Rewriting Higher-Order Stack Trees

Vincent Penelle

Rewriting Higher-Order Stack Trees *

Vincent Penelle

Université Paris-Est, LIGM (CNRS UMR 8049), UPEM, CNRS,
F-77454 Marne-la-Vallée, France
vincent.penelle@u-pem.fr

Abstract. Higher-order pushdown systems and ground tree rewriting systems can be seen as extensions of suffix word rewriting systems. Both classes generate infinite graphs with interesting logical properties. Indeed, the model-checking problem for monadic second order logic (respectively first order logic with a reachability predicate) is decidable on such graphs. We unify both models by introducing the notion of stack trees, trees whose nodes are labelled by higher-order stacks, and define the corresponding class of higher-order ground tree rewriting systems. We show that these graphs retain the decidability properties of ground tree rewriting graphs while generalising the pushdown hierarchy of graphs.

1 Introduction

Since Rabin's proof of the decidability of monadic second order logic (MSO) over the full infinite binary tree Δ_2 [14], there has been an effort to characterise increasingly general classes of structures with decidable MSO theories. This can be achieved for instance using families of graph transformations which preserve the decidability of MSO - such as the unfolding or the MSO-interpretation and applying them to graphs of known decidable MSO theories, such as finite graphs or the graph Δ_2.

This approach was followed in [8], where it is shown that the prefix (or suffix) rewriting graphs of recognisable word rewriting systems, which coincide (up to graph isomorphism) with the transition graphs of pushdown automata (contracting ε-transitions), can be obtained from Δ_2 using inverse regular substitutions, a simple class of MSO-compatible transformations. They also coincide with those obtained by applying MSO interpretations to Δ_2 [1]. Alternately unfolding and applying inverse regular mappings to these graphs yields a strict hierarchy of classes of trees and graphs with a decidable MSO theory [9, 7] coinciding with the transition graphs of *higher-order pushdown automata* and capturing the solutions of *safe higher-order program schemes*[1], whose MSO decidability had already been established in [12]. We will henceforth call this the *pushdown hierarchy* and the graphs at its n-th level *n-pushdown graphs* for simplicity.

* This work was partially supported by the French National Research Agency (ANR), through excellence program BÃĺzout (ANR-10-LABX-58)

[1] This hierarchy was extended to encompass *unsafe* schemes and *collapsible* automata, which are out of the scope of this paper. See [4, 6, 3] for recent results on the topic.

Also well-known are the automatic and tree-automatic structures (see for instance [2]), whose vertices are represented by words or trees and whose edges are characterised using finite automata running over tuples of vertices. The decidability of first-order logic (FO) over these graphs stems from the well-known closure properties of regular word and tree languages, but it can also be related to Rabin's result since tree-automatic graphs are precisely the class of graphs obtained from Δ_2 using *finite-set interpretations* [10], a generalisation of WMSO interpretations mapping structures with a decidable MSO theory to structures with a decidable FO theory. Applying finite-set interpretations to the whole pushdown hierarchy therefore yields an infinite hierarchy of graphs of decidable FO theory, which is proven in [10] to be strict.

Since prefix-recognisable graphs can be seen as word rewriting graphs, another variation is to consider similar rewriting systems over trees. This yields the class of *ground tree rewriting graphs*, which strictly contains that of real-time order 1 pushdown graphs. This class is orthogonal to the whole pushdown hierarchy since it contains at least one graph of undecidable MSO theory, for instance the infinite 2-dimensional grid. The transitive closures of ground tree rewriting systems can be represented using *ground tree transducers*, whose graphs were shown in [11] to have decidable $\mathrm{FO}[\xrightarrow{*}]$ theories by establishing their closure under iteration and then showing that any such graph is tree-automatic.

The purpose of this work is to propose a common extension to both higher-order stack operations and ground tree rewriting. We introduce a model of *higher-order ground tree rewriting* over trees labelled by higher-order stacks (henceforth called *stack trees*), which coincides, at order 1, with ordinary ground tree rewriting and, over unary trees, with the dynamics of higher-order pushdown automata. Following ideas from the works cited above, as well as the notion of recognisable sets and relations over higher-order stacks defined in [5], we introduce the class of *ground (order n) stack tree rewriting systems*, whose derivation relations are captured by *ground stack tree transducers*. Establishing that this class of relations is closed under iteration and can be finite-set interpreted in n-pushdown graphs yields the decidability of their $\mathrm{FO}[\xrightarrow{*}]$ theories.

The remainder of this paper is organised as follows. Section 2 recalls some of the concepts used in the paper. Section 3 defines stack trees and stack tree rewriting systems. Section 4 explores a notion of recognisability for binary relations over stack trees. Section 5 proves the decidability of $\mathrm{FO}[\xrightarrow{*}]$ model checking over ground stack tree rewriting graphs. Finally, Section 6 presents some further perspectives.

2 Definitions and notations

Trees. Given an arbitrary set Σ, an ordered Σ-labelled tree t of arity at most $d \in \mathbb{N}$ is a *partial function* from $\{1, \ldots, d\}^*$ to Σ such that the domain of t, $\mathrm{dom}(t)$ is prefix-closed (if u is in $\mathrm{dom}(t)$, then every prefix of u is also in $\mathrm{dom}(t)$) and left-closed (for all $u \in \{1, \ldots, d\}^*$ and $2 \leq j \leq d$, $t(uj)$ is defined only if $t(ui)$ is for every $i < j$). Node uj is called the j-th *child* of its *parent* node u. Additionally, the nodes of t are totally ordered by the natural length-lexicographic ordering $\leq_{\text{llex}}$ over $\{1,\ldots,d\}^*$. By abuse of notation, given a symbol $a \in \Sigma$, we simply denote by a the tree $\{\epsilon \mapsto a\}$ reduced to a unique a-labelled node. The frontier of t is the set $\mathrm{fr}(t) = \{u \in \mathrm{dom}(t) \mid u1 \notin \mathrm{dom}(t)\}$. Trees will always be drawn in such a way that the left-to-right placement of leaves respects $\leq_{\text{lex}}$. The set of trees labelled by Σ is denoted by $\mathcal{T}(\Sigma)$. In this paper we only consider finite trees, i.e. trees with finite domains.

Given nodes u and v, we write $u \sqsubseteq v$ if u is a prefix of v, i.e. if there exists $w \in \{1,\cdots,d\}^*$, $v = uw$. We will say that u is an *ancestor* of v or is *above* v, and symmetrically that v is *below* u or is its *descendant*. We call $v_{\leq i}$ the prefix of v of length i. For any $u \in \mathrm{dom}(t)$, $t(u)$ is called the *label* of node u in t and $t_u = \{v \mapsto t(uv) \mid uv \in \mathrm{dom}(t)\}$ is the sub-tree of t rooted at u. For any $u \in \mathrm{dom}(t)$, we call $\#_t(u)$ the *arity* of u, i.e. its number of children. When t is understood, we simply write $\#(u)$. Given trees $t, s_1, \ldots, s_k$ and a k-tuple of positions $\mathbf{u} = (u_1, \ldots, u_k) \in \mathrm{dom}(t)^k$, we denote by $t[s_1, \ldots s_k]_{\mathbf{u}}$ the tree obtained by replacing the sub-tree at each position u_i in t by s_i, i.e. the tree in which any node v not below any u_i is labelled $t(v)$, and any node $u_i.v$ with $v \in \mathrm{dom}(s_i)$ is labelled $s_i(v)$. In the special case where t is a k-*context*, i.e. contains leaves $u_1, \ldots, u_k$ labelled by special symbol $\diamond$, we omit $\mathbf{u}$ and simply write $t[s_1, \ldots, s_k] = t[s_1, \ldots, s_k]_{\mathbf{u}}$.

Directed Graphs. A *directed graph* G with edge labels in Γ is a pair (V_G, E_G) where V_G is a set of vertices and $E_G \subseteq (V_G \times \Gamma \times V_G)$ is a set of edges. Given two vertices x and y, we write $x \xrightarrow{\gamma}_G y$ if $(x, \gamma, y) \in E_G$, $x \rightarrow_G y$ if there exists $\gamma \in \Gamma$ such that $x \xrightarrow{\gamma}_G y$, and $x \xrightarrow{\Gamma'}_G y$ if there exists $\gamma \in \Gamma'$ such that $x \xrightarrow{\gamma}_G y$. There is a *directed path* in G from x to y labelled by $w = w_1 \ldots w_k \in \Gamma^*$, written $x \xrightarrow{w}_G y$, if there are vertices $x_0, \ldots, x_k$ such that $x = x_0$, $x_k = y$ and for all $1 \leq i \leq k$, $x_{i-1} \xrightarrow{w_i}_G x_i$. We additionally write $x \xrightarrow{*}_G y$ if there exists w such that $x \xrightarrow{w}_G y$, and $x \xrightarrow{+}_G y$ if there is such a path with $|w| \geq 1$. A directed graph G is *connected* if there exists an *undirected* path between any two vertices x and y, meaning that $(x, y) \in (\rightarrow_G \cup \rightarrow_G^{-1})^*$. We omit G from all these notations when it is clear from the context. A directed graph D is *acyclic*, or is a DAG, if there is no x such that $x \xrightarrow{+} x$. The *empty DAG* consisting of a single vertex (and no edge, hence its name) is denoted by $\square$. Given a DAG D, we denote by I_D its set of vertices of in-degree 0, called *input vertices*, and by O_D its set of vertices of out-degree 0, called *output vertices*. The DAG is said to be of *in-degree* $|I_D|$ and of *out-degree* $|O_D|$. We henceforth only consider finite DAGs.

Rewriting Systems. Let Σ and Γ be finite alphabets. A Γ-labelled *ground tree rewriting system* (GTRS) is a finite set R of triples (ℓ, a, r) called *rewrite rules*, with ℓ and r finite Σ-labelled trees and $a \in \Gamma$ a label. The rewriting graph of R is $\mathcal{G}_R = (V, E)$, where $V = \mathcal{T}(\Sigma)$ and $E = \{(c[\ell], a, c[r]) \mid (\ell, a, r) \in R\}$. The *rewriting relation* associated to R is $\rightarrow_R = \rightarrow_{\mathcal{G}_R}$, its *derivation relation* is $\xrightarrow{*}_R = \xrightarrow{*}_{\mathcal{G}_R}$. When restricted to words (or equivalently unary trees), such systems are usually called *suffix* (or *prefix*) *word rewriting systems*.

$$\begin{tikzcd}[column sep=small]
 & & {[[aa]_1[bab]_1]_2} \arrow[dl] \arrow[drr] & & & \\
 & {[[aa]_1[aaa]_1]_2} \arrow[dl] & & & {[[aa]_1[a]_1[b]_1]_2} \arrow[dl] \arrow[dr] & \\
{[[ab]_1]_2} & & & {[[ba]_1[ba]_1[b]_1]_2} & & {[[abb]_1[ab]_1]_2}
\end{tikzcd}$$

Fig. 1: A 3-stack-tree.

3 Higher-Order Stack Trees

3.1 Higher-Order Stacks

We briefly recall the notion of higher-order stacks (for details, see for instance [5]). In order to obtain a more straightforward extension from stacks to stack trees, we use a slightly tuned yet equivalent definition, whereby the hierarchy starts at level 0 and uses axs different set of basic operations.

In the remainder, Σ will denote a fixed finite alphabet and n a positive integer. We first define stacks of order n (or n-stacks). Let $Stacks_0(\Sigma) = \Sigma$ denote the set of 0-stacks. For $n > 0$, the set of n-stacks is $Stacks_n(\Sigma) = (Stacks_{n-1}(\Sigma))^+$, the set of non-empty sequences of $(n-1)$-stacks. When Σ is understood, we simply write $Stacks_n$. For $s \in Stacks_n$, we write $s = [s_1, \cdots, s_k]_n$, with $k > 0$ and $n > 0$, for an n-stack of size $|s| = k$ whose topmost $(n-1)$-stack is s_k. For example, $[[[aba]_1]_2[[aba]_1[b]_1[aa]_1]_2]_3$ is a 3-stack of size 2, whose topmost 2-stack $[[aba]_1[b]_1[aa]_1]_2$ contains three 1-stacks, etc.

Basic Stack Operations. Given two letters $a, b \in \Sigma$, we define the partial function $\mathrm{rew_{a,b}} : Stacks_0 \to Stacks_0$ such that $\mathrm{rew_{a,b}}(c) = b$, if $c = a$ and is not defined otherwise. We also consider the identity function $\mathrm{id} : Stacks_0 \to Stacks_0$. For $n \geq 1$, the function $\mathrm{copy}_n : Stacks_n \to Stacks_n$ is defined by $\mathrm{copy}_n(s) = [s_1, \cdots, s_k, s_k]_n$, for every $s = [s_1, \cdots, s_k]_n \in Stacks_n$. As it is injective, we denote by $\overline{\mathrm{copy}}_n$ its inverse (which is a partial function).

Each level ℓ operation θ is extended to any level $n > \ell$ stack $s = [s_1, \cdots, s_k]_n$ by letting $\theta(s) = [s_1, \cdots, s_{k-1}, \theta(s_k)]_n$. The set Ops_n of basic operations of level n is defined as: $Ops_0 = \{\mathrm{rew_{a,b}} \mid a, b \in \Sigma\} \cup \{\mathrm{id}\}$, and for $n \geq 1$, $Ops_n = Ops_{n-1} \cup \{\mathrm{copy}_n, \overline{\mathrm{copy}}_n\}$.

