# Determinisation of Finitely-Ambiguous Copyless Cost Register Automata

**Théodore Lopez**
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
theodore.lopez@univ-amu.fr

**Benjamin Monmege**
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
benjamin.monmege@univ-amu.fr

**Jean-Marc Talbot**
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
jean-marc.talbot@univ-amu.fr

## Abstract

Cost register automata (CRA) are machines reading an input word while computing values using write-only registers: values from registers are combined using the two operations, as well as the constants, of a semiring. Particularly interesting is the subclass of copyless CRAs where the content of a register cannot be used twice for updating the registers. Originally deterministic, non-deterministic variant of CRA may also be defined: the semantics is then obtained by combining the values of all accepting runs with the additive operation of the semiring (as for weighted automata). We show that finitely-ambiguous copyless non-deterministic CRAs (i.e. the ones that admit a bounded number of accepting runs on every input word) can be effectively transformed into an equivalent copyless (deterministic) CRA, without requiring any specific property on the semiring. As a corollary, this also shows that regular look-ahead can effectively be removed from copyless CRAs.

**2012 ACM Subject Classification** Theory of computation → Quantitative automata

**Keywords and phrases** Cost-register automata, Look-ahead removal, Unambiguity, Determinisation

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2019.75

**Funding** This work has been funded by the DeLTA project (ANR-16-CE40-0007).

## 1 Introduction

Quantitative languages extend "classical" languages by associating with each word a weight or a cost from an algebraic structure. Such algebraic structures could be monoids, semirings, fields or any other convenient structures. Quantitative languages have been successfully applied to various domains such as natural language processing [15] or modelisation of stochastic systems [17, 16]. The seminal work on quantitative languages from Schützenberger [18] introduces the model of weighted automata, that associates with each word a weight from a semiring. The weight of a run is the *product* of its transition weights whereas the weights of the multiple runs on a single word (due to non-determinism) are combined by *sum*. Classical word languages are then the particular case of weighted languages over the Boolean semiring. There have been a long line of research that studied properties of weighted automata [10]. As for finite-state automata, two-way [6] and alternating [12, 7] automata have been considered, as well as extensions to infinite words [8].

Recently, Alur et al. [3] introduced another automata model for defining mappings from words to some algebraic structures (in particular semirings), named Cost Register Automata (CRA). These are deterministic machines equipped with a finite collection of registers storing values: while reading the input word, each transition reads an input letter and updates the registers by combining the current contents of registers and values from the considered

algebraic structure. A distinguished subclass of CRAs is the one of *copyless* CRAs: intuitively, for such a machine, the content of a register cannot be used twice for updating the registers. Properties regarding expressiveness of this class have been studied in [14, 13]. The decision problem of equivalence of copyless CRA on the tropical semiring is proved undecidable in [1]. Some other works have also considered the register minimisation problem (computing the smallest number of registers to define a particular function) for copyless CRAs [5, 9].

Following [14], we consider both deterministic and non-deterministic copyless CRAs on semirings (to distinguish them, we call NCRA the class of non-deterministic CRAs). Non-determinism is resolved, as in weighted automata, by the sum operation of the semiring: the value associated with some input word is computed as the sum of the values computed by each accepting run on this input. We investigate a limited form of non-determinism, $k$-ambiguity: a non-deterministic CRA is said to be $k$-ambiguous if it has at most $k$ accepting runs per input. In the context of weighted automata, $k$-ambiguous weighted automata are strictly more powerful than deterministic weighted automata, yet less powerful than weighted automata, leading to an appealing class of weighted languages with good decision properties (the equivalence problem becomes decidable for $k$-ambiguous weighted automata over the $(\max, +)$-semiring [11]). Surprisingly, in the context of CRAs, our main result is:

▶ **Theorem 1.** *Every finitely-ambiguous copyless* NCRA *can be effectively transformed into an equivalent copyless (deterministic)* CRA*.*

Moreover, the example developed in [14, Theorem 2] allows one to build a linearly-ambiguous copyless NCRA that cannot be recognised with a finitely-ambiguous copyless NCRA, showing that our result cannot be improved regarding ambiguity. An alternative way to resolve non-determinism is to consider a regular look-ahead. When reading a word from left to right, the look-ahead provides some (regular) information about the unread suffix of the word that allows to determine the unique transition to be applied at each step. In this case, the machine is said to be deterministic with look-ahead. In [3], this class is introduced and named CRA-RLA. It is proved there that for copyless CRA-RLA, the look-ahead can be removed preserving the copyless property provided that the considered algebraic structure is extended with unary mappings (using the so-called *streaming string-to-tree transducers* [2] as an intermediate step). In [14], Mazowiecki and Riveros proved that copyless CRA, unlike weighted automata, are not closed under reverse but claimed that "Like for unambiguous copyless CRAs, we do not know if extending copyless CRAs with regular look-ahead results in a more expressive model". However, they defined the subclass of bounded-alternation copyless CRAs which are closed under reverse and for which deterministic look-aheads do not increase expressiveness.

A look-ahead can be given as a complete co-deterministic automaton $B$ and transitions of the CRA $A$ are then parameterised by some state of $B$. It is folklore that one can compute the product of $A$ and $B$ to obtain a machine equivalent to $A$ which is look-ahead free, now non-deterministic, but still unambiguous. This construction still applies when $A$ is a copyless CRA and the product yields an unambiguous copyless CRA. Therefore, by Theorem 1, we close the open problem stated in [14]:

▶ **Theorem 2.** *Every copyless CRA with look-ahead can be effectively transformed into an equivalent copyless CRA.*

The article is structured as follows: in Section 2, we define CRAs as well as several subclasses (copyless CRAs, $\diamond$-less CRAs, and bounded-copy CRAs). They are defined by means of flow graphs representing the flow of registers during runs in an abstract way. The proof of Theorem 1 is given as a cascade of three transformations leading from one subclass to another one, that are described successively in Sections 3, 4 and 5.

## 2 Cost register automata

**Terms and substitutions.** Fix a semiring $\mathbb{S} = (S, +, \times, 0, 1)$ and a finite set of variables $\mathcal{X}$ disjoint from $S$. We denote by $\mathrm{Term}(\mathcal{X})$ the set of *terms* generated by the grammar:

$$t ::= s \mid x \mid t + t \mid t \times t$$

where $s \in S$ and $x \in \mathcal{X}$. For a term $t \in \mathrm{Term}(\mathcal{X})$, we denote by $\mathrm{Var}(t)$ its set of variables. We call $t$ a *ground term* if $\mathrm{Var}(t) = \emptyset$, and then define $[\![t]\!] \in S$ to be the evaluation of $t$ with respect to $\mathbb{S}$. We call a term $t \in \mathrm{Term}(\mathcal{X})$ *copyless* if every variable appears at most once in $t$. In the following, we will represent terms as binary trees, where leaves are labelled by variables or constants, and internal nodes are labelled by operations of the semiring.

▶ **Example 3.** On the semiring $(\mathbb{N}, +, \times, 0, 1)$, examples of terms are $3x + 1$ (we often make implicit the product operator) or $y + 2zx + 3$, making use of the associativity of both operators to drop useless parentheses. On the semiring $(\mathbb{Z} \cup \{\infty\}, \min, +, \infty, 0)$, the same terms are written $\min(3 + x, 1)$ and $\min(y, 2 + z + x, 3)$. Other examples of terms on the non-commutative semiring $(\mathfrak{P}(\{a, b\}^*), \cup, \cdot, \emptyset, \{\varepsilon\})$ of languages over alphabet $\{a, b\}$ are $\{a\}x\{\varepsilon, aab\} \cup y \cup \{b\}$. All these terms are copyless.

