q, \nu)\) be its configuration. Then there is a run of \( A' \) on \( w \) reaching configuration \(((t, B), \xi)\), where \( \xi \) is a valuation over \( \mathcal{Y} \). Moreover \( Q(w) = \text{range}(B) \) and for all \( q \in Q(w) \) there exists \( u \in \text{leaves}(t) \) such that \( B(u) = q \) and

\[
\nu = \xi \circ t^I(u).
\]

In other words, we can recover the configuration \((q, \nu)\) from a leaf \( u \) by applying \( B \) over \( u \) and by composing the substitutions of the branch from the root of \( t \) into \( u \).

We show that \( Q' \) is a finite set. We bound the size of any state \((t, B) \in Q' \) in terms of the values \(|Q|, |\mathcal{X}|, \) and \( N \). First, the number of nodes in \( t \) is bounded by \( 2 \cdot |Q| \), that is, \( \text{nodes}(t) \leq 2 \cdot |\text{leaves}(t)| \leq 2 \cdot |Q| \) since \( t \) is a complete tree and \( B \) is an injective function over \( \text{leaves}(t) \). Second, each copyless and constants-free expression that one can write with \(|\mathcal{X} \cup \mathcal{Y}| \) variables is of size at most \( 2 \cdot (|\mathcal{X}| + |\mathcal{Y}|) \).

Recall that the size of an expression is defined as the number of nodes in its parse tree and parse trees are complete trees. This means that the size of any copyless and constants-free substitutions \( \sigma \in \text{Subs}(\mathcal{X}, \mathcal{Y}) \) satisfies \( |\sigma| \leq 2 \cdot |\mathcal{X}| \cdot (|\mathcal{X}| + |\mathcal{Y}|) \) and thus the number of possible labels for \( t \) is finite. Finally, \( B \) is an injective function from \( \text{leaves}(t) \) into \( Q \). Given that \( t \) is of bounded size, then \( B \) is also bounded.

Before we define the transition function \( \delta' \) we need to give a battery of definitions. A partial substitution \( \sigma : \mathcal{Y} \rightarrow \text{Expr}(\mathcal{Y}) \) is a substitution whose domain is a subset of \( \mathcal{Y} \). For an expression \( e \) we use the notation \( \text{Var}_\mathcal{Y}(e) = \text{Var}(e) \cap \mathcal{Y} \), i.e., the set of variables from \( \mathcal{Y} \). The key idea in the construction of \( \delta' \) is the notion of an \( \mathcal{X} \)-reduction. An \( \mathcal{X} \)-reduction of \( p \in \text{Parse}(\mathcal{X} \cup \mathcal{Y}) \) is a tuple \((r, \sigma)\) where \( r \in \text{Parse}(\mathcal{X} \cup \mathcal{Y}) \) is a parse tree without constants and \( \sigma : \mathcal{Y} \rightarrow \text{Expr}(\mathcal{Y}) \) is a partial substitution such that \( \sigma(\text{exp}(r)) \) is equivalent to \( \text{exp}(p) \). Moreover the domain of \( \sigma \) is equal to \( \text{Var}_\mathcal{Y}(\text{exp}(r)) \) and \( \text{exp}(r) \) is a copyless expression. The last requirement ensures that each variable from \( \mathcal{Y} \) occurs in \( r \) at most once.

The goal of an \( \mathcal{X} \)-reduction is to factorize constants into \( \mathcal{Y} \)-variables in such a way that we do not loose the previous tree. Of course, an \( \mathcal{X} \)-reduction could increment the number of \( \mathcal{Y} \)-variables introduced to remove constants. The trick here is to construct, for each parse tree \( p \), an \( \mathcal{X} \)-reduction \((r, \sigma)\) such that the number of nodes of \( r \) depends only on \(|N|\) and \(|\mathcal{X}|\). We define this \( \mathcal{X} \)-reduction by induction over the depth of the parse tree by using a function \( \text{red}_\mathcal{X}(\cdot) \) that receives a parse tree \( p \) and produces a tuple \((r, \sigma)\). For this definition we assume that \( \mathcal{Y} \) is possibly infinite. Later, in Lemma 10, we show that we need only a finite set of labels, which proves that this definition is correct.

For the base case, if \( p = x \in \mathcal{X} \) then we define \( \text{red}_\mathcal{X}(p) = (x, \sigma_\emptyset) \) where \( \sigma_\emptyset \) is the empty function. Otherwise if \( p \) is a constant or a variable from \( \mathcal{Y} \) then we
choose a variable \( y \in \mathcal{Y} \) and define \( \text{red}_\mathcal{X}(p) = (y, \sigma) \), where \( \sigma \) is defined only on \( y \) by \( \sigma(y) = p \).