3.2 Stack Trees

We introduce the set $ST_n(\Sigma) = \mathcal{T}(Stacks_{n-1}(\Sigma))$ (or simply ST_n when Σ is understood) of *n-stack-trees*. Observe that an n-stack-tree of degree 1 is isomorphic to an n-stack, and that $ST_1 = \mathcal{T}(\Sigma)$. Figure 1 shows an example of a 3-stack tree. The notion of stack trees therefore subsumes both higher-order stacks and ordinary trees.

Basic Stack Tree Operations. We now extend n-stack operations to stack trees. There are in general several positions where one may perform a given operation on a tree. We thus first define the *localised* application of an operation to a specific position in the tree (given by the index of a leaf in the lexicographic ordering of leaves), and then derive a definition of stack tree operations as binary relations, or equivalently as partial functions from stack trees to sets of stack trees.

Any operation of Ops_{n-1} is extended to ST_n as follows: given $\theta \in Ops_{n-1}$, and an integer $i \leq |\mathrm{fr}(t)|$, $\theta_{(i)}(t) = t[\theta(s)]_{u_i}$ with $s = t(u_i)$, where u_i is the i^{th} leaf of the tree, with respect to the lexicographic order. If θ is not applicable to s, $\theta_i(t)$ is not defined. We define $\theta(t) = \{\theta_{(i)}(t) \mid i \leq |\mathrm{fr}(t)|\}$, i.e. the set of stack trees obtained by applying θ to a leaf of t.

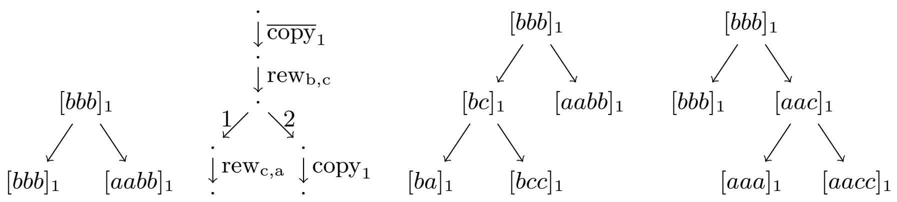
The k-fold duplication of a stack tree leaf and its label is denoted by $\mathrm{copy}_n^k : ST_n \to 2^{ST_n}$. Its application to the i^{th} leaf of a tree t is: $\mathrm{copy}_{n(i)}^k(t) = t \cup \{u_i j \mapsto t(u_i) \mid j \leq k\}$, with $i \leq |\mathrm{fr}(t)|$. Let $\mathrm{copy}_n^k(t) = \{\mathrm{copy}_{n(i)}^k(t)\}$ be the set of stack trees obtained by applying copy_n^k to a leaf of t. The inverse operation, written $\overline{\mathrm{copy}}_n^k$, is such that $t' = \overline{\mathrm{copy}}_{n(i)}^k(t)$ if $t = \mathrm{copy}_{n(i)}^k(t')$. We also define $\overline{\mathrm{copy}}_n^k(t) = \{\overline{\mathrm{copy}}_{n(i)}^k(t)\}$. Notice that $t' \in \overline{\mathrm{copy}}_n^k(t)$ if $t \in \mathrm{copy}_n^k(t')$.

For simplicity, we will henceforth only consider the case where stack trees have arity at most 2 and $k \leq 2$, but all results go through in the general case. We denote by $TOps_n = Ops_{n-1} \cup \{\mathrm{copy}_n^k, \overline{\mathrm{copy}}_n^k \mid k \leq 2\}$ the set of basic operations over ST_n.

3.3 Stack Tree Rewriting

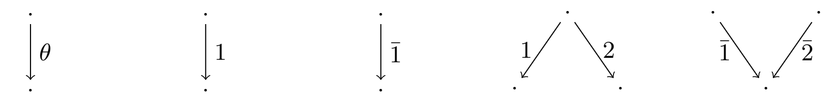
As already mentioned, ST_1 is the set of trees labelled by Σ. In contrast with basic stack tree operations, a tree rewrite rule (ℓ, r) expresses the replacement of an arbitrarily large ground subtree ℓ of some tree $s = c[\ell]$ into r, yielding the tree $c[r]$. Contrary to the case of order 1 stacks (which are simply words), composing basic stack tree operations does not allow us to directly express such an operation, because there is no guarantee that two successive operations will be applied to the same part of a tree. We thus need to find a way to consider compositions of basic operations acting on a single sub-tree. In our notations, the effect of a ground tree rewrite rule could thus be seen as the *localised* application of a sequence of rew and $\overline{\mathrm{copy}}_1^2$ operations followed by a sequence of rew and copy_1^2 operations. The relative positions where these operations must be applied could be represented as a pair of trees with edge labels in Ops_0.

From level 2 on, this is no longer possible. Indeed a localised sequence of operations may be used to perform introspection on the stack labelling a node without destroying it, by first performing a copy_2 operation followed by a sequence of level 1 operations and a $\overline{\mathrm{copy}}_2$ operation. It is thus impossible to directly represent such a transformation using pairs of trees labelled by stack tree operations. We therefore adopt a presentation of *compound operations* as DAGs, which allows us to specify the relative application positions of successive basic operations. However, not every DAG represents a valid compound operation, so



(a) Stack tree t (b) Operation D (c) $D_{(1)}(t)$ (d) $D_{(2)}(t)$

Fig. 2: The application of an operation D to a stack tree t.



(a) D_θ (b) $D_{\text{copy}_n^1}$ (c) $D_{\overline{\text{copy}}_n^1}$ (d) $D_{\text{copy}_n^2}$ (e) $D_{\overline{\text{copy}}_n^2}$

Fig. 3: DAGs of the basic n-stack tree operations (here θ ranges over Ops_{n-1}).

we first need to define a suitable subclass of DAGs and associated concatenation operation. An example of the model we aim to define can be found in Fig. 2.

Concatenation of DAGs. Given two DAGs D and D' with $O_D = \{b_1, \dots, b_\ell\}$ and $I_{D'} = \{a'_1, \dots, a'_{k'}\}$ and two indices i and j with $1 \leq i \leq \ell$ and $1 \leq j \leq k'$, we denote by $D \cdot_{i,j} D'$ the unique DAG D'' obtained by merging the $(i+m)$-th output vertex of D with the $(j+m)$-th input vertex of D' for all $m \geq 0$ such that both b_{i+m} and a'_{j+m} exist. Formally, letting $d = \min(\ell - i, k' - j) + 1$ denote the number of merged vertices, we have $D'' = \text{merge}_f(D \uplus D')$ where $\text{merge}_f(D)$ is the DAG whose set of vertices is $f(V_D)$ and set of edges is $\{(f(x), \gamma, f(x')) \mid (x, \gamma, x') \in E_D\}$, and $f(x) = b_{i+m}$ if $x = a'_{j+m}$ for some $0 \leq m \leq d$, and $f(x) = x$ otherwise. We call D'' the (i,j)-concatenation of D and D'. Note that the (i,j)-concatenation of two connected DAGs remains connected.

Compound Operations We represent compound operations as DAGs. We will refer in particular to the set of DAGs $\mathcal{D}_n = \{D_\theta \mid \theta \in TOps_n\}$ associated with basic operations, which are depicted in Fig. 3. Compound operations are inductively defined below, as depicted in Fig. 4.

Definition 1. *A DAG D is a* compound operation *(or simply an* operation*) if one of the following holds:*

1. $D = \Box$;
2. $D = (D_1 \cdot_{1,1} D_\theta) \cdot_{1,1} D_2$, *with* $|O_{D_1}| = |I_{D_2}| = 1$ *and* $\theta \in Ops_{n-1} \cup \{\text{copy}_n^1, \overline{\text{copy}}_n^1\}$;
3. $D = ((D_1 \cdot_{1,1} D_{\text{copy}_n^2}) \cdot_{2,1} D_3) \cdot_{1,1} D_2$, *with* $|O_{D_1}| = |I_{D_2}| = |I_{D_3}| = 1$;

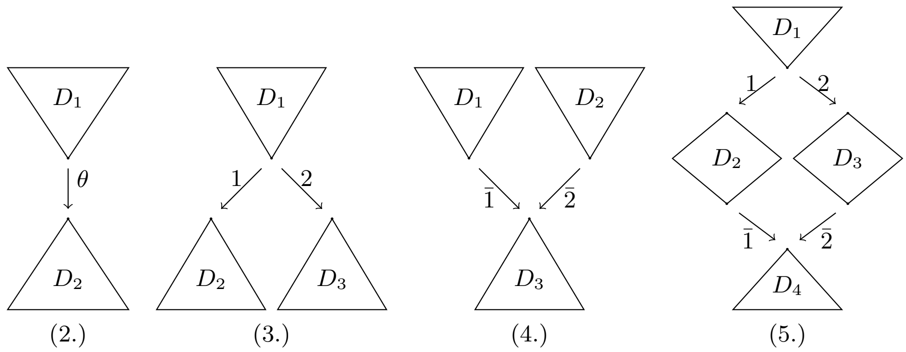


Fig. 4: Possible decompositions of a compound operation, numbered according to the items in Definition 1.

4. *$D = (D_1 \cdot_{1,1} (D_2 \cdot_{1,2} D_{\overline{\mathrm{copy}}^2_n})) \cdot_{1,1} D_3$ with $|O_{D_1}| = |O_{D_2}| = |I_{D_3}| = 1$;*
5. *$D = ((((D_1 \cdot_{1,1} D_{\mathrm{copy}^2_n}) \cdot_{2,1} D_3) \cdot_{1,1} D_2) \cdot_{1,1} D_{\overline{\mathrm{copy}}^2_n}) \cdot_{1,1} D_4$, with $|O_{D_1}| = |I_{D_2}| = |O_{D_2}| = |I_{D_3}| = |O_{D_3}| = |I_{D_4}| = 1$;*

where D_1, D_2, D_3 and D_4 are compound operations.

Additionally, the vertices of D are ordered inductively in such a way that every vertex of D_i in the above definition is smaller than the vertices of D_{i+1}, the order over $\square$ being the empty one. This induces in particular an order over the input vertices of D, and one over its output vertices.

Definition 2. *Given a compound operation D, we define $D_{(i)}(t)$, its* localised application *starting at the i-th leaf of a stack tree t, as follows:*

1. *If $D = \square$, then $D_{(i)}(t) = t$.*
2. *If $D = (D_1 \cdot_{1,1} D_\theta) \cdot_{1,1} D_2$ with $\theta \in Ops_{n-1} \cup \{\mathrm{copy}^1_n, \overline{\mathrm{copy}}^1_n\}$,*
   then $D_{(i)}(t) = D_{2(i)}(\theta_{(i)}(D_{1(i)}(t)))$.
3. *If $D = ((D_1 \cdot_{1,1} D_{\mathrm{copy}^2_n}) \cdot_{2,1} D_3) \cdot_{1,1} D_2$,*
   then $D_{(i)}(t) = D_{2(i)}(D_{3(i+1)}(\mathrm{copy}^2_{n(i)}(D_{1(i)}(t))))$.
4. *If $D = ((D_1 \cdot_{1,1} (D_2 \cdot_{2,1} D_{\overline{\mathrm{copy}}^2_n})) \cdot_{1,1} D_3$,*
   then $D_{(i)}(t) = D_{3(i)}(\overline{\mathrm{copy}}^2_{n(i)}(D_{2(i+1)}(D_{1(i)}(t))))$.
5. *If $D = ((((D_1 \cdot_{1,1} D_{\mathrm{copy}^2_n}) \cdot_{2,1} D_3) \cdot_{1,1} D_2) \cdot_{1,1} D_{\overline{\mathrm{copy}}^2_n}) \cdot_{1,1} D_4$,*
   then $D_{(i)}(t) = D_{4(i)}(\overline{\mathrm{copy}}^2_{n(i)}(D_{3(i+1)}(D_{2(i)}(\mathrm{copy}^2_{n(i)}(D_{1(i)}(t))))))$.

Remark 1. An operation may admit several different decompositions with respect to Def. 1. However, its application is well-defined, as one can show this process is locally confluent.

Given two stack trees t, t' and an operation D, we say that $t' \in D(t)$ if there is a position i such that $t' = D_{(i)}(t)$. Figure 2 shows an example. We call $\mathcal{R}_D$ the relation induced by D: for any stack trees t, t', $\mathcal{R}_D(t, t')$ if and only if $t' \in D(t)$.

Finally, given a k-tuple of operations $\bar{D} = (D_1, \ldots, D_k)$ of respective in-degrees $d_1, \ldots, d_k$ and a k-tuple of indices $\mathbf{i} = (i_1, \ldots, i_k)$ with $i_{j+1} \geq i_j + d_j$ for all $1 \leq j < k$, we denote by $\bar{D}_{(\mathbf{i})}(t)$ the parallel application $D_{1(i_1)}(\ldots D_{k(i_k)}(t) \ldots)$ of $D_1, \ldots, D_k$ to t, $\bar{D}(t)$ the set of all such applications and $\mathcal{R}_{\bar{D}}$ the induced relation.

Since the (i,j)-concatenation of two operations as defined above is not necessarily a licit operation, we need to restrict ourselves to results which are well-formed according to Def. 1. Given D and D', we let $D \cdot D' = \{D \cdot_{i,j} D' \mid D \cdot_{i,j} D' \text{ is an operation}\}$. Given $n > 1$, we define[2] $D^n = \bigcup_{i<n} D^i \cdot D^{n-i}$, and let $D^* = \bigcup_{n \geq 0} D^n$ denote the set of *iterations* of D. These notations are naturally extended to sets of operations.