A *substitution* is a mapping $\sigma : \mathcal{X} \to \mathrm{Term}(\mathcal{X})$. We denote the set of all substitutions over $\mathcal{X}$ by $\mathrm{Subs}(\mathcal{X})$. If $t$ is a term, we let $[x \mapsto t]$ be the substitution defined by $[x \mapsto t](x) = t$ and $[x \mapsto t](y) = y$ for all variables $y \neq x$. A *ground substitution* $\sigma$ is a substitution where the term $\sigma(x)$ is ground for every $x \in \mathcal{X}$. Substitutions are extended canonically to a term morphism, and may thus be composed: $\sigma_1 \circ \sigma_2(x) = \sigma_1(\sigma_2(x))$.

A *valuation* is defined as a substitution of the form $\nu \colon \mathcal{X} \to S$. We denote the set of all valuations over $\mathcal{X}$ by $\mathrm{Val}(\mathcal{X})$. Clearly, any valuation $\nu$ composed with a substitution $\sigma$ defines a ground substitution. We say that two terms $t_1$ and $t_2$ are equivalent (denoted by $t_1 \equiv t_2$) if $[\![\nu(t_1)]\!] = [\![\nu(t_2)]\!]$ for every valuation $\nu \in \mathrm{Val}(\mathcal{X})$. Similarly, we say that two substitutions $\sigma_1$ and $\sigma_2$ are equivalent (denoted by $\sigma_1 \equiv \sigma_2$) if $\sigma_1(x) \equiv \sigma_2(x)$ for every $x \in \mathcal{X}$.

**Non-deterministic cost register automata.** A non-deterministic CRA (NCRA) over the semiring $\mathbb{S}$ is a tuple $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \Delta, I, \nu_{ini}, F, \varphi)$, where $Q$ is a finite set of states, $\Sigma$ is the input alphabet, $\mathcal{X}$ a finite set of registers, $\Delta \subseteq Q \times \Sigma \times \mathrm{Subs}(\mathcal{X}) \times Q$ is the finite transition relation with updates of the registers, $I \subseteq Q$ is the set of initial states, $\nu_{ini} \colon I \to \mathrm{Val}(\mathcal{X})$ defines an initial valuation of the registers for each initial state, $F$ is the set of final states and $\varphi \colon F \to \mathrm{Term}(\mathcal{X})$ the final output function. Transition $(q, a, \sigma, q')$ is denoted by $q \xrightarrow{a|\sigma} q'$.

A configuration of $\mathcal{A}$ is a tuple $(q, \nu)$ where $q \in Q$ and $\nu \in \mathrm{Val}(\mathcal{X})$ represents the current values in the registers of $\mathcal{A}$. Given a word $w = a_1 \cdots a_n \in \Sigma^*$, a *run* $\rho$ of $\mathcal{A}$ over $w$ is a sequence of configurations linked by transitions $(q_0, \nu_0) \xrightarrow{a_1|\sigma_1} (q_1, \nu_1) \xrightarrow{a_2|\sigma_2} \cdots \xrightarrow{a_n|\sigma_n} (q_n, \nu_n)$ such that $q_0 \in I$, $\nu_0 = \nu_{ini}(q_0)$, for $1 \leq i \leq n$, $q_{i-1} \xrightarrow{a_i|\sigma_i} q_i$ is a transition and $\nu_i(x) = [\![\nu_{i-1} \circ \sigma_i(x)]\!]$ for each $x \in \mathcal{X}$. A run is accepting if it ends in a final state $q_n \in F$. The output of an accepting run $\rho = (q_0, \nu_0) \xrightarrow{a_1|\sigma_1} \cdots \xrightarrow{a_n|\sigma_n} (q_n, \nu_n)$, denoted by $[\![\rho]\!]$, is the value $[\![\nu_n(\varphi(q_n))]\!]$. The output of $\mathcal{A}$ over $w$ is defined as $[\![\mathcal{A}]\!](w) = 0$ if there is no accepting run of $\mathcal{A}$ over $w$, and $[\![\mathcal{A}]\!](w) = \sum_\rho [\![\rho]\!]$ over all accepting runs $\rho$ over $w$ otherwise. Two NCRAs are said *equivalent* if they compute the same output on every input word.

For some positive $k$, a NCRA $\mathcal{A}$ is *$k$-ambiguous* if there are at most $k$ accepting runs over each word $w$. $\mathcal{A}$ is *finitely-ambiguous* if it is $k$-ambiguous for some $k$, and *unambiguous* if it is 1-ambiguous. A NCRA is said to be *deterministic*, and denoted by CRA, if $I$ is a singleton and for all $q \in Q$, $a \in \Sigma$, there exists at most one state $q' \in Q$ and one register update $\sigma \in \mathrm{Subs}(\mathcal{X})$ such that $q \xrightarrow{a|\sigma} q' \in \Delta$.

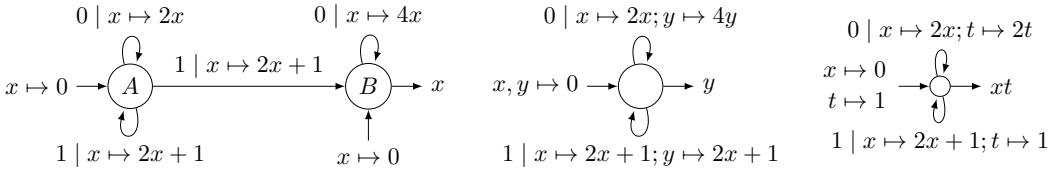



**Figure 1** Three equivalent automata: an unambiguous copyless NCRA $\mathcal{A}_n$ (on the left), a $\diamond$-less CRA $\mathcal{A}_d$ (in the middle), a copyless CRA $\mathcal{A}_c$ (on the right).

▶ **Example 4.** Consider the copyless NCRA $\mathcal{A}_n$ depicted on the left of Figure 1 over the alphabet $\{0,1\}$ and the semiring $(\mathbb{N},+,\times,0,1)$. It has a single register $x$, so that we hereby denote configurations by pairs $(q,\nu(x))$. Every word $w \in \{0,1\}^*$ has two runs, only one being accepting: thus $\mathcal{A}_n$ is unambiguous. For instance, on the word $w = 10100$, the two runs are

$$(A,0) \xrightarrow{1|x\mapsto 2x+1} (A,1) \xrightarrow{0|x\mapsto 2x} (A,2) \xrightarrow{1|x\mapsto 2x+1} (A,5) \xrightarrow{0|x\mapsto 2x} (A,10) \xrightarrow{0|x\mapsto 2x} (A,20)$$
$$(A,0) \xrightarrow{1|x\mapsto 2x+1} (A,1) \xrightarrow{0|x\mapsto 2x} (A,2) \xrightarrow{1|x\mapsto 2x+1} (B,5) \xrightarrow{0|x\mapsto 4x} (B,20) \xrightarrow{0|x\mapsto 4x} (B,80)$$

Therefore, $A$ is a state computing the integer value of the word as binary representation with less significant bits first. To accept, we must jump into the unique final state $B$ while reading the last 1 (or start directly in $B$ if the word contains only 0s), then multiplying by 4 for each remaining 0: it is as if each 0 of the last block of 0s is considered to be duplicated by the CRA. The same function can also be recognised by a deterministic CRA, using two registers $x$ and $y$, depicted in the middle of Figure 1. Instead of using non-determinism to guess the last 1 of the word, $\mathcal{A}_d$ always computes both the multiplications by 2 and 4 in separate registers $x$ and $y$ when reading 0. On each letter 1 though, the content of register $y$ is reset to the same content as register $x$: operationally, this means that the content of register $x$ must be duplicated when reading 1.