For the induction step we need to be able to relabel \( \mathcal{X} \)-reductions. Let \( \sigma \) be a partial substitution whose domain is \( A \subseteq \mathcal{Y} \) and let \( (r, \sigma) \) be an \( \mathcal{X} \)-reduction of \( p \). We say that \((r', \sigma')\) is a relabeling of \((r, \sigma)\) if the domain of \( \sigma' \) is \( A' \) and there is a bijection \( f : A' \to A \) such that \( \sigma'(x) = \sigma(f(x)) \) and \( f(r') = r \), that is substituting variables from \( A' \) into variables from \( A \) changes \( r' \) to \( r \). By definition a relabeling is changing the domain of \( \sigma \) but the codomain of \( \sigma \) is the same. Moreover \((r', \sigma')\) is also an \( \mathcal{X} \)-reduction of \( p \).

Suppose that \( p \) is of the form \( p = \Theta(p_1, \ldots, p_j, p_1', \ldots, p_k') \) where each \( p_i \in \text{Trees}(\mathcal{X} \cup \mathcal{Y}) \) contains at least one variable in \( \mathcal{X} \) and each \( p_i' \in \text{Trees}(\mathcal{Y}) \) contains no variables in \( \mathcal{X} \). Furthermore, suppose that \( \text{red}_\mathcal{X}(p_1) = (r_1, \sigma_1), \ldots, \text{red}_\mathcal{X}(p_j) = (r_j, \sigma_j) \) are already defined. We relabel the \( \mathcal{X} \)-reductions \((p_i, \sigma_i)\) to \((p_i', \sigma_i')\) in such a way that the domains of \( \sigma_i' \) are pairwise disjoint. Notice that we need to have enough labels in \( \mathcal{Y} \) to do such a relabeling. In Lemma 10 we show that there are enough labels in \( \mathcal{Y} \). We define \( \text{red}_\mathcal{X}(p) = (r, \sigma) \) recursively as follows:

\[
    r = \Theta(r_1', \ldots, r_j', y)  \\
    \sigma = (\sigma_1' \cup \ldots \cup \sigma_j')\{y \rightarrow \exp(\Theta(p_1', \ldots, p_k'))\}
\]

where \( y \in \mathcal{Y} \) is a fresh variable not used in \( \sigma_1, \ldots, \sigma_j \) and \( (\sigma_1' \cup \ldots \cup \sigma_j') \) corresponds to the disjoint union of the substitutions. Recall that \( \sigma_i \) have disjoint domains. If \( k = 0 \) (there are no subtrees without label from \( \mathcal{X} \)) then we do not add the additional variable \( y \).

In this construction the subtrees \( p_1', \ldots, p_k' \) that do not use \( \mathcal{X} \) registers are replaced by a new fresh variable \( y \) and the content \( \Theta(p_1', \ldots, p_k') \) is assigned into \( y \). It is clear from the definition that \((r, \sigma)\) is an \( \mathcal{X} \)-reduction for \( p \). Since \( \sigma_i' \) have disjoint domains and \( y \) was chosen as a fresh variable then \( \exp(r) \) is copyless and the domain of \( \sigma \) is equal to \( \sum \text{Var}_\mathcal{Y}(\exp(r_j')) \cup \{y\} = \text{Var}_\mathcal{Y}(r) \).

**Example 5.** Recall the reduced parse tree \( p_e^* \) of the expression \( e \) in Example 4. Suppose that \( x \in \mathcal{X} \) and \( y, z \in \mathcal{Y} \). Then the \( \mathcal{X} \)-reduction \( \text{red}_\mathcal{X}(p_e^*) = (r, \sigma) \) of \( p_e^* \) is equal to:

\[
\begin{align*}
    r : & \cdot \\
    \text{Var} & : u := y \odot 2 \\
    u & := 3 \oplus (z \odot 4)
\end{align*}
\]

where \( u \) and \( v \) are fresh variables in \( \mathcal{Y} \). One can easily check that \( \exp(p_e^*) = \hat{\sigma}(\exp(r)) \) and, thus, the expression defined by \( p_e^* \) is preserved in \((r, \sigma)\).

We say that a partial substitution is copyless if it is copyless with respect to the variables in its domain. We prove that the size of \( r \) depends on the size of \( \mathcal{X} \) and \( N \).
Lemma 10. Suppose that \( p \) is a parse tree of a copyless expression in \( \text{Expr}(\mathcal{X} \cup \mathcal{Y}) \) whose alternation is bounded by \( N \). If \( \text{red}_\mathcal{X}(p) = (r, \sigma) \) then: \( r \) is a parse tree of a copyless expression with alternation bounded by \( N \); \( \sigma \) is a partial copyless substitution; and the number of \( \mathcal{Y} \)-variables in \( r \) is bounded by \( |\mathcal{X}| \cdot N + 1 \).