Proposition 1. *$\mathcal{D}_n^*$ is precisely the set of all well-formed compound operations.*

Proof. Recall that $\mathcal{D}_n$ denotes the set of DAGs associated with basic operations. By definition of iteration, any DAG in $\mathcal{D}_n^*$ is an operation. Conversely, by Def. 1, any operation can be decomposed into a concatenation of DAGs of $\mathcal{D}_n$. $\square$

Ground Stack Tree Rewriting Systems. By analogy with order 1 trees, given some finite alphabet of labels Γ, we call any finite subset of labelled operations in $\mathcal{D}_n^* \times \Gamma$ a labelled *ground stack-tree rewriting system* (GSTRS). We straightforwardly extend the notions of rewriting graph and derivation relation to these systems. Note that for $n = 1$, this class coincides with ordinary ground tree rewriting systems. Moreover, one can easily show that the rewriting graphs of ground stack-tree rewriting systems over unary n-stack trees (trees containing only unary operations, i.e. no edge labelled by 2 or $\bar{2}$) are isomorphic to the configuration graphs of order n pushdown automata performing a finite sequence of operations at each transition.

4 Operation Automata

In this section, in order to provide finite descriptions of possibly infinite sets of operations, in particular the derivation relations of GSTRS, we extend the notion of *ground tree transducers* (or GTT) of [11] to ground tree rewriting systems.

A GTT T is given by a tuple $\big((A_i, B_i)\big)_{1 \leq i \leq k}$ of pairs of finite tree automata. A pair of trees (s,t) is accepted by T if $s = c[s_1, \ldots s_m]$ and $t = c[t_1, \ldots, t_m]$ for some m-context c, where for all $1 \leq j \leq m$, $s_j \in L(A_i)$ and $t_j \in L(B_i)$ for some $1 \leq i \leq k$. It is also shown that, given a relation R recognised by a GTT, there exists another GTT recognising its reflexive and transitive closure R^*.

Directly extending this idea to ground stack tree rewriting systems is not straightforward: contrary to the case of trees, a given compound operation may be applicable to many different subtrees. Indeed, the only subtree to which a ground tree rewriting rule (s,t) can be applied is the tree s. On stack trees,

[2] This unusual definition is necessary because $\cdot$ is not associative. For example, $(D_{\text{copy}_n^2} \cdot_{2,1} D_{\text{copy}_n^2}) \cdot_{1,1} D_{\text{copy}_n^2}$ is in $(D_{\text{copy}_n^2})^2 \cdot D_{\text{copy}_n^2}$ but not in $D_{\text{copy}_n^2} \cdot (D_{\text{copy}_n^2})^2$.

this is no longer true, as depicted in Fig. 2: an operation does not entirely describe the labels of nodes of subtrees it can be applied to (as in the case of trees), and can therefore be applied to infinitely many different subtrees. We will thus express relations by describing sets of compound operations over stack trees. Following [5] where recognisable sets of higher-order stacks are defined, we introduce operation automata and recognisable sets of operations.

Definition 3. *An automaton over $\mathcal{D}_n^*$ is a tuple $A = (Q, \Sigma, I, F, \Delta)$, where*

- *Q is a finite set of states,*
- *Σ is a finite stack alphabet,*
- *$I \subseteq Q$ is a set of initial states,*
- *$F \subseteq Q$ is a set of final states,*
- *$\Delta \subseteq \big(Q \times (Ops_{n-1} \cup \{\mathrm{copy}_n^1, \overline{\mathrm{copy}}_n^1\}) \times Q\big) \cup ((Q \times Q) \times Q) \cup (Q \times (Q \times Q))$ is a set of transitions.*

An operation D is accepted by A if there is a labelling of its vertices by states of Q such that all input vertices are labelled by initial states, all output vertices by final states, and this labelling is consistent with Δ, in the sense that for all x, y and z respectively labelled by states p, q and r, and for all $\theta \in Ops_{n-1} \cup \{\mathrm{copy}_n^1, \overline{\mathrm{copy}}_n^1\}$,

$$\begin{aligned} x \xrightarrow{\theta} y &\implies (p, \theta, q) \in \Delta, \\ x \xrightarrow{1} y \wedge x \xrightarrow{2} z &\implies (p, (q, r)) \in \Delta, \\ x \xrightarrow{\bar{1}} z \wedge y \xrightarrow{\bar{2}} z &\implies ((p, q), r) \in \Delta. \end{aligned}$$

We denote by $\mathrm{Op}(A)$ the set of operations recognised by A. *Rec* denotes the class of sets of operations recognised by operation automata. A pair of stack trees (t, t') is in the relation $\mathcal{R}(A)$ defined by A if for some $k \geq 1$ there is a k-tuple of operations $\bar{D} = (D_1, \cdots, D_k)$ in $\mathrm{Op}(A)^k$ such that $t' \in \bar{D}(t)$. At order 1, we have already seen that stack trees are simply trees, and that ground stack tree rewriting systems coincide with ground tree rewriting systems. Similarly, we also have the following:

Proposition 2. *The classes of relations recognised by order* 1 *operation automataand by ground tree transducers coincide.*

At higher orders, the class *Rec* and the corresponding binary relations retains several of the good closure properties of ground tree transductions.

Proposition 3. *Rec is closed under union, intersection and iterated concatenation. The class of relations defined by operation automata is closed under composition and iterated composition.*

The construction of automata recognising the union and intersection of two recognisable sets, the iterated concatenation of a recognisable set, or the composition of two automata-definable relations, can be found in the appendix. Given automaton A, the relation defined by the automaton accepting $\mathrm{Op}(A)^*$ is $\mathcal{R}(A)^*$.

Normalised automata. Operations may perform "unnecessary" actions on a given stack tree, for instance duplicating a leaf with a copy_n^2 operation and later destroying both copies with $\overline{\mathrm{copy}}_n^2$. Such operations which leave the input tree unchanged are referred to as *loops*. There are thus in general infinitely many operations representing the same relation over stack trees. It is therefore desirable to look for a canonical representative (a canonical operation) for each considered relation. The intuitive idea is to simplify operations by removing occurrences of successive mutually inverse basic operations. This process is a very classical tool in the literature of pushdown automata and related models, and was applied to higher-order stacks in [5]. Our notion of reduced operations is an adaptation of this work.

There are two main hurdles to overcome. First, as already mentioned, a compound operation D can perform introspection on the label of a leaf without destroying it. If D can be applied to a given stack tree t, such a sequence of operations does not change the resulting stack tree s. It does however forbid the application of D to other stack trees by inspecting their node labels, hence removing this part of the computation would lead to an operation with a possibly strictly larger domain. To adress this problem, and following [5], we use *test operations* ranging over regular sets of $(n-1)$-stacks, which will allow us to handle non-destructive node-label introspection.

A second difficulty appears when an operation destroys a subtree and then reconstructs it identically, for instance a $\overline{\mathrm{copy}}_n^2$ operation followed by copy_n^2. Trying to remove such a pattern would lead to a disconnected DAG, which does not describe a compound operation in our sense. We thus need to leave such occurrences intact. We can nevertheless bound the number of times a given position of the input stack tree is affected by the application of an operation by considering two phases: a *destructive* phase during which only $\overline{\mathrm{copy}}_n^i$ and order $n-1$ basic operations (possibly including tests) are performed on the input stack-tree, and a *constructive* phase only consisting of copy_n^i and order $n-1$ basic operations. Similarly to the way ground tree rewriting is performed at order 1.

Formally, a *test* T_L over $Stacks_n$ is the restriction of the identity operation to $L \in Rec(Stacks_n)$[3]. In other words, given $s \in Stacks_n$, $T_L(s) = s$ if $s \in L$, otherwise, it is undefined. We denote by $\mathcal{T}_n$ the set of test operations over $Stacks_n$. We enrich our basic operations over ST_n with $\mathcal{T}_{n-1}$. We also extend compound operations with edges labelled by tests. We denote by $\mathcal{D}_n^{\mathcal{T}}$ the set of basic operations with tests. We can now define the notion of reduced operation analogously to that of reduced instructions with tests in [5].

Definition 4. *For $i \in \{0, \cdots, n\}$, we define the set of words* Red_i *over $Ops_n \cup \mathcal{T}_n \cup \{1, 2, \bar{1}, \bar{2}\}$ as:*

– $\mathrm{Red}_0 = \{\varepsilon, T, \mathrm{rew_{a,b}}, \mathrm{rew_{a,b}} \cdot T, T \cdot \mathrm{rew_{a,b}}, \mathrm{rew_{a,b}} \cdot T \cdot \mathrm{rew_{c,d}} \mid a, b, c, d \in \Sigma, T \in \mathcal{T}_n\}$,

[3] Regular sets of n-stacks are obtained by considering regular sets of sequences of operations of Ops_n applied to a given stack s_0. More details can be found in [5].

– *For* $0 < i < n$, $\mathrm{Red}_i = (\mathrm{Red}_{i-1} \cdot \overline{\mathrm{copy}}_i)^* \cdot \mathrm{Red}_{i-1} \cdot (\mathrm{copy}_i \cdot \mathrm{Red}_{i-1})^*$,
– $\mathrm{Red}_n = (\mathrm{Red}_{n-1} \cdot \{\bar{1}, \bar{2}\})^* \cdot \mathrm{Red}_{n-1} \cdot (\{1, 2\} \cdot \mathrm{Red}_{n-1})^*$.

Definition 5. *An operation with tests D is* reduced *if for every $x, y \in V_D$, if $x \xrightarrow{w} y$, then $w \in \mathrm{Red}_n$.*

Observe that, in the decomposition of a reduced operation D, case 5 of the inductive definition of compound operations (Def. 1) should never occur, as otherwise, there would be a path on which 1 appears before $\bar{1}$, which contradicts the definition of reduced operation.

An automaton A is said to be *normalised* if it only accepts reduced operations, and *distinguished* if there is no transition ending in an initial state or starting in a final state. The following proposition shows that any operation automaton can be normalised and distinguished.

Proposition 4. *For every automaton A, there exists a distinguished normalised automaton with tests A_r such that $\mathcal{R}(A) = \mathcal{R}(A_r)$.*

The idea of the construction is to transform A in several steps, each modifying the set of accepted operations but not the recognised relation. The proof relies on the closure properties of regular sets of $(n-1)$-stacks and an analysis of the structure of A. We show in particular, using a saturation technique, that the set of states of A can be partitioned into *destructive states* (which label the destructive phase of the operation, which does not contain the copy_n^i operation) and the *constructive states* (which label the constructive phase, where no $\overline{\mathrm{copy}}_n^i$ occurs). These sets are further divided into *test states*, which are reached after a test has been performed (and only then) and which are the source of no test-labelled transition, and the others. This transformation can be performed without altering the accepted relation over stack trees.

5 Rewriting Graphs of Stack Trees

In this section, we study the properties of ground stack tree rewriting graphs. Our goal is to show that the graph of any Γ-labelled GSTRS has a decidable $\mathrm{FO}[\xrightarrow{*}]$ theory. We first state that there exists a distinguished and reduced automaton A recognising the derivation relation $\xrightarrow{*}_R$ of R, and then show, following [10], that there exists a finite-set interpretation of $\xrightarrow{*}_R$ and every $\xrightarrow{a}_R$ for $(D, a) \in R$ from a graph with decidable WMSO-theory.

Theorem 1. *Given a Γ-labelled GSTRS R, $\mathcal{G}_R$ has a decidable FO$[\xrightarrow{*}]$ theory.*

To prove this theorem, we show that the graph $\mathcal{H}_R = (V, E)$ with $V = ST_n$ and $E = (\xrightarrow{*}_R) \cup \bigcup_{a \in \Gamma} (\xrightarrow{a}_R)$ obtained by adding the relation $\xrightarrow{*}_R$ to $\mathcal{G}_R$ has a decidable FO theory. To do so, we show that $\mathcal{H}_R$ is finite-set interpretable inside a structure with a decidable WMSO-theory, and conclude using Corollary 2.5 of [10]. Thus from Section 5.2 of the same article, it follows that the rewriting graphs of GSTRS are in the tree-automatic hierarchy.

Given a Γ-labelled GSTRS R over ST_n, we choose to interpret $\mathcal{H}_R$ inside the *order n Treegraph* Δ^n over alphabet $\Sigma \cup \{1,2\}$. Each vertex of this graph is an n-stack, and there is an edge $s \xrightarrow{\theta} s'$ if and only if $s' = \theta(s)$ with $\theta \in Ops_n \cup \mathcal{T}_n$. This graph belongs to the n-th level of the pushdown hierarchy and has a decidable WMSO theory[4].

Given a stack tree t and a position $u \in dom(t)$, we denote by $\mathrm{Code}(t,u)$ the n-stack $[\mathrm{push}_{w_0}(t(\varepsilon)), \mathrm{push}_{w_1}(t(u_{\leq 1})), \cdots, \mathrm{push}_{w_{|u|-1}}(t(u_{\leq |u|-1})), t(u)]_n$, where $\mathrm{push}_w(s)$ is obtained by adding the word w at the top of the top-most 1-stack in s, and $w_i = \#(u_{\leq i})u_{i+1}$. This stack $\mathrm{Code}(t,u)$ is the encoding of the node at position u in t. Informally, it is obtained by storing in an n-stack the sequence of $(n-1)$-stacks labelling nodes from the root of t to position u, and adding at the top of each $(n-1)$-stack the number of children of the corresponding node of t and the next direction taken to reach node u. Any stack tree t is then encoded by the finite set of n-stacks $X_t = \{\mathrm{Code}(t,u) \mid u \in fr(t)\}$, i.e. the set of encodings of its leaves. Observe that this coding is injective.