**Flow of registers.** A crucial notion in NCRAs is their ability to copy, or not, contents of registers into several registers. A NCRA is therefore called *copyless* if no register updates $\sigma$ of $\Delta$ or terms of $\varphi$ copy some register (see [3]). A more graphical definition of copyless can be achieved by gathering the *flows* of registers in a notion of *flow graphs*, that we define now, inspired by a close notion of dependency graph introduced in [4]:

▶ **Definition 5.** *A* flow graph *over the set of variables $\mathcal{X}$ is a (finite) directed acyclic (multi)graph $(V,E)$ where $V = (\mathcal{X} \times \{0,1,\dots,\ell_{max}\}) \uplus \{\Omega\}$ (with $\ell_{max} \geq 0$) is a finite set of vertices $(x,\ell)$ where $\ell$ is called the* layer *of the vertex (with $\ell_{max}$ being the maximal layer of the flow graph), and $E\colon V^2 \to \mathbb{N}$ being a multiset of edges satisfying:*

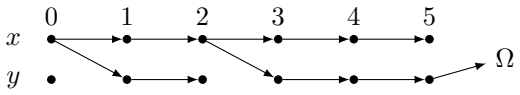
1. *$E$ is consistent with the layers: $E((x_1,\ell_1),(x_2,\ell_2)) \neq 0 \implies \ell_2 = \ell_1 + 1$ , and*
2. *$\Omega$ has no outgoing edges and all its ingoing edges come from the maximal layer: $E((x,\ell),\Omega) \neq 0 \implies \ell = \ell_{max}$ .*

Each run $\rho = q_0 \xrightarrow{a_1|\sigma_1} q_1 \xrightarrow{a_2|\sigma_2} \cdots \xrightarrow{a_k|\sigma_k} q_k$ of a NCRA $\mathcal{A}$ is associated with the flow graph $G_{\mathcal{A}}(\rho) = (V,E)$ defined by: $V = \mathcal{X} \times \{0,\dots,k\} \cup \{\Omega\}$; for $\ell \in \{1,\dots,k\}$, and $x,y \in \mathcal{X}$, $E((x,\ell-1),(y,\ell))$ is the number of occurrences of the variable $x$ in $\sigma_\ell(y)$; $E((x,k),\Omega)$ is the number of occurrences of $x$ in $\varphi(q_k)$, and 0 if $q_k \notin F$.

Copyless restriction of NCRAs can be recovered directly on the flow graphs generated by their runs. For this reason, we say that a vertex of a flow graph is a *copy vertex* if it is the source of at least two edges. We use those to define two other related properties of the flow graphs that will be used in the following.

▶ **Definition 6.** *A flow graph* $(V, E)$ *is* diamondless *(shorten as* $\diamond$-less *in the following) if there is at most one path linking every pair of vertices. It is called* $k$-copy *if every vertex can reach at most* $k$ *copy vertices. We say that it is* copyless *if it is* 0*-copy.*

▶ **Example 7.** Consider once again the word 10100, and the unique run of the CRA $\mathcal{A}_d$ of Figure 1. Here is a pictural representation of the associated flow graph:



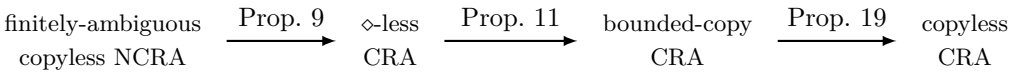

One can observe on this picture that $\mathcal{A}_d$ is not copyless, since register $x$ is copied when reading letter 1. However, there are no diamonds (neither in this particular run, nor in any possible run), which means that $\mathcal{A}_d$ is $\diamond$-less.

By extension, a NCRA is said to be copyless (resp. $\diamond$-less or $k$-copy) if the flow graphs of all possible accepting runs of the NCRA are copyless (resp. $\diamond$-less or $k$-copy). It is said to be bounded-copy if it is $k$-copy for a certain value $k$. This alternative definition of copyless NCRA is equivalent to one of [3], whenever CRAs are supposed to be trimmed (i.e. all states are reachable from the initial ones and can reach a final state). Note that in the flow graphs of a $\diamond$-less CRAs, the multiset $E$ is indeed a set (all pairs evaluate to 0 or 1). However, this does not imply that the CRA is copyless since a register can appear in the updates of two different registers. The $\diamond$-less property simply ensures that every register will flow at most once in the final output, in every possible execution: there may exist copies, but when it is the case, we are sure that at most one copy will resist up to the end; however, this exact copy can not be known yet as it might depend on the input word.

**Contribution.** Our main result is Theorem 1 stated already in the introduction. As the class of copyless CRA is obviously included into the class of finitely-ambiguous copyless NCRA, this result implies that these two classes define the same family of functions.

▶ **Example 8.** Our running example can indeed be recognised by the copyless CRA $\mathcal{A}_c$ on the right of Figure 1 which keeps in a register $x$ the binary value of the input, and keeps in another register $t$ the powers of 2 corresponding to the current longest suffix of letter 0. We multiply $x$ by $t$ in the final output function, and reset $t$ when reading a letter 1.

Our construction is split into a cascade of transformations detailed in the next sections:




The first step in the construction is given in Section 3: a determinisation procedure of the finitely-ambiguous copyless CRA allows us to build a (deterministic) CRA, that may not be copyless, thus trading non-determinism for copies. This new CRA will indeed be $\diamond$-less. The second step is to reduce the number of copies in order to build an equivalent CRA that is bounded-copy: this is not trivial since the $\diamond$-less property does not forbid that unboundedly many copies can be performed. The idea is to delay copies until a moment where we know that enough copies have become useless. This more difficult step is the main contribution of this article and presented in Section 4. Finally, it remains to show in Section 5 how to remove all copies of the bounded-copy CRA, by replicating the registers as many times as needed.

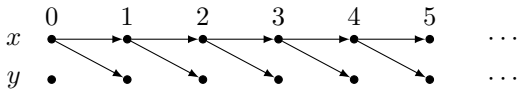
## 3 From finitely-ambiguous CRAs to diamondless CRAs

As a first step in our construction, we start by transforming every $k$-ambiguous NCRA into a $\diamond$-less CRA: this is a determinisation step where we maintain replicas of the registers, indexed by the states and a number in $\{1, \dots, k\}$ to distinguish the $k$ possible runs.

▶ **Proposition 9.** *For every finitely-ambiguous copyless* NCRA*, we can compute an equivalent* $\diamond$*-less* CRA*.*

▶ **Example 10.** Starting from the unambiguous copyless NCRA $\mathcal{A}_n$ of Figure 1, we obtain a $\diamond$-less CRA $\mathcal{A}'$ isomorphic to the CRA $\mathcal{A}_d$: the single state represents the set $\{A, B\}$ of states of $\mathcal{A}$, while registers $(x, A)$ and $(x, B)$ of $\mathcal{A}'$ are the registers $x$ and $y$ on the picture.

The next step aims to limit the number of copies of the $\diamond$-less CRA. Indeed, $\diamond$-less CRAs are not necessarily copyless, and may even copy certain registers an unbounded number of times (in arbitrarily long runs). This is the case for the $\diamond$-less CRA $\mathcal{A}_d$ of Figure 1: for instance, the flow graph associated with the run over the word $11111\cdots$ is of the form




and has thus an unbounded number of copies of register $x$, that are all reachable from vertex $(x, 0)$; thus, $\mathcal{A}_d$ is not bounded-copy.

## 4 From diamondless CRAs to bounded-copy CRAs

In this section, we prove the most difficult step of the overall construction:

▶ **Proposition 11.** *For every* $\diamond$*-less* CRA*, we can construct an equivalent bounded-copy* CRA*.*

As seen in the example of the previous section, this result is not straightforward as $\diamond$-less CRAs may have an unbounded number of copies. Our approach somehow extends the one developed for streaming string transducers in [4]. However, the proof is simpler in the case of transducers where only a single operation exists (the product of a monoid). This allows one to represent in an alternative way the valuation of a register as a product of constants and other registers: instead of storing in registers the value of these products, it is going to be abstracted, storing in fresh registers the constant values separating two consecutive registers in products. In the setting of semirings, with two operations, it is much more intricate to do so, since the content of a register has to be viewed now as a term involving both operations. Hence, it is less clear a priori what could be the constants separating two "consecutive" registers of this term. We start by clarifying this, introducing special terms we call *shapes*, associated with *coefficients* that are these separating constants. Similar kind of shapes are used in [14, Theorem 3] to remove unambiguous non-determinism from bounded-alternation NCRAs, i.e. NCRAs that can only alternate a bounded number of times between sums and products in the register updates. The treatment of shapes is more complex in our case since we have no such limitation on alternations.