Proof. Suppose that \( \text{red}_\mathcal{X}(p) = (r, \sigma) \). By the definition of \( \text{red}_\mathcal{X}(\cdot) \) it is easy to check that \( r \) is copyless and its alternation is bounded by \( N \). In fact, each time that a subtree is replaced in the original tree \( p \), we use a new fresh variable, that is, each variable in \( r \) is used at most one time. Notice that in our recursive definition of \( \text{red}_\mathcal{X} \) there was at most one recursive step for every internal node. In the definition of partial substitutions we assign subtrees of internal nodes to a fresh variable from \( \mathcal{Y} \). The recursion is not called in the subtrees assigned to the new fresh variable thus for different variables \( x \) the expressions \( \sigma(x) \) correspond to different fragments of the tree \( p \). Since \( p \) is copyless this proves that \( \sigma \) is also copyless.

The most interesting part is the bound of \( \mathcal{Y} \)-variables in \( r \). We define \( \mathcal{X}_r = \text{Var}(\exp(r)) \cap \mathcal{X} \), i.e., the labels from \( \mathcal{X} \) used in \( r \). Similarly let \( \mathcal{Y}_r = \text{Var}(\exp(r)) \cap \mathcal{Y} \), i.e., the \( \mathcal{Y} \)-variables used in \( r \). Since \( \exp(r) \) is copyless then every label from \( \mathcal{X}_r \) and \( \mathcal{Y}_r \) occurs only once in \( r \). We prove a slightly stronger variant of the inequality in the lemma. We show that \( |\mathcal{Y}_r| \leq |\mathcal{X}_r| \cdot N + 1 \). Since \( |\mathcal{X}_r| \leq |\mathcal{X}| \) this proves the lemma.

The proof is by induction over the alternation of \( p \). For the base case we have \( |\text{nodes}(r)| = 1 \) thus the bound is trivially true. For the inductive step suppose that the result holds for trees of alternation at most \( N \). Take a parse tree \( p \) with alternation \( N + 1 \) of the form:

\[
p = \bigoplus \{ p_1, \ldots, p_j, p'_1, \ldots, p'_k \}
\]

where each \( p_i \in \text{Parse}(\mathcal{X} \cup \mathcal{Y}) \) contains at least one variable in \( \mathcal{X} \) and each \( p'_i \in \text{Parse}(\mathcal{Y}) \) contains no variables in \( \mathcal{X} \). Since we assumed that parse trees are in their reduced form the alternation of each \( p_i \) is at most \( N \). Using the induction assumption on the parse trees \( p_i \) we get \( |\mathcal{Y}_{p_i}| \leq |\mathcal{X}_{p_i}| \cdot N + 1 \) for every \( i \).

Since the labels in \( \mathcal{X}_{p_i} \) occur also in \( \mathcal{X}_r \) we get the following

\[
\sum_{i=1}^{j} |\mathcal{Y}_{p_i}| \leq \sum_{i=1}^{j} (|\mathcal{X}_{p_i}| \cdot N + 1) \leq |\mathcal{X}_r| \cdot N + j.
\]

The set \( \mathcal{Y}_r \) is divided into the set of all nodes in \( \mathcal{Y}_{p_i} \) and possibly the additional node \( y \) added for the remaining expressions \( p'_i \). Notice that the internal nodes are labeled with \( \bigoplus \) and \( \bigodot \) thus the root of \( r \) does not contribute to the sum. We get the following:

\[
|\mathcal{Y}_r| \leq \sum_{i=1}^{j} |\mathcal{Y}_{p_i}| + 1 \leq |\mathcal{X}_r| \cdot N + j + 1
\]

Notice that since in every parse tree \( p_i \) there is at least one node with a label from \( \mathcal{X} \). This gives us \( j \leq |\mathcal{X}_r| \). Using this bound we get

\[
|\mathcal{X}_r| \cdot N + j + 1 \leq |\mathcal{X}_r| \cdot N + |\mathcal{X}_r| + 1 = |\mathcal{X}_r| \cdot (N + 1) + 1
\]
which proves the induction step. □

We can naturally extend the definition of reduction \( \text{red}_X \) to be defined on expressions \( e \in \text{Expr}(X \cup Y) \). The result is defined as \( (e', \sigma) \), where \( e' = \exp(\text{red}_X(p_e)) \). Furthermore, we extend the function \( \text{red}_X(\cdot) \) from expressions \( \text{Expr}(X \cup Y) \) to substitutions in \( \text{Subs}(X, Y) \). More precisely, for any \( \alpha \in \text{Subs}(X, Y) \) define \( \text{red}_X(\alpha) = (\alpha', \sigma_\alpha) \) such that \( \text{red}_X(\alpha(x)) = (\alpha'(x), \sigma_x) \) for every \( x \in X \) where \( \sigma_x \) is the disjoint union of all assignments of the form \( \sigma_x \). If the domains of \( \sigma_x \) are not disjoint then we relabel the reductions \( (\alpha'(x), \sigma_x) \) in such a way that the domains are disjoint. For simplicity we assume that \( (\alpha'(x), \sigma_x) \) are already relabeled. The domain of \( \sigma_\alpha \) is the sum of all domains of \( \sigma_x \). Finally, we also extend \( \text{red}_X(\cdot) \) from substitutions to substitution trees such that \( \text{red}_X(t) = (t', \sigma_t) \) where \( \text{nodes}(t) = \text{nodes}(t') \), \( \text{red}_X(t(u)) = (t'(u), \sigma_u) \) for each \( u \in \text{nodes}(t) \), and \( \sigma_t \) is the disjoint union of all substitutions \( \sigma_u \). Similarly if the domains of the substitution were not disjoint then we can relabel the reductions. The domain of \( \sigma_t \) is the sum of all domains of \( \sigma_u \). We extend the partial substitution \( \sigma_t \) to a substitution by