Example 1. The coding of the stack tree t depicted in Fig. 1 is:

$$\begin{aligned} X_t = \{\ & [[[aa]_1[bab21]_1]_2[[aa]_1[aaa11]_1]_2[[ab]_1]_2]_3, \\ & [[[aa]_1[bab22]_1]_2[[aa]_1[a]_1[b21]_1]_2[[ba]_1[ba]_1[b]_1]_2]_3, \\ & [[[aa]_1[bab22]_1]_2[[aa]_1[a]_1[b22]_1]_2[[abb]_1[ab]_1]_2]_3\} \end{aligned}$$

We now represent any relation S between two stack trees as a WMSO-formula with two free second-order variables, which holds in Δ^n over sets X_s and X_t if and only if $(s,t) \in S$.

Proposition 5. *Given a Γ-labelled GSTRS R, there exist WMSO-formulæ δ, Ψ_a and ϕ such that:*

- $\Delta^n_{\Sigma \cup \{1,2\}} \models \delta(X)$ *if and only if* $\exists t \in ST_n, X = X_t$,
- $\Delta^n_{\Sigma \cup \{1,2\}} \models \Psi_a(X_s, X_t)$ *if and only if* $t \in D(s)$ *for some* $(D,a) \in R$,
- $\Delta^n_{\Sigma \cup \{1,2\}} \models \phi(X_s, X_t)$ *if and only if* $s \xrightarrow{*}_R t$.

First note that the intuitive idea behind this interpretation is to only work on those vertices of Δ^n which are the encoding of some node in a stack-tree. Formula δ will distinguish, amongst all possible finite sets of vertices, those which correspond to the set of encodings of all leaves of a stack-tree. Formulæ Ψ_a and ϕ then respectively check the relationship through $\xrightarrow{a}_R$ (resp. $\xrightarrow{*}_R$) of a pair of stack-trees. We give here a quick sketch of the formulæ and a glimpse of their proof of correction. More details can be found in appendix C.

Let us first detail formula δ, which is of the form

$$\delta(X) = \mathrm{OnlyLeaves}(X) \wedge \mathrm{TreeDom}(X) \wedge \mathrm{UniqueLabel}(X).$$

$\mathrm{OnlyLeaves}(X)$ holds if every element of X codes for a leaf. $\mathrm{TreeDom}(X)$ holds if the induced domain is the domain of a tree and the arity of each node is

[4] It is in fact a generator of this class of graphs via WMSO-interpretations (see [7] for additional details).

consistent with the elements of X. UniqueLabel(X) holds if for every position u in the induced domain, all elements which include u agree on its label.

From here on, variables X and Y will respectively stand for the encoding of some input stack tree s and output stack-tree t. For each $a \in \Gamma$, $\Psi_a(X,Y)$ is the disjunction of a family of formulæ $\Psi_D(X,Y)$ for each $(D,a) \in R$. Each Ψ_D is defined by induction over D, simulating each basic operations in D, ensuring that they are applied according to their respective positions, and to a single closed subtree of s (which simply corresponds to a subset of X), yielding t.

Let us now turn to formula ϕ. Since the set of DAGs in R is finite, it is recognisable by an operation automaton. Since *Rec* is closed under iteration (Cf. Sec. 4), one may build a distinguished normalised automaton accepting $\xrightarrow{*}_R$. What we thus really show is that given such an automaton A, there exists a formula ϕ such that $\phi(X,Y)$ holds if and only if $t \in \bar{D}(s)$ for some vector $\bar{D} = D_1, \dots D_k$ of DAGs accepted by A. Formula ϕ is of the form

$$\phi(X,Y) = \exists \boldsymbol{Z}, \mathrm{Init}(X,Y,\boldsymbol{Z}) \wedge \mathrm{Diff}(\boldsymbol{Z}) \wedge \mathrm{Trans}(\boldsymbol{Z}).$$

Following a common pattern in automata theory, this formula expresses the existence of an accepting run of A over some tuple of reduced DAGs $\bar{D}$, and states that the operation corresponding to $\bar{D}$, when applied to s, yields t. Here, $\boldsymbol{Z} = Z_{q_1}, \cdots, Z_{q_{|Q_A|}}$ defines a labelling of a subset of $\Delta^n_{\Sigma \cup \{1,2\}}$ with the states of the automaton, each element Z_q of $\boldsymbol{Z}$ representing the set of nodes labelled by a given control state q. Sub-formula Init checks that only the elements of X (representing the leaves of s) are labelled by initial states, and only those in Y (leaves of t) are labelled by final states. Trans ensures that the whole labelling respects the transition rules of A. For each component D of $\bar{D}$, and since every basic operation constituting D is applied locally and has an effect on a subtree of height and width at most 2, this amounts to a local consistency check between at most three vertices, encoding two nodes of a stack tree and their parent node. The relative positions where basic operations are applied is checked using the sets in $\boldsymbol{Z}$, which represent the flow of control states at each step of the transformation of s into t. Finally, Diff ensures that no stack is labelled by two states belonging to the same part (destructive, constructive, testing or non-testing) of the automaton, thus making sure we simulate a unique run of A. This is necessary to ensure that no spurious run is generated, and is only possible because A is normalised.

6 Perspectives

There are several open questions arising from this work. The first one is the strictness of the hierarchy, and the question of finding simple examples of graphs separating each of its levels with the corresponding levels of the pushdown and tree-automatic hierarchies. A second interesting question concerns the trace languages of stack tree rewriting graphs. It is known that the trace languages of higher-order pushdown automata are the indexed languages [8], that the class of languages recognised by automatic structures are the context-sensitive languages [15] and that those recognised by tree-automatic structures form the class ETIME [13]. However there is to our knowledge no characterisation of the languages recognised by ground tree rewriting systems. It is not hard to define a 2-stack-tree rewriting graph whose path language between two specific vertices is $\{u \sqcup\!\sqcup u \mid u \in \Sigma^*\}$, which we believe cannot be recognised using tree rewriting systems or higher-order pushdown automata[5]. Finally, the model of stack trees can be readily extended to trees labelled by trees. Future work will include the question of extending our notion of rewriting and Theorem 1 to this model.

References

1. Klaus Barthelmann. When can an equational simple graph be generated by hyperedge replacement? In *MFCS*, volume 1450, pages 543–552. Springer, 1998.
2. Achim Blumensath and Erich Grädel. Finite presentations of infinite structures: Automata and interpretations. *Theory Comput. Syst.*, 37(6):641–674, 2004.
3. Christopher H. Broadbent, Arnaud Carayol, Matthew Hague, and Olivier Serre. A saturation method for collapsible pushdown systems. In *ICALP (2)*, volume 7392 of *LNCS*, pages 165–176. Springer, 2012.
4. Christopher H. Broadbent, Arnaud Carayol, C.-H. Luke Ong, and Olivier Serre. Recursion schemes and logical reflection. In *LICS*, pages 120–129. IEEE Computer Society, 2010.
5. Arnaud Carayol. Regular sets of higher-order pushdown stacks. In *MFCS*, volume 3618 of *LNCS*, pages 168–179. Springer, 2005.
6. Arnaud Carayol and Olivier Serre. Collapsible pushdown automata and labeled recursion schemes: Equivalence, safety and effective selection. In *LICS*, pages 165–174. IEEE, 2012.
7. Arnaud Carayol and Stefan Wöhrle. The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *FSTTCS*, volume 2914 of *LNCS*, pages 112–123. Springer, 2003.
8. Didier Caucal. On infinite transition graphs having a decidable monadic theory. In *ICALP*, volume 1099 of *LNCS*, pages 194–205. Springer, 1996.
9. Didier Caucal. On infinite terms having a decidable monadic theory. In *MFCS*, volume 2420 of *LNCS*, pages 165–176. Springer, 2002.
10. Thomas Colcombet and Christof Löding. Transforming structures by set interpretations. *Logical Methods in Computer Science (LMCS)*, 3(2), 2007.
11. Max Dauchet and Sophie Tison. The theory of ground rewrite systems is decidable. In *LICS*, pages 242–248. IEEE Computer Society, 1990.
12. Teodor Knapik, Damian Niwinski, and Paweł Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS*, volume 2303, pages 205–222. Springer, 2002.
13. Antoine Meyer. Traces of term-automatic graphs. *ITA*, 42(3):615–630, 2008.
14. Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *BULLETIN of the American Mathematical Society*, 74:1025–1029, July 1968.
15. Chloe Rispal. The synchronized graphs trace the context-sensitive languages. *Electr. Notes Theor. Comput. Sci.*, 68(6):55–70, 2002.

[5] $\sqcup\!\sqcup$ denotes the shuffle product. For every $u, v \in \Sigma^*$ and $a, b \in \Sigma$, $u \sqcup\!\sqcup \varepsilon = \varepsilon \sqcup\!\sqcup u = u$, $au \sqcup\!\sqcup bv = a(u \sqcup\!\sqcup bv) \cup b(au \sqcup\!\sqcup v)$

A Properties of Operation Automata

In this section, we show that Rec is closed under union, intersection, iteration and contains the finite sets of operations.

Proposition 6. *Given two automata A_1 and A_2, there exists an automaton A such that* $\mathrm{Op}(A) = \mathrm{Op}(A_1) \cap \mathrm{Op}(A_2)$

Proof. We will construct an automaton which witness Prop 6. First, we ensure that the two automata are complete by adding a sink state if some transitions do not exist. We construct then the automaton A which is the product automaton of A_1 and A_2:

$Q = Q_{A_1} \times Q_{A_2}$

$I = I_{A_1} \times I_{A_2}$

$F = F_{A_1} \times F_{A_2}$

$\Delta = \{((q_1, q_2), \theta, (q'_1, q'_2)) \mid (q_1, \theta, q'_1) \in \Delta_{A_1} \wedge (q_2, \theta, q'_2) \in \Delta_{A_2}\}$
$\cup \{(((q_1, q_2), (q'_1, q'_2)), (q''_1, q''_2)) \mid ((q_1, q'_1), q''_1) \in \Delta_{A_1} \wedge ((q_2, q'_2), q''_2) \in \Delta_{A_2}\}$
$\cup \{((q_1, q_2), ((q'_1, q'_2), (q''_1, q''_2))) \mid (q_1, (q'_1, q''_1)) \in \Delta_{A_1} \wedge (q_2, (q'_2, q''_2)) \in \Delta_{A_2}\}$

If an operation admits a valid labelling in A_1 and in A_2, then the labelling which labels each states by the two states it has in its labelling in A_1 and A_2 is valid. If an operation admits a valid labelling in A, then, restricting it to the states of A_1 (resp A_2), we have a valid labelling in A_1 (resp A_2). □

Proposition 7. *Given two automata A_1 and A_2, there exists an automaton A such that* $\mathrm{Op}(A) = \mathrm{Op}(A_1) \cup \mathrm{Op}(A_2)$

Proof. We take the disjoint union of A_1 and A_2:

$Q = Q_{A_1} \uplus Q_{A_2}$

$I = I_{A_1} \uplus I_{A_2}$

$F = F_{A_1} \uplus F_{A_2}$

$\Delta = \Delta_{A_1} \uplus \Delta_{A_2}$

If an operation admits a valid labelling in A_1 (resp A_2), it is also a valid labelling in A. If an operation admits a valid labelling in A, as A is a disjoint union of A_1 and A_2, it can only be labelled by states of A_1 or of A_2 (by definition, there is no transition between states of A_1 and states of A_2) and then the labelling is valid in A_1 or in A_2. □

Proposition 8. *Given an automaton A, there exists A' which recognises* $\mathrm{Op}(A)^*$.

Proof. We construct A'.

$Q = Q_A \uplus \{q\}$

$I = I_A \cup \{q\}$

$F = F_A \cup \{q\}$

The set of transition Δ contains the transitions of A together with multiple copies of each transition ending with a state in F_A, modified to end in a state belonging to I_A

$\Delta = \Delta_A$
$\cup \{(q_1, \theta, q_i) \mid q_i \in I_A, \exists q_f \in F_A, (q_1, \theta, q_f) \in \Delta_A\}$
$\cup \{((q_1, q_2), q_i) \mid q_i \in I_A, \exists q_f \in F_A, ((q_1, q_2), q_f) \in \Delta_A\}$
$\cup \{(q_1, (q_2, q_i)) \mid q_i \in I_A, \exists q_f \in F_A, (q_1, (q_2, q_f)) \in \Delta_A\}$
$\cup \{(q_1, (q_i, q_2)) \mid q_i \in I_A, \exists q_f \in F_A, (q_1, (q_f, q_2)) \in \Delta_A\}$
$\cup \{(q_1, (q_i, q'_i)) \mid q_i, q'_i \in I_A, \exists q_f, q'_f \in F_A, (q_1, (q_f, q'_f)) \in \Delta_A\}$

For every $k \in \mathbb{N}$, if $D \in (\mathrm{Op}(A)^k)$, it has a valid labelling in A': The operation $\square$ has a valid labelling because q is initial and final. So it is true for $(\mathrm{Op}(A)^0)$ If it is true for $(\mathrm{Op}(A)^k)$, we take an operation G in $(\mathrm{Op}(A)^{k+1})$ and decompose it in D of $\mathrm{Op}(A)$ and F of $\mathrm{Op}(A)^k$ (or symmetrically, $D \in \mathrm{Op}(A)^k$ and $F \in \mathrm{Op}(A)^k$), such that $G \in D \cdot F$. The labelling which is the union of some valid labellings for D and F and labels the identified nodes with the labelling of F (initial states) is valid in A.