**Shapes and coefficients.** The *shape* of a term $t$ over the set of variables $\mathcal{X}$ is a term with no constants, but with additional variables, obtained as follows. First, all constants are removed from the term to obtain a term $\tilde{t}$ where all leaves are labelled with variables of $\mathcal{X}$: for instance, if $t = (3x_1(5+2)+4) \times (2x_2) + 3$, then $\tilde{t} = x_1 \times x_2$. Doing so, we lose much

information on the term, that we then recover by decorating every subterm $t'$ of $\tilde{t}$ with some fresh coefficients $\alpha$, $\beta$, $\gamma$, replacing the root of $t'$ by $\alpha t'\beta + \gamma$: $\alpha$, $\beta$ represent respectively the left and the right multiplicative coefficient, and $\gamma$ the additive coefficient. The *shape* of $t$ is the term obtained by decorating each subterm of $\tilde{t}$. It is associated with a valuation $\chi$ mapping each coefficient of the shape to its value. On the example, the shape obtained is $\alpha_3[(\alpha_1 x_1 \beta_1 + \gamma_1) \times (\alpha_2 x_2 \beta_2 + \gamma_2)]\beta_3 + \gamma_3$, and the valuation of its coefficients is defined by $\chi(\alpha_1) = 3$, $\chi(\beta_1) = 7$, $\chi(\gamma_1) = 4$, $\chi(\alpha_2) = 2$, $\chi(\beta_2) = \chi(\alpha_3) = \chi(\beta_3) = 1$, $\chi(\gamma_2) = 0$, $\chi(\gamma_3) = 3$.[1] For the special case of a constant term $t$ (without any variables), the shape is reduced to a special coefficient $\omega$.

Shapes are canonical way to store terms. In particular, note that there is only a finite number of shapes of *copyless* terms over $\mathcal{X}$, denoted by $\mathrm{Shape}(\mathcal{X})$, though there are infinitely many possible coefficient valuations associated with these shapes. This will allow us to store the shapes in states of the bounded-copy CRA, while keeping in registers the valuations of coefficients (that could not fit in a CRA with a finite number of states).

Given a shape $\tau$ and an associated coefficient valuation $\chi$, we denote by $\chi(\tau)$ the term obtained by replacing each coefficient of $\tau$ by its value: thus, $\chi(\tau)$ is a term over variables $\mathcal{X}$.

▶ **Proposition 12.** *There is a linear-time algorithm that, given a term $t$, builds a shape $\tau$ of $t$ and a coefficient valuation $\chi$, such that $t$ and $\chi(\tau)$ are equivalent terms.*

**Unfolding of a CRA.** In the rest of the section, we let $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \Delta, I = \{q_{ini}\}, \nu_{ini}, F, \varphi)$ be a $\diamond$-less CRA. We construct another CRA $\mathcal{A}_\infty = (Q', \Sigma, \mathcal{X}', \Delta', I = \{q'_{ini}\}, \nu'_{ini}, F', \varphi')$ equivalent to $\mathcal{A}$, by unfolding: states contain shapes, and registers store the valuations of all corresponding coefficients. At first, $\mathcal{A}_\infty$ will thus have an infinite number of states: we explain afterwards how to reduce it to a finite number.

States of $\mathcal{A}_\infty$ are pairs $(q, s)$, where $q$ is a state of $\mathcal{A}$ and $s$ maps each pair $(x, \ell) \in \mathcal{X} \times \{1, \dots, \ell_{max}\}$, where $\ell_{max} \in \mathbb{N}$ depends on the state of $\mathcal{A}_\infty$, to a term of $\mathrm{Shape}(\mathcal{X} \times \{\ell - 1\})$, and each pair $(x, 0)$ to a shape of the form $\omega$: $s$ records the register updates applied so far, keeping only the shapes in memory. We call $s$ the *shape substitution* of the state. Moreover, we enforce all coefficients appearing in all the shapes of $s$ to be different. Notice that such a state $(q, s)$ is associated uniquely with a flow graph $G = (\mathcal{X} \times \{0, \dots, \ell_{max}\} \uplus \{\Omega\}, E)$ where $E$ is the set[2] defined by $((x, \ell - 1), (y, \ell)) \in E$ iff $(x, \ell - 1)$ appears in $s(y, \ell)$, and $((x, \ell_{max}), \Omega) \in E$ iff $x \in \varphi(q)$. In the following, we use the notions of layers originating from flow graphs directly on $s$. All vertices of the flow graph that cannot reach a vertex of the maximal layer are useless : their value will not be used in the output of the CRA. Hence, we clean up $s$ by mapping them to constant coefficients $\omega$. In the following, we always consider that shape substitutions $s$ are cleaned up this way.

Registers of $\mathcal{A}_\infty$ are all the possible coefficients appearing in its states (notice that there is an infinite number of them). However, at each point of the execution of $\mathcal{A}_\infty$, only coefficients that appear in the shape substitution of the current state are initialised, other registers being useless at this point. We may call *coefficients* the registers of $\mathcal{A}_\infty$ to emphasise their role.

The initial state is $q'_{ini} = (q_{ini}, s_{ini})$ with $s_{ini}$ mapping $(x, 0)$ to a distinct coefficient $\omega_x$ for all $x$ ($\ell_{max} = 0$ for this state); hence, only coefficients $\omega_x$ appear in these shapes of $q'_{ini}$ and their value is given by $\nu'_{ini}(q'_{ini})(\omega_x) = \nu_{ini}(q_{ini})(x)$. All other registers can be set to 0 (their value will never be used in the following).

[1] Notice that the use of separate left and right multiplicative is only necessary in non-commutative semirings: in commutative ones, we could merge both of them in a single $\alpha$ coefficient.

[2] $E$ turns out to be not some arbitrary multiset, as we start from a $\diamond$-less CRA $\mathcal{A}$

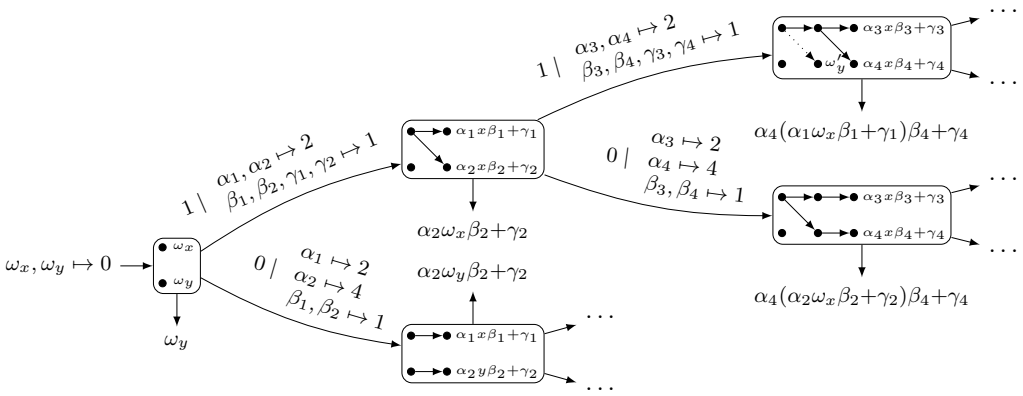



**Figure 2** An infinite CRA equivalent to the $\diamond$-less CRA $\mathcal{A}_d$.

Final states of $F'$ are all pairs $(q, s)$ with $q \in F$, and the associated final output $\varphi'$ is the term using only coefficients obtained by applying the shape substitution as many times as the number of layers from the maximal layer, in order to remove all variables $(x, \ell)$: $\varphi'(q, s) = \big[x \mapsto s^{\ell_{max}+1}(x, \ell_{max}) \quad \forall x \in \mathcal{X}\big]\big(\varphi(q)\big)$ where $s^h$ is the composition of $s$ with itself $h$ times, and $\ell_{max}$ is the maximal layer of the flow graph associated with $s$.