\[
\sigma_t'(x) = \begin{cases} 
\sigma_t(x) & \text{if } x \text{ is in domain of } \sigma \\
0 & \text{otherwise.}
\end{cases}
\]

Notice that \( \sigma_t(t') = \sigma_t'(t') \) since the set of \( Y \)-variables in \( t' \) is the domain of \( \sigma_t \).

We assume that the reduction \( \text{red}_X \) on substitution trees returns an extended substitution defined on \( Y \). The following lemma, regarding \( \text{red}_X(\cdot) \) over substitution trees, will be useful in the correctness proof of \( \delta' \).

**Lemma 11.** For any copyless substitution tree \( t \in \text{Trees}(X, Y) \), if \( \text{red}_X(t) = (t', \sigma_t) \), then \( t' \) is a copyless and constants-free substitution tree and \( \sigma_t \) is a copyless partial substitution. Furthermore, the number of \( Y \)-variables in \( t' \) is bounded by \( |\text{nodes}(t)| \cdot |X| \cdot (|X| \cdot N + 1) \).

**Proof.** We start from showing that \( \sigma_t \) is copyless. Suppose contrary that there exist two registers \( x, y \) such that \( \text{Var}(\sigma_t(x)) \cap \text{Var}(\sigma_t(y)) \neq \emptyset \). We can assume that both registers \( x \) and \( y \) are used in \( t' \) because otherwise \( \sigma_t(x) = \emptyset \) or \( \sigma_t(y) = \emptyset \) and \( \text{Var}(\emptyset) = \emptyset \), which is a contradiction. By definition \( \sigma_t(x) = \sigma_u(x) = \sigma_z(x) \) and \( \sigma_t(y) = \sigma_u(y) = \sigma_z(y) \) for some \( u, u' \in t \) and \( z, z' \in X \), where \( \sigma_z(x), \sigma_z(y) \) are disjoint fragments of expressions \( t(u)(x) \) and \( t(u')(y) \). If \( u \neq u' \) then this violates the condition (17). If \( u = u' \) and \( z \neq z' \) then it violates the condition that \( t(u) \) is a copyless substitution. Otherwise \( u = u' \) and \( z = z' \) then \( x \) and \( y \) are from the same parse tree. It violates Lemma 10 which proves that \( \sigma_t \) is copyless.

Let \( \text{red}_X(p) = (r, \sigma) \) be a reduction on parse trees. By definition the set of \( Y \)-variables in \( r \) is the domain of \( \sigma \) and each \( Y \)-variable occurs only once in \( r \). The reduction \( \text{red}_X \) on \( t \) uses fresh variables for every parse tree. Thus the set of \( Y \)-variable used in \( t' \) is the domain of \( \sigma_t \) and each variable occurs at most once. This proves that \( t' \) is copyless. Since each parse tree in \( t' \) is without constants then the tree \( t' \) is also constant-free.
Regarding the number of fresh $Y$-variables needed to construct $t'$, we know by Lemma 10 that $\text{red}_X(\cdot)$ applied over an expression with alternation bounded by $N$ needs $|X| \cdot N + 1$ new fresh variables. Then $\text{red}_X(\cdot)$ applied over a substitution in $\text{Subs}(X,Y)$ needs at most $|X| \cdot (|X| \cdot N + 1)$ fresh $Y$-variables and over a substitution tree $t$ at most $|=\text{nodes}(t)||X| \cdot (|X| \cdot N + 1)$ fresh $Y$-variables. 

Comming back to the definition of the transition function $\delta'$ of $A'$, recall that states of $A'$ are of the form $(t, B)$ where $t$ is a complete, copyless, and constant-free substitution tree and $B : \text{leaves}(t) \rightarrow Q$ is an injective function. For each transition of the form $\delta'(t, B) = ((t', B'), \sigma)$, we show how to convert $t$ into $t'$ and how to update $B$ into $B'$ by a composing four different processes: extend, prune, shrink, and reduce. In the sequel, we explain each procedure in detail.