If an operation admits a valid labelling in A', we can separate several parts of the operation, separating on the added transitions, and we obtain a collection of operations of $\mathrm{Op}(A)$. Then we have a graph in $\mathrm{Op}(A)^k$ for a given k. Then $\mathrm{Op}(A') = \bigcup_{k \geq 0} \mathrm{Op}(A)^k$, then A' recognises $\mathrm{Op}(A)^*$. $\square$

Proposition 9. *Given an operation D, there exists an automaton A such that $\mathrm{Op}(A) = \{D\}$.*

Proof. If $D = (V, E)$, we take:
$Q = V$
I is the set of incoming vertices
F is the set of output vertices
$\Delta = \{(q, \theta, q') \mid (q, \theta, q') \in E\}$
$\cup \{(q, (q', q'')) \mid (q, 1, q') \in E \wedge (q, 2, q'') \in E\}$
$\cup \{((q, q'), q'') \mid (q, 1, q'') \in E \wedge (q', 2, q'') \in E\}$
The recognised connected part is D by construction. $\square$

B Normalised Automata

Definition 6. *An automaton is normalised if all its recognised operations are reduced.*

Theorem 2. *Given an operation automaton with tests, there exists a distinguished normalised operation automaton with tests which accepts the same language.*

Proof. The first thing to remark is that if we don't have any tree transitions, we have a higher-order stack automaton as in [5] and that the notions of normalised automaton coincide. The idea is thus to separate the automaton in two parts, one containing only tree transitions and the other stack transitions, to normalise each part separately and then to remove the useless transitions used to separate the automaton.

$$\begin{tikzcd}
 & Q_s & Q_{t_1} \\
q_{t_1} \arrow[r, "\mathrm{id}"] & q_s \arrow[r, "\mathrm{id"}] \arrow[u, dashed] & q_{t_2} \arrow[u, dashed] \\
Q_{t_2} \arrow[u, dashed] & Q_s \arrow[u, dashed] &
\end{tikzcd}$$

Fig. 5: Step 1: The splitting of a state q

Step 1: In this transformation, we will use a new special basic operation: id such that its associated operation D_{id} is the following DAG: $V_{D_{\mathrm{id}}} = \{x, y\}$ and $E_{D_{\mathrm{id}}} = \{(x, \mathrm{id}, y)\}$. For every stack tree t and any integer $i \leq |\mathrm{fr}(t)|$, $\mathrm{id}_{(i)}(t) = t$. We will use this operation to separate our DAGs in several parts linked with id operations, and will remove them at the end of the transformation. We suppose that we start with an automaton without such id transitions.

We begin by splitting the set of control states of the automaton into three parts. We create three copies of Q:

- Q_s which are the sources and targets of all the stack transitions, target of id transitions from Q_{t_1} and source of id-transitions to Q_{t_2}.
- Q_{t_1} which are the targets of all the tree transitions and the sources of id-transitions to Q_s.
- Q_{t_2} which are the sources of all the tree transitions and the targets of id-transitions from Q_s.

The idea of what we want to obtain is depicted in Fig. 5.

Formally, we replace the automaton $A = (Q, I, F, \Delta)$ by $A_1 = (Q', I', F', \Delta')$ with:

$$\begin{aligned}
Q' &= \{q_{t_1}, q_{t_2}, q_s \mid q \in Q\} \\
I' &= \{q_s \mid q \in I\} \\
F' &= \{q_s \mid q \in F\} \\
\Delta &= \{(q_s, \theta, q'_s) \mid (q, \theta, q') \in \Delta\} \\
&\quad \cup \{(q_{t_2}, (q'_{t_1}, q''_{t_1})) \mid (q, (q', q'')) \in \Delta\} \\
&\quad \cup \{((q_{t_2}, q'_{t_2}), q''_{t_1}) \mid ((q, q'), q'') \in \Delta\} \\
&\quad \cup \{(q_{t_2}, \mathrm{copy}^1_n, q'_{t_1}) \mid (q, \mathrm{copy}^1_n, q') \in \Delta\} \\
&\quad \cup \{(q_{t_2}, \overline{\mathrm{copy}}^1_n, q'_{t_1}) \mid (q, \overline{\mathrm{copy}}^1_n, q') \in \Delta\} \\
&\quad \cup \{(q_{t_1}, \mathrm{id}, q_s), (q_s, \mathrm{id}, q_{t_2}) \mid q \in Q\}
\end{aligned}$$

where for every $q \in Q$, q_{t_1}, q_{t_2}, q_s are fresh states.

Lemma 1. *A and A_1 recognise the same relation.*

Proof. To prove this lemma, we prove that for every operation D recognised by A, there is an operation D' recognised by A_1 such that $R_D = R_{D'}$, and vice versa.

Let us take D recognised by A. We prove, by induction on the structure of D that we can construct D' such that $R_D = R_{D'}$ and for every labelling ρ_D of D consistent with Δ, with I_D labelled by $\boldsymbol{q}$ and O_D by $\boldsymbol{q'}$, there exists ρ'_D a labelling of D' consistent with Δ' such that $I_{D'}$ is labelled by $\boldsymbol{q_s}$ and $O_{D'}$ by $\boldsymbol{q'_s}$.

If $D = \Box$, we take $D' = \Box$. We have $R_D = R_{D'}$. For every labelling ρ_D which labels the unique node of D by q, we take $\rho_{D'}$ which labels the unique node of D' by q_s. These labellings are consistent by Δ and Δ', by vacuity.

Suppose now that we have F and F' such that for every labelling ρ_F we can define a labelling $\rho_{F'}$ satisfying the previous condition. Let us consider the following cases:

- $D = (F \cdot_{1,1} D_\theta) \cdot_{1,1} G$, for $\theta \in \{\mathrm{copy}^1_n, \overline{\mathrm{copy}}^1_n\}$. We call x the output node of F and y the input node of G. We have $V_D = V_F \cup V_G$ and $E_D = E_F \cup E_G \cup \{x \xrightarrow{\theta} y\}$.
  By induction hypothesis, we consider F' and G', and construct $D' = (((F' \cdot_{1,1} D_{\mathrm{id}}) \cdot_{1,1} D_\theta) \cdot_{1,1} D_{\mathrm{id}}) \cdot_{1,1} G'$, with $V_{D'} = V_{F'} \cup V_{G'} \cup \{x'_1, x'_2\}$ and $E_{D'} = E_{F'} \cup E_{G'} \cup \{x' \xrightarrow{\mathrm{id}} x'_1, x'_1 \xrightarrow{\theta} x'_2, x'_2 \xrightarrow{\mathrm{id}} y'\}$, where x' is the output node of F' and y' the input node of G'.
  We take ρ_D a labelling of D and ρ_F (resp. ρ_G) its restriction to F (resp. G). We have $\rho_D(x) = q$ and $\rho_D(y) = q'$. By induction hypothesis, we consider $\rho_{F'}$ (resp. $\rho_{G'}$) the corresponding labelling of F' (resp. G'), with $\rho_{F'}(x') = q_s$ (resp. $\rho_{G'}(y') = q'_s$). Then, we construct $\rho_{D'} = \rho_{F'} \cup \rho_{G'} \cup \{x'_1 \to q_{t_2}, x'_2 \to q'_{t_1}\}$.
  As ρ_D is consistent with Δ, (q, θ, q') is in Δ, then by construction $(q_{t_2}, \theta, q'_{t_1})$ is in Δ'. We have also $(q_s, \mathrm{id}, q_{t_2})$ and $(q'_{t_1}, \mathrm{id}, q'_s)$ are in Δ'. Then, ρ'_D is consistent with Δ'.
  To prove that $R_D = R_{D'}$, we just have to remark that, from the definition of application of operation, we have for every stack tree t and integer i, we have $D'_{(i)}(t) = G'_{(i)}(\mathrm{id}_{(i)}(\theta_{(i)}(\mathrm{id}_{(i)}(F'_{(i)}(t))))) = G_{(i)}(\theta_{(i)}(F_{(i)}(t))) = D_{(i)}(t)$.

  The other cases being similar, we just give D' and $\rho_{D'}$ and leave the details to the reader.
- $D = (F \cdot_{1,1} D_\theta) \cdot_{1,1} G$, for $\theta \in Ops_{n-1} \cup \mathcal{T}_{n-1}$. We call x the output node of F and y the input node of G. We have $V_D = V_F \cup V_G$ and $E_D = E_F \cup E_G \cup \{x \xrightarrow{\theta} y\}$.
  By induction hypothesis, we consider F' and G', and construct $D' = (F' \cdot_{1,1} \theta) \cdot_{1,1} G'$, with $V_{D'} = V_{F'} \cup V_{G'}$ and $E_{D'} = E_{F'} \cup E_{G'} \cup \{x' \xrightarrow{\theta} y'\}$, where x' is the output node of F' and y' the input node of G'.
  We take ρ_D a labelling of D and ρ_F (resp. ρ_G) its restriction to F (resp. G). We have $\rho_D(x) = q$ and $\rho_D(y) = q'$. By induction hypothesis, we consider $\rho_{F'}$ (resp. $\rho_{G'}$) the corresponding labelling of F' (resp. G'), with $\rho_{F'}(x') = q_s$ (resp. $\rho_{G'}(y') = q'_s$). Then, we construct $\rho_{D'} = \rho_{F'} \cup \rho_{G'}$.
- $D = ((F \cdot_{1,1} D_{\mathrm{copy}^2_n}) \cdot_{2,1} H) \cdot_{1,1} G$. We call x the output node of F, y the input node of G and z the input node of H. We have $V_D = V_F \cup V_G \cup V_H$ and $E_D = E_F \cup E_G \cup E_H \cup \{x \xrightarrow{1} y, x \xrightarrow{2} z\}$.

By induction hypothesis, we consider F', G' and H', and construct $D' = (((((F \cdot_{1,1} D_{\mathrm{id}}) D_{\mathrm{copy}_n^2}) \cdot_{2,1} D_{\mathrm{id}}) \cdot_{2,1} H) \cdot_{1,1} D_{\mathrm{id}}) \cdot_{1,1} G$, with $V_{D'} = V_{F'} \cup V_{G'} \cup V_{H'} \cup \{x'_1, x'_2, x'_3\}$ and $E_{D'} = E_{F'} \cup E_{G'} \cup E_{H'} \{x' \xrightarrow{\mathrm{id}} x'_1, x'_1 \xrightarrow{1} x'_2, x'_1 \xrightarrow{2} x'_3, x'_2 \xrightarrow{\mathrm{id}} y', x'_3 \xrightarrow{\mathrm{id}} z'\}$, where x' is the output node of F', y' the input node of G' and z' the input node of H'.

We take ρ_D a labelling of D and ρ_F (resp. ρ_G, ρ_H) its restriction to F (resp. G, H). We have $\rho_D(x) = q$, $\rho_D(y) = q'$ and $\rho_D(z) = q''$. By induction hypothesis, we consider $\rho_{F'}$ (resp. $\rho_{G'}$,$\rho_{H'}$) the corresponding labelling of F' (resp. G',H'), with $\rho_{F'}(x') = q_s$ (resp. $\rho_{G'}(y') = q'_s$, $\rho_{H'}(z') = q''_s$). Then, we construct $\rho_{D'} = \rho_{F'} \cup \rho_{G'} \cup \rho_{H'} \cup \{x'_1 \to q_{t_2}, x'_2 \to q'_{t_1}, x'_3 \to q''_{t_1}\}$.

- $D = (F \cdot_{1,1} (G \cdot_{1,2} D_{\overline{\mathrm{copy}}_n^2})) \cdot_{1,1} H$. We call x the output node of F, y the output node of G and z the input node of H. We have $V_D = V_F \cup V_G \cup V_H$ and $E_D = E_F \cup E_G \cup E_H \cup \{x \xrightarrow{\bar{1}} z, y \xrightarrow{\bar{2}} z\}$.

  By induction hypothesis, we consider F', G' and H', and construct $D' = (((F \cdot_{1,1} D_{\mathrm{id}}) \cdot_{1,1} ((G \cdot_{1,1} D_{\mathrm{id}}) \cdot_{1,2} D_{\overline{\mathrm{copy}}_n^2})) \cdot_{1,1} D_{\mathrm{id}}) \cdot_{1,1} H$, with $V_{D'} = V_{F'} \cup V_{G'} \cup V_{H'} \cup \{x'_1, x'_2, x'_3\}$ and $E_{D'} = E_{F'} \cup E_{G'} \cup E_{H'} \{x' \xrightarrow{\mathrm{id}} x'_1, y' \xrightarrow{\mathrm{id}} x'_2, x'_1 \xrightarrow{\bar{1}} x'_3, x'_2 \xrightarrow{\bar{2}} x'_3, x'_3 \xrightarrow{\mathrm{id}} z'\}$, where x' is the output node of F', y' the input node of G' and z' the input node of H'.