We finally describe the transitions of $\mathcal{A}_\infty$. For all states $(q, s) \in Q'$ and transitions $q \xrightarrow{a|\sigma} q'$ in $\Delta$, we add a transition $(q, s) \xrightarrow{a|\sigma'} (q', s')$ in $\Delta'$, where $s'$ is the extension of $s$ with an additional layer. If the maximal layer of $s$ is $\ell_{max}$, then the maximal layer of $s'$ is $\ell_{max} + 1$. For the additional maximal layer $\ell_{max} + 1$, for all $x \in \mathcal{X}$, the term $\sigma(x)$ associated with the update of the register $x$ in $\mathcal{A}$ can be decomposed, by Prop. 12, into a shape $\tau$ and a valuation $\chi$ of its fresh coefficients: we have $\chi(\tau) \equiv \sigma(x)$. Then, we let $s'(x, \ell_{max} + 1)$ be the shape $\tau$ in which every variable $y$ is replaced by $(y, \ell_{max})$. Corresponding (fresh) coefficients $\kappa$ of $\tau$ are set to their value in $\chi$: $\sigma'(\kappa) = \chi(\kappa)$. Layers $0, 1, \ldots, \ell_{max}$ are kept intact, except that vertices that can no longer reach layer $\ell_{max} + 1$ are mapped to a constant shape $\omega$. More precisely, for all $\ell \in \{0, 1, \ldots, \ell_{max}\}$ and $x \in \mathcal{X}$, if $(x, \ell)$ can reach layer $\ell_{max} + 1$ (in the so-extended flow graph) then $s'(x, \ell) = s(x, \ell)$ and the corresponding coefficients $\kappa$ remain the same by the update $\sigma'(\kappa) = \kappa$. Otherwise $s'(x, \ell)$ is mapped to a constant shape $\omega$ and $\sigma'(\omega) = 0$: coefficients of $s(x, \ell)$ are freed, e.g. by resetting them to 0 by $\sigma'$.

▶ **Example 13.** We illustrate this construction on the $\diamond$-less CRA $\mathcal{A}_d$ of Figure 1. A portion of the infinite CRA is shown in Figure 2. We depict in each state the associated flow graph (without the output vertices) as well as the shapes that are different than the corresponding shapes in the predecessor state. In the updates, we only show the values different from 0. Notice the dotted edge in the state reached after having read word 11: this edge disappears from the flow graph since the vertex $(y, 1)$ can no longer reach the maximal layer.

The infinite CRA $\mathcal{A}_\infty$ satisfies the following invariant: each run $\rho$ of $\mathcal{A}$, ending in state $q$, is bijectively mapped to a run $\rho'$ of $\mathcal{A}_\infty$ that ends in a state $(q, s)$ associated with a flow graph isomorphic to $G_{\mathcal{A}}(\rho)$. Moreover,

▶ **Invariant 14.** *For all words $w$, if $(q, \nu)$ is the (unique) configuration reached by $\mathcal{A}$ over word $w$, the configuration $((q', s), \nu')$ reached by $\mathcal{A}'$ reading $w$ satisfies: $q = q'$ and for every $x \in \mathcal{X}$, $\nu(x) = [\![\nu' \circ s^{\ell_{max}+1}(x, \ell_{max})]\!]$ where $\ell_{max} + 1$ is the number of layers in $s$.*

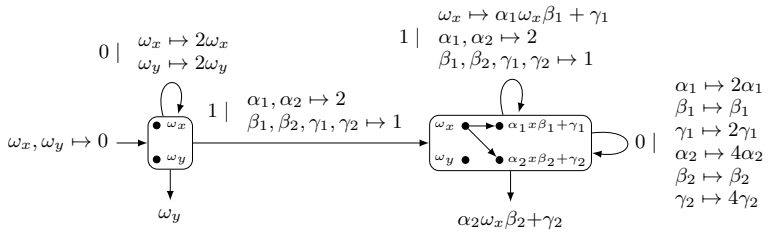



**Figure 3** Finite CRA obtained by merging copyless layers of the CRA in Figure 2.

In particular, both CRAs are equivalent by construction since

$$\begin{aligned}[\![\mathcal{A}]\!](w) &= [\![\nu(\varphi(q))]\!] \\ &= \left[\!\!\left[ [x \mapsto \nu' \circ s^{\ell_{max}+1}(x, \ell_{max}) \ \forall x \in \mathcal{X}] \ (\varphi(q)) \right]\!\!\right] = [\![\nu'(\varphi'(q,s))]\!] = [\![\mathcal{A}']\!](w)\end{aligned}$$

The goal is now to make $\mathcal{A}_\infty$ finite by contracting the flow graph associated with its states. The main operation is the merging of a layer with the previous one; this merge will require some copies of registers. Starting from the finite $\diamond$-less (but not necessarily bounded-copy) CRA $\mathcal{A}$, the merge operation will turn $\mathcal{A}_\infty$ into a finite CRA, that is moreover bounded-copy.

**Merge of two consecutive layers.** Let $s$ be the shape substitution in a state of $\mathcal{A}_\infty$ with maximal layer $\ell_{max}$ and let $L$ be some layer. We now explain how to merge the layers $L-1$ and $L$ of $s$, leading to a new shape substitution $s'$ with one layer less. This will require an update $\sigma'$ of the coefficients appearing in the corresponding shapes. $s'$ and $\sigma'$ are defined as follows for all layers $\ell \in \{0, 1, \ldots, \ell_{max}\}$:

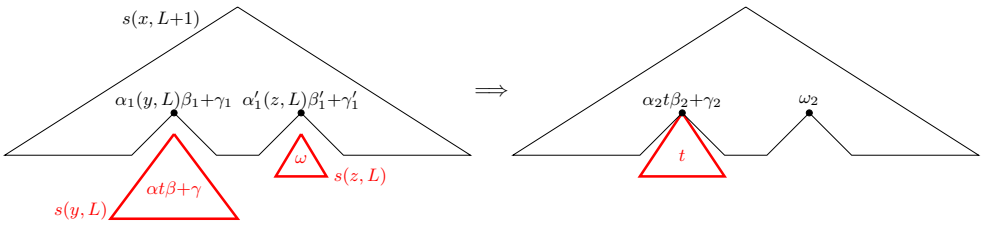
1. if $\ell < L-1$, nothing is changed: for all $x \in \mathcal{X}$, $s'(x,\ell) = s(x,\ell)$ and all the corresponding coefficients $\kappa$ are left unchanged, i.e. $\sigma'(\kappa) = \kappa$;
2. if $\ell \geq L$, we simply shift down all the layers by one: for all $x \in \mathcal{X}$, $s'(x,\ell) = s(x,\ell+1)$ and the corresponding coefficients $\kappa$ are left unchanged too;
3. if $\ell = L-1$, we must incorporate the shapes in-between layers $L-1$ and $L$ into layer $L-1$:
   - if $s(x,L)$ is a constant shape $\omega$, then we simply shift it as before: $s'(x,L-1) = \omega$ and coefficient $\omega$ is left unchanged by $\sigma'$;
   - otherwise, consider the term $t = s^2(x,L)$ obtained by replacing all variables $y$ appearing in the shape $s(x,L)$ by the shape $s(y,L-1)$. Unfortunately, $t$ may not be a shape anymore, but Proposition 12 allows us to recover a new shape $\tau$ from $t$ with new coefficients whose values are given by $\chi$. We thus have to store $s'(x,L-1) = \tau$ in the state of $\mathcal{A}_\infty$ and update the coefficients accordingly, so that $\sigma'(s'(x,L-1)) = \chi(\tau) \equiv t$.