**Extend.** The first step is to extend branches in $(t, B)$ to the next states when reading a letter $a \in \Sigma^*$. We define this process formally by the function $\text{extend}((t, B), a) = (t_1, B_1)$ that receives a state $(t, B) \in Q'$ and a letter $a \in \Sigma^*$ and outputs a pair $(t_1, B_1)$ where $t_1$ is a substitution tree and $B_1 : \text{leaves}(t_1) \rightarrow Q$ is a partial injective function. The substitution tree $t_1$ is defined as an extension of $t$ (i.e. nodes$(t) \subseteq \text{nodes}(t_1)$) such that $t_1(u) = t(u)$ whenever $u \in \text{nodes}(t)$ and for every $v \in \text{leaves}(t)$, if there exists a transition $(B(v), a, q, \sigma) \in \delta$, then there exists $i \in \mathbb{N}$ such that $t_1(v \cdot i) = \sigma$. The function $B_1$ is defined only on the new leaves by $B_1(v \cdot i) = q$. Intuitively, $\text{extend}((t, B), a) = (t_1, B_1)$ extends $t$ whenever the state on a leaf of $t$ can evolve to a new state by reading $a$. Notice that $\delta$ is not deterministic and a leaf $v \in \text{leaves}(t)$ could be extended with more than one nodes. We assume that the trees are unordered thus extend is a deterministic procedure.

**Prune.** The problem of the function $\text{extend}((t, B), a) = (t_1, B_1)$ is that there could exist leaves in $t_1$ that are not marked by the function $B_1$ and therefore $(t_1, B_1) \notin Q'$. This happens when for a state $B(v)$ from a leaf $v$ there is no transition $(B(v), a, q, \sigma)$ for any $q$ and $\sigma$, and this branch becomes a dead run. This is the purpose of the function $\text{prune}(t_1, B_1) = (t_2, B_2)$: to prune branches that are dead and to update $B_1$ into a total function $B_2$. Formally, $t_2$ is an induced substitution tree of $t_1$ (i.e. nodes$(t_2) \subseteq \text{nodes}(t_1)$) and $t_2(u) = t_1(u)$ for every $u \in \text{nodes}(t_2)$ such that $u \in \text{nodes}(t_1)$ iff $v \cdot u \in \text{dom}(B_1)$ for some $u \in b1^*$. In other words, we keep only nodes that are ancestors of leaves that are marked by $B_1$ and the remaining nodes are pruned. Finally, we define $B_2 : \text{leaves}(t_2) \rightarrow Q$ by $B_2 = B_1$. This definition is correct because we did not remove any node from the domain of $B_1$. Moreover the paths from the root to leaves in $t_2$ did not change and in particular for every $u \in \text{leaves}(t_2)$ we have $t_2(u) = t_1(u)$.

**Shrink.** By adding and removing branches with the procedures extend and prune it could happen that $t_2$ is not a complete tree and $(t_2, B_2) \notin Q'$ (i.e. $t_2$ could contain internal nodes with just one child). The problem with these nodes is that they are redundant despite that they can be easily removed from $t_1$ by shrinking into one node. For this purpose, we define the procedure $\text{shrink}(t_2, B_2) = (t_3, B_3)$ recursively. We show by induction on the depth of $t_2$ the following properties: $t_3$ is a complete tree; $B_3$ is an injective function from leaves$(t_3)$ into $Q$; range$(B_3) =$
range($B_2$) and that for every $u \in \text{leaves}(t_2)$ there is $u' \in \text{leaves}(t_3)$ such that $B_2(u) = B_3(u')$ and $t_2^i(u) = t_3^i(u')$. In particular this means that the size of $t_3$ is bounded by $2 \cdot |Q|$.

For trees that have only one node we define shrink as the identity function and the properties are kept trivially. Suppose $t_2 = \sigma(r_1, \ldots, r_n)$ for some $\sigma \in \text{Subs}(\mathcal{X}, \mathcal{Y})$. For every $j \in \{1, \ldots, n\}$ let $i_j \in \mathbb{N}$ be the node in $t_2$ corresponding to the root of $r_j$. Then for every leaf $u \in \text{leaves}(r_j)$ the node $i_j \cdot u$ is a leaf in $t_2$. Moreover $\text{leaves}(t_2) = \bigcup_j \{ i_j \cdot u \mid u \in \text{leaves}(r_j) \}$. We define the functions $C_j : \text{leaves}(r_j) \to Q$ by $C_j(u) = B_2(i_j \cdot u)$. Notice that $\text{range}(B_2) = \bigcup_j \text{range}(C_j)$.