  We take ρ_D a labelling of D and ρ_F (resp. ρ_G, ρ_H) its restriction to F (resp. G, H). We have $\rho_D(x) = q$, $\rho_D(y) = q'$ and $\rho_D(z) = q''$. By induction hypothesis, we consider $\rho_{F'}$ (resp. $\rho_{G'}$, $\rho_{H'}$) the corresponding labelling of F' (resp. G', H'), with $\rho_{F'}(x') = q_s$ (resp. $\rho_{G'}(y') = q'_s$, $\rho_{H'}(z') = q''_s$). Then, we construct $\rho_{D'} = \rho_{F'} \cup \rho_{G'} \cup \rho_{H'} \cup \{x'_1 \to q_{t_2}, x'_2 \to q'_{t_2}, x'_3 \to q''_{t_1}\}$.

- $D = (((((F \cdot_{1,1} D_{\mathrm{copy}_n^2}) \cdot_{2,1} H) \cdot_{1,1} G) \cdot_{1,1} D_{\overline{\mathrm{copy}}_n^2}) \cdot_{1,1} K$. We call x the output node of F, y_1 the input node of G and y_2 its output node, z_1 the input node of H and z_2 its output node and w the input node of K. We have $V_D = V_F \cup V_G \cup V_H \cup V_K$ and $E_D = E_F \cup E_G \cup E_H \cup E_K \cup \{x \xrightarrow{1} y_1, x \xrightarrow{2} z_1, y_2 \xrightarrow{\bar{1}} t, z_2 \xrightarrow{\bar{2}} t\}$.

  By induction hypothesis, we consider F', G', H' and K', and construct $D' = ((((((F' \cdot_{1,1} D_{\mathrm{id}}) \cdot_{1,1} D_{\mathrm{copy}_n^2}) \cdot_{2,1} (D_{\mathrm{id}} \cdot_{1,1} H')) \cdot_{1,1} (D_{\mathrm{id}} \cdot_{1,1} G')) \cdot_{1,1} D_{\overline{\mathrm{copy}}_n^2}) \cdot_{1,1} D_{\mathrm{id}}) \cdot_{1,1} K'$, with $V_{D'} = V_{F'} \cup V_{G'} \cup V_{H'} \cup V_{K'} \cup \{x'_1, x'_2, x'_3, x'_4, x'_5, x'_6\}$ and $E_{D'} = E_{F'} \cup E_{G'} \cup E_{H'} \cup E_{K'} \{x' \xrightarrow{\mathrm{id}} x'_1, x'_1 \xrightarrow{1} x'_2, x'_1 \xrightarrow{2} x'_3, x'_2 \xrightarrow{\mathrm{id}} y'_1, x'_3 \xrightarrow{\mathrm{id}} z'_1, y'_2 \xrightarrow{\mathrm{id}} x'_4, z'_2 \xrightarrow{\mathrm{id}} x'_5, x'_4 \xrightarrow{\bar{1}} x'_6, x'_5 \xrightarrow{\bar{2}} x'_6, x'_6 \xrightarrow{\mathrm{id}} t'\}$, where x' is the output node of F', y'_1 the input node of G', y'_2 its output node, z'_1 the input node of H', z'_2 its output node and t' the input node of K'.

  We take ρ_D a labelling of D_D and ρ_F (resp. ρ_G, ρ_H, ρ_K) its restriction to F (resp. G, H, K). We have $\rho_D(x) = q$, $\rho_D(y_1) = q'$, $\rho_D(z_1) = q''$, $\rho_D(y_2) = r'$, $\rho_D(z_2) = r''$ and $\rho_D(t) = r''$. By induction hypothesis, we consider $\rho_{F'}$ (resp. $\rho_{G'}$, $\rho_{H'}$, $\rho_{K'}$) the corresponding labelling of F' (resp. G', H', K'), with $\rho_{F'}(x') = q_s$ (resp. $\rho_{G'}(y'_1) = q'_s$, $\rho_{H'}(z'_1) = q''_s$, $\rho_{G'}(y'_2) = r'_s$, $\rho_{H'}(z'_2) = r''_s$, $\rho_{K'}(t') = r''_s$). Then, we construct $\rho_{D'} = \rho_{F'} \cup \rho_{G'} \cup \rho_{H'} \cup \{x'_1 \to q_{t_2}, x'_2 \to q'_{t_1}, x'_3 \to q''_{t_1}, x'_4 \to r_{t_2}, x'_5 \to r'_{t_2}, x'_6 \to r''_{t_1}\}$.

To do the other direction, we take D' recognised by A_1 and show that we can construct D recognised by A with $R_D = R_{D'}$ by an induction on the structure of D' similar to the previous one (for each id transition, we do not modify the constructed DAG and for all other transition, we add them to the DAG). All the arguments are similar to the previous proof, so we let the reader detail it. $\square$

We start by normalising the tree part of the automaton. To do so, we just have to prevent the automaton to recognise DAGs which contain $((D_{\mathrm{copy}_n^2} \cdot_{1,1} F_1) \cdot_{2,1} F_2) \cdot_{1,1} D_{\overline{\mathrm{copy}}_n^2}$, or $(D_{\mathrm{copy}_n^1} \cdot_{1,1} F) \cdot_{1,1} D_{\overline{\mathrm{copy}}_n^1}$ as a subDAG. Such a subDAG will be called a bubble. However, we do not want to modify the recognised relation. We will do it in two steps: first we allow the automaton to replace the bubbles with equivalent tests (after remarking that a bubble can only be a test) in any recognised DAG (step 2), and then by ensuring that there won't be any $\overline{\mathrm{copy}}_n^i$ transition below the first copy_n^j transition (step 3).

Step 2: Let $A_1 = (Q, I, F, \Delta)$ be the automaton obtained after step 1. Given two states q_1, q_2, we denote by $L_{A_{q_1,q_2}}$ the set $\{s \in \mathit{Stacks}_{n-1} \mid \exists D \in \mathcal{D}(A_1), D_{(1)}(s) = s\}$ where A_{q_1,q_2} is a copy of A_1 in which we take q_1 as the unique initial state and q_2 as the unique final state. In other words, $L_{A_{q_1,q_2}}$ is the set of $(n-1)$-stacks such that the trees with one node labelled by this stack remains unchanged by an operation recognised by A_{q_1,q_2}. We define $A_2 = (Q, I, F, \Delta')$ with

$$\begin{aligned}\Delta' = \Delta\\ &\cup \{(q_s, T_{L_{A_{r_s,r'_s}} \cap L_{A_{s_s,s'_s}}}, q'_s) \mid (q_{t_2}, (r_{t_1}, s_{t_1})), ((r'_{t_2}, s'_{t_2}), q'_{t_1}) \in \Delta\}\\ &\cup \{(q_s, T_{L_{r_s,s'_s}}, q'_s \mid (q_{t_2}, \mathrm{copy}_n^1, r_{t_1}), (r'_{t_2}, \overline{\mathrm{copy}}_n^1, q'_{t_1}) \in \Delta\}\end{aligned}$$

The idea of the construction is depicted in Fig. 6.

We give the following lemma for the binary bubble. The case of the unary bubble is very similar and thus if left to the reader.

Lemma 2. *Let $C_1 = (Q_{C_1}, \{i_{C_1}\}, \{f_{C_1}\}, \Delta_{C_1})$ and $C_2 = (Q_{C_2}, \{i_{C_2}\}, \{f_{C_2}\}, \Delta_{C_2})$ be two automata recognising DAGs without tree operations. The two automata $B_1 = (Q_1, I, F, \Delta_1)$ and $B_2 = (Q_2, I, F, \Delta_2)$, with $I = \{q_1\}$, $F = \{q_2\}$, $Q_1 = \{q_1, q_2\}$, $\Delta_1 = \{(q_1, T_{L_{C_1} \cap L_{C_2}}, q_2)\}$, $Q_2 = \{q_1, q_2\} \cup Q_{C_1} \cup Q_{C_2}$ and $\Delta_2 = \{(q_1, (i_{C_1}, i_{C_2})), ((f_{C_1}, f_{C_2}), q_2)\} \cup \Delta_{C_1} \cup \Delta_{C_2}$ recognise the same relation.*

Proof. An operation D recognised by B_2 is of the form $D = D_{\mathrm{copy}_n^2} \cdot_{1,1} (F_1 \cdot_{1,1} (F_2 \cdot_{2,2} D_{\overline{\mathrm{copy}}_n^2}))$, where F_1 is recognised by C_1 and F_2 by C_2. We have:

$$\begin{aligned}D_{(i)}(t) &= \overline{\mathrm{copy}}^2_{n(i)}(F_{1(i)}(F_{2(i+1)}(\mathrm{copy}^2_{n(i)}(t))))\\ &= \overline{\mathrm{copy}}^2_{n(i)}(F_{1(i)}(F_{2(i+1)}(t \cup \{u_i1 \mapsto t(u_i), u_i2 \mapsto t(u_i)\})))\\ &= \overline{\mathrm{copy}}^2_{n(i)}(t \cup \{u_i1 \mapsto F_1(t(u_i)), u_i2 \mapsto F_2(t(u_i))\}).\end{aligned}$$

So this operation is defined if and only if $F_1(t(u_i)) = F_2(t(u_i)) = t(u_i)$. In this case, $D_i(t) = t$. Thus, B_2 accepts only operations which are tests, and these tests are the intersection of the tests recognised by C_1 and C_2. So the relation recognised by B_2 is exactly the relation recognised by $T_{L_{C_1} \cap L_{C_2}}$, which is the only operation recognised by B_1. $\square$

We have the following corollary as a direct consequence of this lemma.

Corollary 1. *A_1 and A_2 recognises the same relation.*

Indeed, all the new operations recognised do not modify the relation recognised by the automaton as each test was already present in the DAGs containing a bubble.

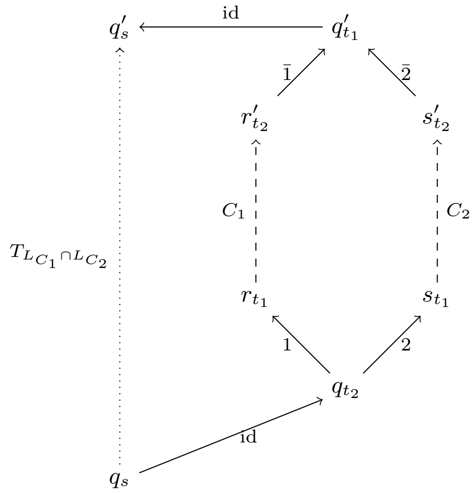


Fig. 6: Step 2: The added test transition to shortcut the bubble is depicted with a dotted line

Step 3: Suppose that $A_2 = (Q, I, F, \Delta)$ is the automaton obtained after step 2. We now want to really forbid these bubbles. To do so, we split the control states automaton in two parts: We create 2 copies of Q:

- Q_d which are target of no copy_n^d transition,
- Q_c which are source of no $\overline{\mathrm{copy}}_n^d$ transition.

We construct $A_3 = (Q', I', F', \Delta')$ with:

$Q' = \{q_d, q_c \mid q \in Q\}$

$I' \ = \{q_d, q_c \mid q \in I\}$

$F' = \{q_d, q_c \mid q \in F\}$

$\Delta' = \{(q_d, \theta, q'_d), (q_c, \theta, q'_c) \mid (q, \theta, q') \in \Delta, \theta \in Ops_{n-1} \cup \mathcal{T}_{n-1} \cup \{\mathrm{id}\}\}$

$\quad \cup \{((q_d, q'_d), q''_d) \mid ((q, q'), q'') \in \Delta\}$

$\quad \cup \{(q_d, \overline{\mathrm{copy}}_n^1, q'_d) \mid (q, \overline{\mathrm{copy}}_n^1, q') \in \Delta\}$

$\quad \cup \{(q_c, (q'_c, q''_c)), (q_d, (q'_c, q''_c)) \mid (q, (q', q'')) \in \Delta\}$

$\quad \cup \{(q_c, \mathrm{copy}_n^1, q'_c), (q_d, \mathrm{copy}_n^1, q'_c) \mid (q, \mathrm{copy}_n^1, q') \in \Delta\}$

Lemma 3. *A_2 and A_3 recognise the same relation*

Proof. A_3 recognises the operations recognised by A_2 which contain no bubble. Indeed, every labelling of such an operation in A_2 can be modified to be a labelling in A_3 (left to the reader). Conversely, each operation recognised by A_3 is recognised by A_2.

Let us take D recognised by A_2 which contains at least one bubble. Suppose that D contains a bubble F and that $D = D[F]_x$ where D is a DAG with one bubble less and we obtain D by replacing the node x by F in D. From step 2, there exist four states of A_2, r_s, r'_s, s_s, s'_s such that $G = D[T_{L_{A_{r_s,r'_s}}} \cap L_{A_{s_s,s'_s}}]_x$ is recognised by A_2. Then $R_D \subseteq R_G$, and G has one less bubble than D.

Iterating this process, we obtain an operation D' without any bubble such that $R_D \subseteq R_{D'}$ and D' is recognised by A_2. As it contains no bubble, it is also recognised by A_3.

Then every relation recognised by an operation with bubbles is already included in the relation recognised by an operation without bubbles. Then A_2 and A_3 recognise the same relation. □

We call the destructive part the restriction $A_{3,d}$ of A_3 to Q_d and the constructive part its restriction $A_{3,c}$ to Q_c.

Step 4: We consider an automaton A_3 obtained after the previous step. Observe that in the two previous steps, we did not modify the separation between Q_{t_1}, Q_{t_2} and Q_s. We call $A_{3,s}$ the restriction of A_3 to Q_s.