The most promising merge to perform in the infinite CRA $\mathcal{A}_\infty$ in a state $(q,s)$ is the merge of a *copyless* layer $L$ (such that the substitution $[x \mapsto s(x,L)]$ is copyless) with layer $L-1$.

▶ **Example 15.** Consider the infinite CRA $\mathcal{A}_\infty$ built in Figure 2. Three of its depicted states contain copyless layers: notice in particular that, in the state above right, it becomes possible to merge the first two layers after removing the useless (dotted) edge in the flow graph. After these merges, we obtain the finite copyless (and thus bounded-copy) CRA of Figure 3.

The definition of the new transition function (and coefficient updates) given above is correct, but not precise enough to prove afterwards that we obtain a bounded-copy CRA. Therefore, we need to describe an operational method to build $s'$ and $\sigma'$ with as few copies of coefficients as possible. There are two difficulties.

First, when glueing together $s(x, L+1)$ with all the shapes $s(y, L)$ (with $(y, L)$ appearing in $s(x, L+1)$), we need to gather the coefficients at the glueing interface. There are two cases:




Consider first the case where $s(y, L)$ is a (non-constant) shape of the form $\alpha t\beta + \gamma$ with $t$ a term. Let $\alpha_1(y, L)\beta_1 + \gamma_1$ be the subterm of $s(x, L + 1)$ where $(y, L)$ appears. The glueing should replace this subterm by $\alpha_1(\alpha t\beta + \gamma)\beta_1 + \gamma_1 \equiv \alpha_1\alpha t\beta\beta_1 + \alpha_1\gamma\beta_1 + \gamma_1$. This is obtained by replacing it by a shape $\alpha_2 t\beta_2 + \gamma_2$ with $\alpha_2$, $\beta_2$ and $\gamma_2$ fresh coefficients that we set via $\sigma'(\alpha_2) = \alpha_1\alpha$, $\sigma'(\beta_2) = \beta\beta_1$ and $\sigma'(\gamma_2) = \alpha_1\gamma\beta_1 + \gamma_1$. Coefficients of $t$ are preserved by the update $\sigma'$. Notice that $\sigma'$ uses twice the content of coefficients $\alpha_1$ and $\beta_1$.

The other case is the one where $s(y, L)$ is a constant shape $\omega$. Then, we replace in $s(x, L{+}1)$ the subterm $\alpha_1'(y, L)\beta_1' + \gamma_1'$ by a fresh constant coefficient $\omega_2$, set via $\sigma'(\omega_2) = \alpha_1'\omega\beta_1' + \gamma_1'$.

After glueing, in case where certain shapes we glued were constant shapes $\omega$, a final step is required. Either the term does not contain variables of $\mathcal{X}$ anymore (all variables have been replaced by constant shapes $\omega$), and we then replace the whole term $t$ by a constant shape $\omega$ with $\sigma'(\omega) = t$. Or, there are still variables of $\mathcal{X}$, in which case we need to remove all $\omega$-leaves. They are removed one by one, thus modifying the term and the update of coefficients. If there is a subterm of the form $\alpha_1(\omega_2 \odot \omega_3)\beta_1 + \gamma_1$ with two $\omega$-leaves (and $\odot \in \{+, \times\}$), we replace this subterm by a fresh constant coefficient $\omega_4$ set via $\sigma'(\omega_4) = \alpha_1(\omega_2 \odot \omega_3)\beta_1 + \gamma_1$. Once removed all those terms, there might remain isolated $\omega$-leaves: consider thus a subterm with a single $\omega$-leaf, e.g. of the form $\alpha_1\big((\alpha_2 t\beta_2 + \gamma_2) \odot \omega_3\big)\beta_1 + \gamma_1$ with $t$ a shape without any $\omega$. Notice that we can rewrite this term as:

$$\alpha_1\big((\alpha_2 t\beta_2 + \gamma_2) \odot \omega_3\big)\beta_1 + \gamma_1 \equiv \begin{cases} \alpha_1\alpha_2 t\beta_2\beta_1 + \alpha_1(\gamma_2 + \omega_3)\beta_1 + \gamma_1 & \text{if } \odot = + \\ \alpha_1\alpha_2 t\beta_2\omega_3\beta_1 + \alpha_1\gamma_2\omega_3\beta_1 + \gamma_1 & \text{if } \odot = \times \end{cases}$$

Then, this subterm is replaced by a shape $\alpha_4 t\beta_4 + \gamma_4$ with the coefficient updates:

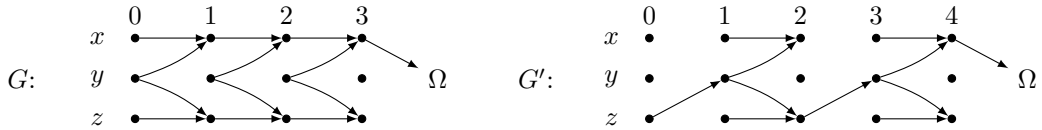
$$\sigma'(\alpha_4) = \alpha_1\alpha_2 \qquad \sigma'(\beta_4) = \begin{cases} \beta_2\beta_1 & \text{if } \odot = + \\ \beta_2\omega_3\beta_1 & \text{if } \odot = \times \end{cases} \qquad \sigma'(\gamma_4) = \alpha_1(\gamma_2 \odot \omega_3)\beta_1 + \gamma_1$$

Notice the copy of $\omega_3$ in the case $\odot = \times$, and the copies of $\alpha_1$ and $\beta_1$ in both cases.

During a whole merge step, we can show that the only coefficients to be copied are the ones of layer $L+1$ and of the constant shapes $\omega$ of layer $L$, and these are copied at most twice.

**Primarily-copyless layer.** The merge of copyless layers with their predecessor may not be sufficient to obtain a finite CRA, as shown in the following more involved example.

▶ **Example 16.** Consider the $\diamond$-less CRA $\mathcal{A}_3$ over the semiring $(\mathfrak{P}(\{a, b\}), \cup, \cdot, \emptyset, \{\varepsilon\})$ and alphabet $\{a, b\}$, with one state $q$ and three registers $x, y, z$, that outputs $\varphi(q) = x$ and with transitions $q \xrightarrow{a|\sigma_a} q$ and $q \xrightarrow{a|\sigma_b} q$ using updates $\sigma_a = [x \mapsto xy, y \mapsto a, z \mapsto zya]$ and $\sigma_b = [x \mapsto \varepsilon, y \mapsto zb, z \mapsto \varepsilon]$. Here are the flow graphs $G$ and $G'$ obtained after reading the input words $aaa$ and $baba$ respectively:




The flow graph $G'$ illustrates that $\mathcal{A}_3$ is not bounded-copy, since the flow graphs on input words $(ba)^n$ require a chain of $n$ copies. In $G$ though, no layer is copyless, and thus no merging based on copyless layers can be performed. Notice that the value of register $z$ is not used in the output, but it would be so if the word $aaa$ is extended with $b$: thus, we cannot simply remove register $z$ to recover some copyless layers. Here, the idea is rather to notice that the second time we see register $y$ being copied (in-between layers 1 and 2), we learn the fact that indeed the previous content of register $y$ is not copied many times (because of the $\diamond$-less hypothesis): if we do not consider this copy anymore, layer 2 becomes copyless and can thus safely be merged with layer 1. It will cost copies, but not too much, as we will see later.