If $n > 1$ then shrink($t_2, B_2$) = ($\sigma(r'_1, \ldots, r'_n)$, $B_3$), where shrink($r_j, C_j$) = ($r'_j, C'_j$) and $B_3(i_j \cdot u) = C'_j(u)$ for every $j$ and $u \in C_j$. By induction the properties are kept in shrink($r_j, C_j$) for all $j$. Then it is easy to see that they are also kept for shrink($t_2, B_2$). The final case is when $n = 1$, for simplicity we skip the indexes and write $t_2 = \sigma(r)$ and $C : \text{leaves}(r) \to Q$. Let $r'$ be a tree such that $\text{nodes}(r') = \text{nodes}(r)$, $r'(u) = r(u)$ for every node $u \neq \epsilon$ and $r' (\epsilon) = \sigma \circ r(\epsilon)$. Then we define shrink($t_2, B_2$) = shrink($r', C$). That is, the edge between the root of $\sigma(r)$ and its unique child $r$ is shrunk. By induction the properties are kept in the step from ($r', C$) to ($t_3, B_3$) = shrink($r', C$)). Thus we only have to prove that $\text{range}(B_3) = \text{range}(B_2)$ and that for every $u \in \text{leaves}(t_2)$ there is $u' \in \text{leaves}(t_3)$ such that $t_2^i(u) = t_3^i(u')$. The first property follows from the fact that $\text{range}(B_2) = \text{range}(C) = \text{range}(B_3)$. To prove the second property let $u \in \text{leaves}(t_2)$. By definition there exists an $i$ such that $u = i \cdot v$ and $B_2(i \cdot v) = C(v)$. Let $|u| = k$ then:

$$t_2^i(u) = t_2(\epsilon) \circ t_2(u[\cdot, 1]) \circ t_2(u[\cdot, 2]) \circ \ldots \circ t_2(u[\cdot, k]) = \sigma \circ t_2(u[\cdot, 1]) \circ t_2(u[\cdot, 2]) \circ \ldots \circ t_2(u[\cdot, k])$$

$$= r'^i(v).$$

By the induction assumption there is $u' \in \text{leaves}(t_3)$ such that $r'^i(v) = t_3^i(u')$.

**Reduce.** The pair ($t_3, B_3$) is almost ready after shrinking single-child internal nodes, except that substitutions inside $t_3$ could have constants (e.g. $\text{extend}()$ could have introduced constants). To solve this issue, we apply the $\mathcal{X}$-reduction procedure $\text{red}_\mathcal{X}()$ to reduce all substitutions in $t_3$. Formally, we define the procedure reduce($t_3, B_3$) = ($t_4, B_4, \sigma$) where $\text{red}_\mathcal{X}(t_3) = (t_4, \sigma)$ and $B_3 = B_4$. The application of $\text{red}_\mathcal{X}$ over $t_3$ does not change its tree structure, in particular $\text{nodes}(t_3) = \text{nodes}(t_4)$, but the labels of the tree that are substitutions. Thus $|\text{nodes}(t_3)| = |\text{nodes}(t_4)| \leq 2 \cdot |Q|$. By Lemma 11, this implies that the number of fresh $\mathcal{Y}$-variables needed to apply the procedure $\text{red}_\mathcal{X}()$ is at most $|\text{nodes}(t_3)| \cdot |\mathcal{X}| \cdot (|\mathcal{X}| \cdot N + 1) = 2 \cdot |Q| \cdot |\mathcal{X}| \cdot (|\mathcal{X}| \cdot N + 1)$ which is exactly the size of $\mathcal{Y}$. The reduction $\text{red}_\mathcal{X}$ often chooses fresh variables which is not deterministic. To determine this procedure for every given tree $t_3$ we set a deterministic choice of every fresh variable. With this change the procedure reduce is deterministic.
With the definitions of the procedures extend, prune, shrink and reduce, we are ready to define formally the transition function $\delta'$ of $\mathcal{A}'$. Specifically, for every state $(t, B) \in Q'$ and every $a \in \Sigma^*$ we define:

$$\delta'((t, B), a) = \text{reduce}(\text{shrink}(\text{prune}(\text{extend}((t, B), a)))) = ((t_4, B_4), \sigma)$$

additionally we require that $|\text{range}(B_4) \cap F| = 1$ and that $B_4$ is injective. Otherwise $\delta'((t, B), a)$ is undefined. All procedures: reduce, shrink, prune, extend are deterministic so $\delta'$ is also deterministic. In the next lemma, we show that the definition of $\delta'$ is correct.

**Lemma 12.** For any $(t, B) \in Q'$ and $a \in \Sigma$ let:

$$\text{reduce}(\text{shrink}(\text{prune}(\text{extend}((t, B), a)))) = ((t_4, B_4), \sigma).$$

If $|\text{range}(B_4) \cap F| = 1$ and $B_4$ is injective then $(t_4, B_4) \in Q'$ and $\sigma$ is a copyless substitution over $\mathcal{Y}$.