We now want to normalise $A_{3,s}$. As this part of the automaton only contains transitions labelled by operations of $Ops_{n-1} \cup \mathcal{T}_{n-1}$, we can consider it as an automaton over higher-order stack operations. So we will use the process of normalisation over higher-order stack operations defined in [5]. For each pair (q_s, q'_s) of states in Q_s, we construct the normalised automaton A_{q_s,q'_s} of A' where A' is a copy of $A_{3,s}$ where $I_{A'} = \{q_s\}$ and $F_{A'} = \{q'_s\}$. We suppose that these automata are distinguished, i.e. that states of $I_{A_{q_s,q'_s}}$ are target of no transitions and states of $F_{A_{q_s,q'_s}}$ are source of no transitions. We moreover suppose that it is not possible to do two test transitions in a row (this is not a strong supposition because such a sequence would not be normalised, but it is worth noticing it).

We replace $A_{3,s}$ with the union of all the A_{q_s,q'_s}: we define $A_4 = (Q', I', F', \Delta')$:

$$
\begin{aligned}
Q' &= Q_{t_1} \cup Q_{t_2} \cup \textstyle\bigcup_{q_s,q'_s} Q_{A_{q_s,q'_s}}\\
I' &= \textstyle\bigcup_{q_s \in I, q'_s \in Q_s} I_{A_{q_s,q'_s}}\\
F' &= \textstyle\bigcup_{q_s \in Q_s, q'_s \in F} F_{A_{q_s,q'_s}}\\
\Delta' &= \{K \in \Delta \mid K = (q,(q',q'')) \vee K = ((q,q'),q'') \vee K = (q, \mathrm{copy}^1_n, q')\\
&\quad \vee K = (q, \overline{\mathrm{copy}}^1_n, q')\}\\
&\quad \cup \textstyle\bigcup_{q_s,q'_s \in Q_s} \Delta_{A_{q_s,q'_s}}\\
&\quad \cup \{(q_{t_1}, \mathrm{id}, i) \mid (q_{t_1}, \mathrm{id}, q'_s) \in \Delta, i \in \textstyle\bigcup_{q''_s \in Q} I_{A_{q'_s,q''_s}}\}\\
&\quad \cup \{(f, \mathrm{id}, q_{t_2}) \mid (q'_s, \mathrm{id}, q_{t_2}) \in \Delta, f \in \textstyle\bigcup_{q''_s \in Q} F_{A_{q''_s,q'_s}}\}\\
&\quad \cup \{(q_{t_1}, \mathrm{id}, f) \mid (q_{t_1}, \mathrm{id}, q'_s) \in \Delta, f \in \textstyle\bigcup_{q''_s \in Q} F_{A_{q''_s,q'_s}}\}\\
&\quad \cup \{(i, \mathrm{id}, q_{t_2}) \mid (q'_s, \mathrm{id}, q_{t_2}) \in \Delta, i \in \textstyle\bigcup_{q''_s \in Q} I_{A_{q'_s,q''_s}}\}
\end{aligned}
$$

Lemma 4. *A_3 and A_4 recognise the same relation.*

Proof. For every operation D recognised by A_3, we can construct D' by replacing each sequence of $Ops_{n-1} \cup \mathcal{T}_{n-1}$ operations by their reduced sequence, which is recognised by A_4 and define the same relation. The details are left to the reader.

Conversely, for every D' recognised by A_4, we can construct D recognised by A_3 which define the same relation, by replacing every reduced sequence of $Ops_{n-1} \cup \mathcal{T}_{n-1}$ operations by a sequence of $Ops_{n-1} \cup \mathcal{T}_{n-1}$ operations defining the same relation such that D is recognised by A_3. We leave the details to the reader. □

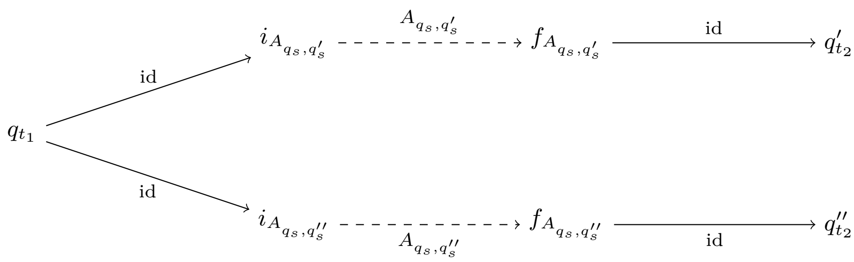


Fig. 7: Step 4: The splitting of the stack part of the automaton

Step 5: We now have a normalised automaton, except that we have id transitions. We remove them by a classical saturation mechanism. Observe that in all the previous steps, we never modified the separation between Q_{t_1}, Q_s and Q_{t_2}, so that all id transitions are from Q_{t_1} to Q_s and from Q_s to Q_{t_2}. We take $A_4 = (Q, I, F, \Delta)$ obtained after the previous step. We construct $A_5 = (Q', I', F', \Delta')$

with $Q' = Q_s$, $I' = I$, $F' = F$ and

$$\begin{aligned}
\Delta' = {} & \Delta \setminus \{(q, \mathrm{id}, q') \in \Delta\} \\
& \cup \{(q_s, \mathrm{copy}^1_n, q'_s) \mid \exists q''_{t_2}, q'''_{t_1}, (q''_{t_2}, \mathrm{copy}^1_n, q'''_{t_1}), (q'''_{t_1}, \mathrm{id}, q'_s), (q_s, \mathrm{id}, q''_{t_2}) \in \Delta\} \\
& \cup \{(q_s, \overline{\mathrm{copy}}^1_n, q'_s) \mid \exists q''_{t_2}, q'''_{t_1}, (q''_{t_2}, \overline{\mathrm{copy}}^1_n, q'''_{t_1}), (q'''_{t_1}, \mathrm{id}, q'_s), (q_s, \mathrm{id}, q''_{t_2}) \in \Delta\} \\
& \cup \{(q_s, (q'_s, q''_s)) \mid \exists q_1, q_2, q_3, (q_1, (q_2, q_3)), (q_s, \mathrm{id}, q_1), (q_2, \mathrm{id}, q'_s), (q_3, \mathrm{id}, q''_s) \in \Delta\} \\
& \cup \{((q_s, q'_s), q''_s) \mid \exists q_1, q_2, q_3, ((q_1, q_2), q_3), (q_s, \mathrm{id}, q_1), (q'_s, \mathrm{id}, q_2), (q_3, \mathrm{id}, q''_s) \in \Delta\}
\end{aligned}$$

Lemma 5. *A_4 and A_5 recognise the same relation.*

Proof. We prove it by an induction on the structure of relations similar to the one of step 1, so we leave it to the reader. □

Step 6: We now split the control states set into two parts:

- Q_T, the states which are target of all and only test transitions and source of no test transition,
- Q_C, the states which are source of all test transitions and target of no test transition.

Given automaton $A_5 = (Q, I, F, \Delta)$ obtained from the previous step, we define $A_6 = (Q', I', F', \Delta')$ with

$$\begin{aligned}
Q' &= \{q_T, q_C \mid q \in Q\}, \\
I' &= \{q_C \mid q \in I\}, \\
F' &= \{q_T, q_C \mid q \in F\}, \\
\Delta' &= \{(q_C, \theta, q'_C), (q_T, \theta, q'_C) \mid (q, \theta, q') \in \Delta, \theta \in Ops_{n-1} \cup \mathcal{T}_{n-1}\{\mathrm{copy}^1_n, \overline{\mathrm{copy}}^1_n\}\} \\
&\quad \cup \{((q_C, q'_C), q''_C), ((q_C, q'_T), q''_C), ((q_T, q'_C), q''_C), ((q_T, q'_T), q''_C) \mid ((q, q'), q'') \in \Delta\} \\
&\quad \cup \{(q_C, (q'_C, q''_C)), (q_T, (q'_C, q''_C)) \mid (q, (q', q'')) \in \Delta\} \\
&\quad \cup \{(q_C, T_L, q'_T) \mid (q, T_L, q') \in \Delta\}.
\end{aligned}$$

Lemma 6. *A_5 and A_6 recognise the same relation.*

Proof. As, from step 4 it is not possible to have two successive test transitions, the set of recognised operations is the same in both automata, only the labelling is modified. The details are left to the reader. □

Finally, we suppose that an automaton obtained by these steps is distinguished, i.e. initial states are target of no transition and final states are source of no transition. If not, we can distinguish it by a classical transformation (as in the case of word automata). We now have a normalised automaton with tests A_6 obtained after the application of the six steps which recognises the same relation as the initial automaton A. In subsequent constructions, we will be considering the subsets of states Q_T, Q_C, Q_d, Q_c as defined in steps 6 and 3, and $Q_{u,d} = Q_u \cap Q_d$ with $u \in \{T, C\}$ and $d \in \{d, c\}$. □

C Finite set interpretation

In this section, we formally define a finite set interpretation I_R from $\Delta^n_{\Sigma\cup\{1,2\}}$ to the rewriting graph of a GSTRS R. In the whole section, we consider a distinguished normalised automaton with tests $A = (Q, I, F, \Delta)$ recognising R^*, constructed according to the process of the previous section.

Let us first formally define a possible presentation of the graph $\Delta^n_{\Sigma\cup\{1,2\}}$. Vertices of this graph are n-stacks over alphabet $\Sigma \cup \{1, 2\}$, and there is an edge (x, θ, y) in $\Delta^n_{\Sigma\cup\{1,2\}}$ if $\theta \in Ops_n(\Sigma \cup \{1, 2\}) \cup \mathcal{T}_n$ and $y = \theta(x)$.

Since we are building an unlabelled graph, our interpretation consists of these formulæ:

- $\delta(X)$ which describes which subsets of $Stacks_n(\Sigma \cup \{1, 2\})$ are in the graph,
- $\Psi_D(X_s, X_t)$ which is true if $\mathcal{R}_D(s, t)$, for $D \in R$,
- $\phi(X_s, X_t)$ which is true if $\mathcal{R}(A)(s, t)$.

C.1 Notations and Technical Formulæ

We will use the push_d and pop_d operations to simplify the notations. They have the usual definition (as can be encountered in [5]), but notice that we can define them easily with our operations: $\mathrm{push}_d(x) = y$ if there exists $z \in V, a \in \Sigma\cup\{1, 2\}$ such that $x \xrightarrow{\mathrm{copy}_1} z \xrightarrow{\mathrm{rew}_{\mathrm{a,d}}} y$, and $\mathrm{pop}_d(x) = y$ if $x = \mathrm{push}_d(y)$. Observe that $\mathrm{push}_d(x)$ and $\mathrm{pop}_d(x)$ are well defined as there can only be one a such that the definition holds: the a which is the topmost letter of x. We extend this notations to push and pop words to simplify notations.

We first define some formulæ over $\Delta^n_{\Sigma\cup\{1,2\}}$ which will be used to construct the set of stacks used to represent stack trees over $\Delta^n_{\Sigma\cup\{1,2\}}$.

Given $\theta \in Ops_{n-1}(\Sigma) \cup \mathcal{T}_{n-1}$, we define ψ_θ such that, given two n-stacks x, y, $\psi_\theta(x, y) = x \xrightarrow{\theta} y$. $\psi_{\mathrm{copy}^i_n,d}(x, y) = \exists a \in \Sigma, z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8 \in V, x \xrightarrow{\mathrm{copy}_1} z_1 \xrightarrow{\mathrm{rew}_{\mathrm{a,i}}} z_2 \xrightarrow{\mathrm{copy}_1} z_3 \xrightarrow{\mathrm{rew}_{\mathrm{i,d}}} z_4 \xrightarrow{\mathrm{copy}_n} z_5 \xrightarrow{\mathrm{rew}_{\mathrm{d,i}}} z_6 \xrightarrow{\overline{\mathrm{copy}_1}} z_7 \xrightarrow{\mathrm{rew}_{\mathrm{i,a}}} z_8 \xrightarrow{\overline{\mathrm{copy}_1}} y$.

$\psi_\theta(x, y)$ is true if y is obtained by applying θ to x. $\psi_{\mathrm{copy}^i_n,d}(x, y)$ is true if y is obtained by adding i and d to the topmost 1-stack of x, duplicating its topmost $(n-1)$-stack and then removing d and i from its topmost 1-stack.

We now give a technical formula which ensures that a given stack y is obtained from a stack x using only the previous formulæ: $\mathrm{Reach}(x, y)$

$$\mathrm{Reach}(x, y) = \forall X, ((x \in X \wedge \forall z, z', (z \in X \wedge (\bigvee_{\theta\in Ops_{n-1}\cup\mathcal{T}_{n-1}} \psi_\theta(z, z') \vee \bigvee_{i\in\{1,2\}} \bigvee_{d\leq i} \psi_{\mathrm{copy}^i_n,d}(z, z'))) \Rightarrow z' \in X) \Rightarrow y \in X)$$

This formula is true if for every set of n-stacks X, if x is in X and X is closed by the relations defined ψ_θ and $\psi_{\mathrm{copy}^i_n,d}$, then y is in X.