Hence, removing copyless layers is not a sufficient criterion to obtain a *finite* (bounded-copy) CRA from $\mathcal{A}_\infty$. To define the correct criterion, we distinguish some special vertices in a flow graph. A *source* is a vertex $(x, \ell)$ without predecessors (in the state of $\mathcal{A}_\infty$, this means that $s(x, \ell) = \omega$). The sources of a vertex are all its ancestors that are sources: in the previous flow graph $G$ of Example 16, sources of $(x, 2)$ are $(x, 0)$, $(y, 0)$ and $(y, 1)$. A source is *primary* if it is a source, with lowest layer, of a vertex in the maximal layer: $(x, 0)$, $(y, 0)$, $(z, 0)$ and $(y, 3)$ are all the primary sources of $G$ (note that $(y, 3)$ is source with lowest layer of $(y, 3)$ itself), while $(y, 1)$ and $(y, 2)$ are not. A vertex is *primary* if it is a descendant of a primary source. All other vertices are called *secondary*: in $G$, $(y, 1)$ and $(y, 2)$ are all the secondary vertices. In particular, all vertices of the maximal layer are primary. On the minimal layer, all vertices that can reach the maximal layer are primary sources. Notice that whenever the flow graph is extended, some primary sources may turn secondary, and new primary sources may appear, but only on the maximal layer; also, vertices may not reach the maximal layer anymore in which case they are removed. A layer $L$ is *primarily-copyless* when only secondary vertices of layer $L - 1$ are possibly copied, i.e. all primary vertices $(x, L - 1)$ are linked to a single vertex of layer $L$. In $G$, only layers 2 and 3 are primarily-copyless.

Obviously, we do not build the infinite CRA $\mathcal{A}_\infty$, but instead build a finite CRA $\mathcal{A}'$ step by step by extending it along transitions and directly merging each primarily-copyless layers with the previous one, as much as possible. The idea is that merging such primarily-copyless layers will cost some copies of coefficients, but not too many, yielding the fact that $\mathcal{A}'$ is a bounded-copy CRA with a finite number of states equivalent to the $\diamond$-less CRA $\mathcal{A}$. These properties are proved in the rest of the section, and imply Proposition 11.

**$\mathcal{A}'$ is equivalent to $\mathcal{A}$.** By Invariant 14, the infinite CRA (without merge of primarily-copyless layers) is equivalent to $\mathcal{A}$. Then, we prove that it remains true when we perform the merge of two consecutive layers. It is a consequence of the following invariant:

▶ **Invariant 17.** *Let $s$ be a shape substitution with $\ell_{max} + 1$ layers, and let $s'$ and $\sigma'$ be the shape substitution and update obtained by merging layers $L$ and $L - 1$ from $s$. Then $\sigma' \circ {s'}^{\ell_{max}}(x, \ell_{max} - 1) \equiv s^{\ell_{max}+1}(x, \ell_{max})$.*

**$\mathcal{A}'$ is finite.** Given a state $(q, s) \in Q'$, using Lemma 18 (below), $s$ has less than $|\mathcal{X}|^4$ vertices. The substitution is exactly defined by the shapes $s(x, \ell)$ for every vertices $(x, \ell)$. In each of these shapes, each register in $\mathcal{X}$ appears at most once and there are $|\mathcal{X}|^{\mathcal{O}(|\mathcal{X}|)}$ possible of such shapes. Thus, $|Q'| \leq |Q| \cdot |\mathcal{X}|^{\mathcal{O}(|\mathcal{X}|^5)}$ and $\mathcal{A}'$ is finite.

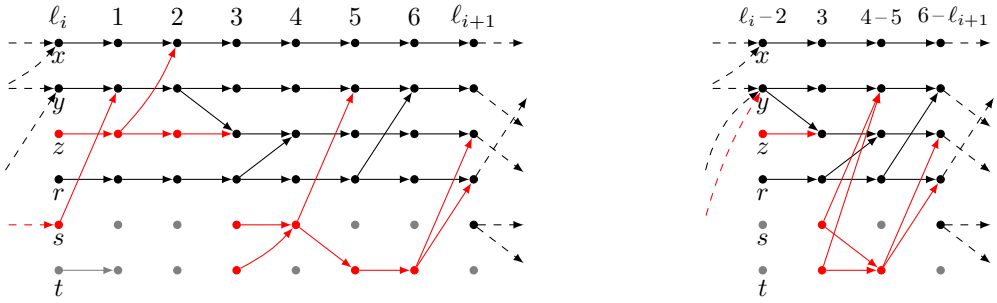



**Figure 4** Merging of primarily copyless layers with the previous one.

▶ **Lemma 18.** *For all states $(q, s)$ in $\mathcal{A}'$, the flow graph associated with $s$ has less than $|\mathcal{X}|^2$ primary sources, $|\mathcal{X}|^3$ layers, and $|\mathcal{X}|^4$ vertices.*

**Proof.** Recall that there are $|\mathcal{X}|$ vertices per layer. In the flow graph, each vertex of the maximal layer defines exactly one minimal layer for its ancestors. All primary sources are on those layers and there are at most $|\mathcal{X}|$ such layers. Each layer contains at most $|\mathcal{X}|$ vertices, thus there are at most $|\mathcal{X}|^2$ primary sources in the flow graph. Also, this implies that the bound on the number of vertices is a direct consequence of the bound on the number of layers. We note $\ell_1, \dots, \ell_k$ the layers where primary sources are, with $k \leq |\mathcal{X}|$ and $0 = \ell_1 < \ell_2 < \cdots < \ell_k$ (the first layer contains only primary sources). For $1 \leq i < k$, we now bound the number of layers separating $\ell_i$ from $\ell_{i+1}$.

Recall that a copy vertex is a vertex with at least two out-going edges. A *primary copy vertex* is a copy vertex that is primary. A *bad layer* is a layer that is not primarily-copyless layer. Bad layers are exactly layers that remain after the removal of all primarily-copyless layers. Note that by definition, a bad layer contains at least one primary copy vertex. Also, as every primary vertex only reaches primary vertices and as the flow graph is $\diamond$-less, a primary copy vertex reaches primary vertices of the next layer that pairwise cannot flow in the same vertex while they all flow in the maximal layer.

Figure 4 depicts vertices of a flow graph between two layers $\ell_i$ and $\ell_{i+1}$ before removing primarily copyless layers on the left and after on the right. Primary vertices are represented in black, secondary vertices are represented in red, and vertices that cannot reach the last layer are represented in gray. Vertices $(r, \ell_i)$ and $(s, \ell_{i+1})$ are primary sources. Between $\ell_i$ and $\ell_{i+1}$, there are 3 primary copy vertices: $(y, 2)$, $(r, 3)$ and $(r, 5)$. Layers $\ell_i$, 1 and 2 are merged, as well as layers 4 and 5, and also layers 6 and $\ell_{i+1}$.

As the flow graph is $\diamond$-less, every primary vertex $(x, \ell_i)$ flows at most once in every primary vertex of layer $\ell_{i+1}$. We note $n_{x,i}$ the number of vertices of layer $\ell_{i+1}$ reachable from $(x, \ell_i)$, we have that $n_{x,i} \leq |\mathcal{X}|$. Then, $(x, \ell_i)$ reaches at most $n_{x,i} - 1$ copy vertices between layers $\ell_i$ (included) and $\ell_{i+1}$ (excluded), all of which are primary. Since this holds for all primary vertices of layer $\ell_i$, there are at most $|\mathcal{X}|\,(|\mathcal{X}| - 1)$ primary copy vertices between layers $\ell_i$ and $\ell_{i+1}$. This directly implies the same bound for the number of (bad) layers. The argument still holds for (bad) layers between $\ell_k$ and the maximal layer $\ell_{max}$ of the flow graph. As a consequence, there are at most $|\mathcal{X}|^2\,(|\mathcal{X}| - 1) \leq |\mathcal{X}|^3$ layers in $s$. ◀

**$\mathcal{A}'$ is bounded-copy.** We need some additional intuition on how secondary vertices flow. Consider a secondary vertex $v$ that flows into vertices of the last layer, that are primary. Along the path from $v$ to the last layer, there are secondary vertices, followed by primary vertices. The first primary vertex encountered is called a *target* of $v$: in the flow graph $G$ of

Example 16, targets of $(y, 1)$ are $(x, 2)$ and $(z, 2)$. Notice that every target is a descendant of some primary source, by definition of primary vertices. A primary vertex is its own target. *Primary sources* of a vertex are all the sources of its targets, that are primary: again in the example $G$, primary sources of $(y, 2)$ are $(x, 0)$, $(y, 0)$ and $(z, 0)$.