**Proof.** Assume that we have $(t_1, B_1) = \text{extend}((t, B), a)$, $(t_2, B_2) = \text{prune}(t_1, B_1)$, $(t_3, B_3) = \text{shrink}(t_2, B_2)$ and $(t_4, B_4) = \text{reduce}(t_3, B_3)$. We first check that $t_4$ is a complete, copyless, and constants-free substitution tree in $\text{Trees}(\mathcal{X}, \mathcal{Y})$. We start from showing that $t_4$ is complete. The tree $t_3$ is a result of $\text{prune}(t_2)$, which by definition is a complete tree. This proves completeness because $t_4$ has the same structure as $t_3$.

We show that $t_4$ is copyless and constant-free. The procedure $\text{extend}(\cdot)$ introduces new variables in $t$ only from $\mathcal{X}$ in new nodes. We label the new nodes with substitutions $\sigma$ from $\mathcal{A}$, which by definition are substitutions from $\text{Subs}(\mathcal{X}, \mathcal{Y})$. Moreover it does not introduce new variables from $\mathcal{Y}$ thus the tree $t_1$ is copyless. The procedures prune($\cdot$) and shrink($\cdot$) do not introduce new variables in $t_1$, and $t_2$. This shows that the tree $t_3$ is copyless. By Lemma 11 we conclude that $t_4$ is copyless and constants-free. Since we assumed that $|\text{range}(B_4) \cap F| = 1$ and $B_4$ is injective then this proves that $(t_4, B_4) \in Q'$. Moreover, Lemma 11 also implies that $\sigma$ is a copyless substitution. $\square$

Now we have all the ingredients to prove Lemma 9.

**Proof (of Lemma 9).** The lemma is proved by induction over the size of $w \in \Sigma^*$.

For the base case $w = \epsilon$ by definition $t^0_0 = (t_0, B_0)$ where $t_0$ is a single-node tree labeled with $\sigma_0 \in \text{Subs}(\mathcal{X}, \mathcal{Y})$ and $B_0(\epsilon) = q_0$, where $\sigma_0(x) = \hat{x}$ for every $x \in \mathcal{X}$, where $\hat{x}$ is the copy of $x$ in $\mathcal{X}$. The initial function $\nu^0_0$ is defined as $\nu^0_0(\hat{x}) = \nu_0(x)$ for every $\hat{x} \in \mathcal{X}$ and $\nu^0_0(y) = \emptyset$ for every $y \in \mathcal{Y} \setminus \mathcal{X}$. In this setting

$$\xi \circ t^4(u)(x) = \nu^0_0 \circ \sigma_0(x) = \nu^0_0(\hat{x}) = \nu_0(x).$$

In the initial configuration there is only one run which ends in $q_0$ and $\text{range}(B_0) = \{q_0\}$, which finishes the proof of the base case.
For the induction step suppose that the lemma holds for a word \( w \). Let \( a \in \Sigma \), we show that the lemma also holds for \( w \cdot a \). Let \((q', \nu')\) be a configuration reached by a run of \( \mathcal{A} \) over \( w \cdot a \). By Lemma 6 there is a unique run of \( \mathcal{A} \) on \( w \) that ends in a configuration \((q, \nu)\) such that \((q, a, q', \sigma) \in \delta\) and \( \nu' = \nu \circ \sigma \).

By the induction assumption we have that there is a run of \( \mathcal{A}' \) on \( w \) reaching configuration \((t, B, \xi)\) such that \( Q(w) = \text{range}(B) \) and for every \( x \in \mathcal{X} \) that there exists \( u \in \text{nodes}(t) \) such that \( B(u) = q \) and

\[
\nu = \xi \circ t^i(u).
\] (19)

Let \((t_1, B_1) = \text{extend}((t, B), a)\), \((t_2, B_2) = \text{prune}(t_1, B_1)\), \((t_3, B_3) = \text{shrink}(t_2, B_2)\) and \((t_4, B_4, \tau) = \text{reduce}(t_3, B_3)\). By definition of extend the set \( \text{range}(B_1) \) is the set of all \( p \) such that \((B(v), a, p, \sigma_p) \in \delta\) for some \( \sigma_p \) and \( v \in \text{leaves}(t)\). Since \( Q(w) = \text{range}(B) \) then \( Q(w \cdot a) = \text{range}(B_1) \). Clearly by definition of the procedures prune, shrink, reduce we have \( \text{range}(B_1) = \text{range}(B_2) = \text{range}(B_3) = \text{range}(B_4) \). By the unambiguity of \( \mathcal{A} \) we get that \( |\text{range}(B_4) \cap F| = |Q(w \cdot a) \cap F| = 1 \). Similarly to prove that \( B_4 \) is injective it suffices to show that \( B_1 \) is injective. Suppose contrary that \( B_1 \) maps two different leaves to the same state \( p \). Then by definition of \( B_1 \) there are at least two runs of \( \mathcal{A} \) on \( w \cdot a \) that end in \( q \), which is a contradiction with Lemma 6.