We define a measure on the coefficients of all the shapes in a state of $\mathcal{A}'$ that bounds the number of times it can copy into other coefficients in the future. The measure of a coefficient $\kappa$ appearing in the shape of a vertex $v = (x, \ell)$ of the shape substitution is defined as the tuple $\|\kappa\| = (\ell, p, c, \ell_\pi, g)$ where $p$ is the number of primary sources in $s$ restricted to the previous layers $\{0, \dots, \ell - 1\}$; $c = 1$ if the coefficient $\kappa$ is an $\omega$-coefficient, and $c = 0$ otherwise; $\ell_\pi$ is the maximal layer of a primary source of vertex $v$; $g$ is the number of targets of vertex $v$. We can show that along the update of every transition of $\mathcal{A}'$, if a coefficient $\kappa$ is used to update the value of another coefficient $\kappa'$ then $\|\kappa\| \geq \|\kappa'\|$ (where tuples are ordered lexicographically). Moreover, if $\kappa$ is used in at least two coefficients, then $\|\kappa\| > \|\kappa'\|$. We can bound the length of all decreasing sequences of tuples by $2|\mathcal{X}|^9$. When a coefficient is copied, it flows into at most $2|\mathcal{X}|$ coefficients. Thus, $\mathcal{A}'$ is $(2|\mathcal{X}|)^{2|\mathcal{X}|^9}$-copy.

## 5 From bounded-copy CRAs to copyless CRAs

The final step of our construction is to remove all copies from a bounded-copy CRA. This is achieved in a purely greedy manner: it suffices to have enough replicas of each register from the beginning, and then split the replicas evenly when the bounded-copy CRA performs copies.

▶ **Proposition 19.** *For every $k$-copy* CRA*, we can construct an equivalent copyless* CRA*.*

▶ Remark 20. Note that our definition of bounded-copy differs from the one of [4]. There, flow graphs are first trimmed with respect to the output vertex, and $k$-copy means that there are at most $k$ copy vertices in the whole trimmed flow graph. In this sense, 0-copy implies $\diamond$-less in our setting. Therefore, removing copies is the core of their transformation from functional copyless (non-deterministic) streaming string transducers into copyless deterministic ones.

This ends the proof of Theorem 1. Actually, this last step in the proof extends easily to show that bounded-copy finitely-ambiguous CRAs can be effectively transformed into equivalent finitely-ambiguous CRAs. Applying then Theorem 1, we obtain

▶ **Corollary 21.** *Every bounded-copy finitely-ambiguous* NCRA *can be effectively transformed into an equivalent copyless* CRA*.*

## 6 Conclusion

We have shown a construction removing the finite-ambiguity of a copyless NCRA, with an application to the removal of regular look-aheads in copyless CRA-RLAs in arbitrary semirings. It can be shown that the result does no longer hold for linear-ambiguous copyless NCRAs using the example of [14, Theorem 2], therefore finite-ambiguity seems the weakest condition for which our result holds. Moreover, existing techniques of regular look-ahead removal for streaming string-to-tree transducers [3] cannot be used directly, as the addition of unary mappings update strictly increases the expressive power of copyless CRAs. Going back to the example of Figure 1, notice that the application of our construction from the finitely-ambiguous NCRA $\mathcal{A}_n$ yields a copyless CRA that is bigger than the alternative solution $\mathcal{A}_c$: it has more states, and more registers. As future works, we thus plan to study minimisation of copyless CRAs in the general setting of semirings. Our results work even for non-commutative semirings. However it heavily relies on the distributivity property of

semirings, in order to normalise the terms into shapes. The transformation from a $\diamond$-less CRA to a bounded-copy CRA also relies on the capability to multiply two registers (coefficients) together: this is thus unclear how to extend our approach to the so-called *additive* copyless CRAs where products in register updates must happen between a register and a constant.

## References


1 Shaull Almagor, Michaël Cadilhac, Filip Mazowiecki, and Guillermo A. Pérez. Weak Cost Register Automata Are Still Powerful. In *Proceedings of the 22nd International Conference on Developments in Language Theory (DLT 2018)*, volume 11088 of *LNCS*, pages 83–95. Springer, 2018. doi:10.1007/978-3-319-98654-8_7.

2 Rajeev Alur and Loris D'antoni. Streaming Tree Transducers. *Journal of the ACM*, 64(5):1–55, August 2017. doi:10.1145/3092842.

3 Rajeev Alur, Loris D'Antoni, Jyotirmoy V. Deshmukh, Mukund Raghothaman, and Yifei Yuan. Regular Functions and Cost Register Automata. In *Proceedings of the 28th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2013)*, pages 13–22. IEEE Computer Society, 2013. doi:10.1109/LICS.2013.65.

4 Rajeev Alur, Emmanuel Filiot, and Ashutosh Trivedi. Regular Transformations of Infinite Strings. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science (LICS 2012)*, pages 65–74. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.18.

5 Rajeev Alur and Mukund Raghothaman. Decision Problems for Additive Regular Functions. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming, Part II (ICALP 2013)*, volume 7966 of *LNCS*, pages 37–48. Springer, 2013. doi:10.1007/978-3-642-39212-2_7.

6 Marcella Anselmo. Two-Way Automata with Multiplicity. In *Proceedings of the 17th International Colloquium on Automata, Languages and Programming (ICALP 1990)*, volume 443 of *LNCS*, pages 88–102. Springer, 1990.

7 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Alternating Weighted Automata. In *Proceedings of the 17th International Conference on Fundamentals of computation theory (FCT'09)*, volume 5699 of *Lecture Notes in Computer Science*, pages 3–13, 2009.

8 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM Transactions on Computational Logic*, 11(4), 2010. doi:10.1145/1805950.1805953.

9 Laure Daviaud, Pierre-Alain Reynier, and Jean-Marc Talbot. A Generalised Twinning Property for Minimisation of Cost Register Automata. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '16)*, pages 857–866. ACM, 2016. doi:10.1145/2933575.2934549.

10 Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata*. Springer, 2009.

11 Kosaburo Hashiguchi, Kenichi Ishiguro, and Shuji Jimbo. Decidability of The Equivalence Problem for Finitely Ambiguous Finance Automata. *International Journal of Algebra and Computation*, 12(3):445, 2002. doi:10.1142/S0218196702000845.

12 Peter Kostolányi and Filip Misún. Alternating weighted automata over commutative semirings. *Theoretical Computer Science*, 740:1–27, 2018. doi:10.1016/j.tcs.2018.05.003.

13 Filip Mazowiecki and Cristian Riveros. Maximal Partition Logic: Towards a Logical Characterization of Copyless Cost Register Automata. In *24th EACSL Annual Conference on Computer Science Logic (CSL 2015)*, volume 41 of *LIPIcs*, pages 144–159. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.CSL.2015.144.

14 Filip Mazowiecki and Cristian Riveros. Copyless cost-register automata: Structure, expressiveness, and closure properties. *Journal of Computer and System Sciences*, 100:1–29, March 2019. doi:10.1016/j.jcss.2018.07.002.

**15** Mehryar Mohri, Fernando C. N. Pereira, and Michael Riley. The Design Principles of a Weighted Finite-State Transducer Library. *Theoretical Computer Science*, 231(1):17–32, 2000. doi:10.1016/S0304-3975(99)00014-6.

**16** Azaria Paz. Some Aspects of Probabilistic Automata. *Information and Computation*, 9(1):26–60, 1966.

**17** Michael O. Rabin. Probabilistic Automata. *Information and Control*, 6(3):230–245, 1963.

**18** Marcel Paul Schützenberger. On the Definition of a Family of Automata. *Information and Control*, 4(2-3):245–270, 1961. doi:10.1016/S0019-9958(61)80020-X.