By Lemma 12 we conclude that \( \delta'((t, B), a) = (t_4, B_4) \) and that the automaton \( \mathcal{A}' \) is in configuration \((t_4, B_4, \xi \circ \tau)\) after reading \( w \cdot a \). Since \( \text{range}(B_4) = Q(w \cdot a) \) then there exists \( i \in \mathbb{N} \) such that \( ui \in \text{leaves}(t_1) \), \( B_1(u \cdot i) = q' \), and \( t_1(ui) = \sigma \). If we compose both sides of Equation 19 with \( \sigma \) then we get:

\[

\text{\underbrace{\nu \circ \sigma =}^{\xi \circ t^i(u) \circ \sigma} \text{\underbrace{\nu'}_{t^i(u \cdot i)}}}

\]

Thus we get

\[

\nu' = \xi \circ t^i(u \cdot i).
\]

We know that the procedure \( \text{prune}() \) and \( \text{shrink}() \) preserve the outputs of the collapse operation. Thus there exists \( v \in \text{nodes}(t_3) \) such that \( B_3(v) = B'(v) = q' \) and

\[

\nu' = \xi \circ t^i_3(v).
\]

Given that \( (t_4, B_4, \tau) = \text{reduce}(t_3, B_3) \) is an \( \mathcal{X} \)-reduction, we know that \( \tau \circ t^i_4(v) = t^i_3(v) \). By replacing the last equation in the above formula, we get

\[

\nu' = \xi \circ \tau \circ t^i_3(v)
\]

which proves the induction step. \( \square \)

Lemma 9 in particular proves that \( \delta' \) is a total function when \( Q' \) is restricted to the reachable states \( Q'_r \). For this reason we restrict the set of states to \( Q'_r \). With \( Q'_r \) as the set of states the automaton \( \mathcal{A}' \) is a deterministic CRA.
To conclude the proof we show that $A$ and $A'$ have the same output on every word $w$. Let $(q, \nu)$ be the configuration of the unique accepting run of $A$ on $w$ and let $((t, B), \xi)$ be the configuration of $A'$ of the run on $w$. By Lemma 9 there exists a node $u$ such that $u \in \text{leaves}(t)$ such that $B(u) = q$ and 

$$\nu = \xi \circ t^t(u).$$

The output of $A'$ on $w$ is defined as $\xi \circ \mu'(t, B)$. By definition $q \in F$ and $|\text{range}(B) \cap F| = 1$ thus $q$ is the unique accepting state in $\text{leaves}(t)$. By definition $\mu'(t, B) = t^t(u) \circ \mu(q)$. Thus we get the following equalities for the output of $A'$ on $w$:

$$\xi \circ \mu'(t, B) = \xi \circ t^t(u) \circ \mu(q) = \nu \circ \mu(q).$$

This finishes the proof because the expression $\nu \circ \mu(q)$ is the output of $A$ on $w$.

References

1. S. Almagor, U. Boker, and O. Kupferman. What’s decidable about weighted automata? In ATVA, pages 482–491, 2011.
2. R. Alur, L. D’Antoni, J. Deshmukh, M. Raghoothaman, and Y. Yuan. Regular functions and cost register automata. In Proceedings of the 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science, pages 13–22. IEEE Computer Society, 2013.
3. K. Culik II and J. Kari. Image compression using weighted finite automata. In Mathematical Foundations of Computer Science 1993, pages 392–402. Springer, 1993.
4. M. Droste and P. Gastin. Weighted automata and weighted logics. Theor. Comput. Sci., 380(1-2):69–86, 2007.
5. M. Droste, W. Kuich, and H. Vogler. Handbook of Weighted Automata. Springer, 1st edition, 2009.
6. E. Filiot, R. Gentilini, J.-F. Raskin, V. Raman, and S. Suresh. Finite-valued weighted automata. In 34th International Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS 2014), volume 29, pages 133–145. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2014.
7. J. E. Hopcroft and J. D. Ullman. Introduction to Automata Theory, Languages and Computation. Addison-Wesley, 1979.
8. I. Klimann, S. Lombardy, J. Mairesse, and C. Prieur. Deciding unambiguity and sequentiality from a finitely ambiguous max-plus automaton. Theor. Comput. Sci., 327(3):349–373, 2004.
9. S. Kreutzer and C. Riveros. Quantitative monadic second-order logic. In Proceedings of the 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science, pages 113–122. IEEE Computer Society, 2013.
10. D. Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. In ICALP, pages 101–112, 1992.
11. M. Mohri. Finite-state transducers in language and speech processing. Computational Linguistics, 23(2):269–311, 1997.
12. C. H. Papadimitriou. Computational complexity. Addison-Wesley, 1993.
13. J. Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.
14. M. P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.
15. A. Weber. Finite-valued distance automata. *Theor. Comput. Sci.*, 134(1):225–251, 1994.
16. A. Weber and H. Seidl. On the degree of ambiguity of finite automata. *Theoretical Computer Science*, 88(2):325–349, 1991.