Lemma 7. *For all n-stacks $x = [x_1, \cdots, x_m]_n$ and $y = [y_1, \cdots, y_{m'}]_n$,*
$$\mathrm{Reach}(x, y)$$
holds if and only if $y = [x_1, \cdots, x_{m-1}, \mathrm{push}_{i_m d_m}(y_m), \mathrm{push}_{i_{m+1} d_{m+1}}(y_{m+1}), \cdots, \mathrm{push}_{i_{m'-1} d_{m'-1}}(y_{m'-1}), y_{m'}]_n$ where for all $m \leq j < m'$, $i_j \in \{1, 2\}$, $d_j \leq i_j$ and for all $m \leq j \leq m'$, there exists a sequence of operations $\rho_j \in (Ops_{n-1}(\Sigma) \cup \mathcal{T}_{n-1})^*$ such that $\rho_j(x_m, y_j)$.

Corollary 2. *For every n-stack x and $a \in \Sigma$, $\mathrm{Reach}([a]_n, x)$ holds if and only if there exist a stack tree t and a node u such that $x = \mathrm{Code}(t, u)$.*

Proof. Suppose that there exist a stack tree t and a node u such that $x = \mathrm{Code}(t, u)$. Then $x = [\mathrm{push}_{\#(\varepsilon)u_1}(t(\varepsilon)), \mathrm{push}_{\#(u_{\leq 1})u_2}(t(u_{\leq 1})), \cdots, \mathrm{push}_{\#(u_{\leq |u|-1})u_{|u|}}(t(u_{\leq |u|-1})), t(u)]_n$. As for every i, $t(u_{\leq i})$ is in $Stacks_{n-1}(\Sigma)$, there exists a ρ_i in $(Ops_{n-1}(\Sigma) \cup \mathcal{T}_{n-1})^*$ such that $\rho_i([a]_n, t(u_{\leq i}))$. Then by the previous lemma, $\mathrm{Reach}([a]_n, x)$ is true.

Conversely, suppose that $\mathrm{Reach}([a]_n, x)$ is true. By Lemma 7, we therefore have $x = [\mathrm{push}_{i_0 d_0}(x_0), \mathrm{push}_{i_1 d_1}(x_1), \cdots, \mathrm{push}_{i_{m-1} d_{m-1}}(x_{m-1}), x_m]_n$, where for every j there exists a $\rho_j \in (Ops_{n-1}(\Sigma) \cup \mathcal{T}_{n-1})^*$ such that $x_j = \rho_j([a]_n)$. Then, for every j, $x_j \in Stacks_{n-1}(\Sigma)$.

We take a tree domain U such that $d_0 \cdots d_{m-1} \in U$. We define a tree t of domain U such that for every j, $t(d_0 \cdots d_j) = x_{j+1}$, $t(\varepsilon) = x_0$, every node $d_0 \cdots d_j$ has i_{j+1} sons, the node ε has i_0 sons, and for every $u \in U$ which is not a $d_0 \cdots d_j$, $t(u) = [a]_n$. Then we have $x = \mathrm{Code}(t, d_0 \cdots d_{m-1})$. $\square$

C.2 The formula δ

We now define $\delta(X) = \mathrm{OnlyLeaves}(X)) \wedge \mathrm{TreeDom}(X) \wedge \mathrm{UniqueLabel}(X)$ with

$$\begin{aligned}
\mathrm{OnlyLeaves}(X) &= \forall x, x \in X \Rightarrow \mathrm{Reach}([a]_n, x) \\
\mathrm{TreeDom}(X) &= \forall x, y, z((x \in X \wedge \psi_{\mathrm{copy}_n^2,2}(y, z) \wedge \mathrm{Reach}(z, x)) \Rightarrow \\
&\quad \exists r, z'(r \in X \wedge \psi_{\mathrm{copy}_n^2,1}(y, z') \wedge \mathrm{Reach}(z', r))) \wedge \\
&\quad ((x \in X \wedge \psi_{\mathrm{copy}_n^2,1}(y, z) \wedge \mathrm{Reach}(z, x)) \Rightarrow \\
&\quad \exists r, z'(r \in X \wedge \psi_{\mathrm{copy}_n^2,2}(y, z') \wedge \mathrm{Reach}(z', r))) \\
\mathrm{UniqueLabel}(X) &= \forall x, y, (x \neq y \wedge x \in X \wedge y \in X) \Rightarrow \\
&\quad (\exists z, z', z'', \psi_{\mathrm{copy}_n^2,1}(z, z') \wedge \psi_{\mathrm{copy}_n^2,2}(z, z'') \wedge \\
&\quad ((\mathrm{Reach}(z', x) \wedge \mathrm{Reach}(z'', y)) \vee (\mathrm{Reach}(z'', x) \wedge \mathrm{Reach}(z', y))))
\end{aligned}$$

where a is a fixed letter of Σ.

Formula OnlyLeaves ensures that an element x in X encodes a node in some stack tree. TreeDom ensures that the prefix closure of the set of words $d_0 \cdots d_{m-1}$ such that

$$[\mathrm{push}_{i_0 d_0}(x_0)), \mathrm{push}_{i_1 d_1}(x_1), \cdots, \mathrm{push}_{i_{m-1} d_{m-1}}(x_{m-1}), x_m]_n \in X$$

is a valid domain of a tree, and that the set of words $i_0 \cdots i_{m-1}$ is included in this set (in other words, that the arity announced by the i_j is respected). an Finally UniqueLabel ensures that for any two elements

$$x = [\mathrm{push}_{i_0 d_0}(x_0)), \mathrm{push}_{i_1 d_1}(x_1), \cdots, \mathrm{push}_{i_{m-1} d_{m-1}}(x_{m-1}), x_m]_n$$
$$\text{and } y = [\mathrm{push}_{i'_0 d'_0}(y_0)), \mathrm{push}_{i'_1 d'_1}(y_1), \cdots, \mathrm{push}_{i'_{m-1} d'_{m'-1}}(y_{m'-1}), y_{m'}]_n$$

of X, there exists an index $1 \leq j \leq \min(m, m')$ such that for every $k < j$, $x_k = y_k$, $i_k = i'_k$ and $d_k = d'_k$, $x_j = y_j$, $i_j = i'_j$ and $d_j \neq d'_j$, i.e. for any two elements, the $(n-1)$-stacks labelling common ancestors are equal, and x and y cannot encode the same leaf (as $d_0 \cdots d_{m-1} \neq d'_0 \cdots d'_{m'-1}$). Moreover, it also prevents x to code a node on the path from the root to the node coded by y.

Lemma 8. $\forall X \subseteq Stacks_n(\Sigma \cup \{1,2\})$, $\delta(X) \iff \exists t \in ST_n, X = X_t$
where X ranges only over finite sets of $Stacks_n(\Sigma \cup \{1,2\})$.

Proof. We first show that for every n-stack tree t, $\delta(X_t)$ holds over $\Delta^n_{\Sigma \cup \{1,2\}}$. By definition, for every $x \in X_t$, $\exists u \in fr(t), x = \mathrm{Code}(t, u)$, and then $\mathrm{Reach}([a]_n, x)$ holds (by Corollary 2). Thus OnlyLeaves holds.

Let us take $x \in X_t$ such that $x = \mathrm{Code}(t, u)$ with $u = u_0 \cdots u_i 2 u_{i+2} \cdots u_{|u|}$. As t is a tree, $u_0 \cdots u_i 2 \in dom(t)$ and so is $u_0 \cdots u_i 1$. Then, there exists $v \in fr(t)$ such that $\forall j \leq i, v_j = u_j$, $v_{i+1} = 1$, and $\mathrm{Code}(t, v) \in X_t$. Let us now take $x \in X_t$ such that $x = \mathrm{Code}(t, u)$ with $u = u_0 \cdots u_i 1 u_{i+2} \cdots u_{|u|}$ and $\#(u_0 \cdots u_i 1) = 2$, then $u_0 \cdots u_i 2$ is in $dom(t)$ and there exists $v \in fr(t)$ such that $\forall j \leq i, v_j = u_j$, $v_{i+1} = 2$ and $\mathrm{Code}(t, v) \in X_t$. Thus TreeDom holds.

Let x and y in X_t such that $x \neq y$, $x = \mathrm{Code}(t, u)$ and $y = \mathrm{Code}(t, v)$, and let i be the smallest index such that $u_i \neq v_i$. Suppose that $u_i = 1$ and $v_i = 2$ (the other case is symmetric). We call $z = \mathrm{Code}(t, u_0 \cdots u_{i-1})$, and take z' and z'' such that $\psi_{\mathrm{copy}^2_n,1}(z, z')$ and $\psi_{\mathrm{copy}^2_n,2}(z, z'')$. We have then $\mathrm{Reach}(z', x)$ and $\mathrm{Reach}(z'', y)$. And thus UniqueLabel holds. Therefore, for every stack tree t, $\delta(X_t)$ holds.

Let us now show that for every $X \subseteq Stacks_n(\Sigma \cup \{1,2\})$ such that $\delta(X)$ holds, there exists $t \in ST_n$, such that $X = X_t$. As OnlyLeaves holds, for every $x \in X$,

$$x = [\mathrm{push}_{i_0 u_0}(x_0), \mathrm{push}_{i_1 u_1}(x_1), \cdots, \mathrm{push}_{i_{k-1} u_{k-1}}(x_{k-1}), x_k]_{n-1}$$

with, for all j, $x_j \in Stacks_{n-1}$, $i_j \in \{1,2\}$ and $u_j \leq i_j$. In the following, we denote by u^x the word $u_0 \cdots u_{k-1}$ for a given x, and by $U = \{u \mid \exists x \in X, u \sqsubseteq u^x\}$. U is closed under prefixes. As TreeDom holds, for all u, if $u2$ is in U, then $u1$ is in U as well. Therefore U is the domain of a tree. Moreover, if there is a x such that $u1 \sqsubseteq u^x$ and $i_{|u|} = 2$, then TreeDom ensures that there is y such that $u2 \sqsubseteq u^y$ and thus $u2 \in U$. As UniqueLabel holds, for every x and y two distinct elements of X, there exists j such that for all $k < j$ we have $u^x_k = u^y_k$, and $u^x_j \neq u^y_j$. Then, for all $k \leq j$, we have $x_k = y_k$ and $i_k = i'_k$. Thus, for every $u \in U$, we can define σ_u such that for every x such that $u \sqsubseteq u^x$, $x_{|u|} = \sigma_u$, and the number of sons of each node is consistent with the coding.

Consider the tree t of domain U such that for all $u \in U$, $t(u) = \sigma_u$. We have $X = X_t$, which concludes the proof. $\square$

C.3 The formula Ψ_D associated with an operation

We now take an operation D which we suppose to be reduced, for the sake of simplicity (but we could do so for a non reduced operation, and for any operation, there exists a reduced operation with tests defining the same relation, from the two previous appendices). We define inductively ψ_D as follow:

- $\Psi_\square(X,Y) = (X = Y)$
- $\Psi_{(F \cdot_{1,1} D_\theta) \cdot_{1,1} G}(X,Y) = \exists, z, z', Z, X', Y', z \in Z \wedge X \backslash X' = Y \backslash Y' = Z \backslash \{z\} \wedge \psi_\theta(z,z') \wedge \Psi_F(X,Z) \wedge \Psi_G(Z \cup \{z'\} \backslash \{z\}, Y)$, for $\theta \in Ops_{n-1} \cup \mathcal{T}_n$
- $\Psi_{(F \cdot_{1,1} D_{\text{copy}^1_n}) \cdot_{1,1} G}(X,Y) = \exists z, z', Z, X', Y', z \in Z \wedge X \backslash X' = Y \backslash Y' = Z \backslash \{z\} \wedge \psi_{\text{copy}^1_n,1}(z,z') \wedge \Psi_F(X,Z) \wedge \Psi_G(Z \cup \{z'\} \backslash \{z\}, Y)$
- $\Psi_{(F \cdot_{1,1} D_{\overline{\text{copy}^1_n}}) \cdot_{1,1} G}(X,Y) = \exists z, z', Z, X', Y', z \in Z \wedge X \backslash X' = Y \backslash Y' = Z \backslash \{z\} \wedge \psi_{\text{copy}^1_n,1}(z',z) \wedge \Psi_F(X,Z) \wedge \Psi_G(Z \cup \{z'\} \backslash \{z\}, Y)$
- $\Psi_{((F \cdot_{1,1} D_{\text{copy}^2_n}) \cdot_{1,2} H) \cdot_{1,1} G}(X,Y) = \exists z, z', z'', Z, Z', X', Y', z \in Z \wedge X \backslash X' = Y \backslash Y' = Z \backslash \{z\} \wedge \psi_{\text{copy}^1_n,2}(z,z') \wedge \psi_{\text{copy}^2_n,2}(z,z'') \wedge \Psi_F(X,Z) \wedge \Psi_G(Z \cup \{z', z''\} \backslash \{z\}, Z') \wedge z'' \in Z' \wedge z' \notin Z' \wedge \Psi_H(Z', Y)$
- $\Psi_{(F \cdot_{1,1} (G \cdot_{1,2} D_{\overline{\text{copy}^1_n}})) \cdot_{1,1} H}(X,Y) = \exists z, z', z'', Z, Z', X', Y', z \in Z \wedge z' \in Z \wedge z \in Z' \wedge z' \notin Z' \wedge X \backslash X' = Y \backslash Y' = Z \backslash \{z, z'\} \wedge \psi_{\text{copy}^2_n,1}(z'', z) \wedge \psi_{\text{copy}^2_n,2}(z'', z') \wedge \Psi_F(X, Z') \wedge \Psi_G(Z', Z) \wedge \Psi_G(Z \cup \{z''\} \backslash \{z, z'\}, Y)$

As D is a finite DAG, every ψ_D is a finite formula, and is thus a monadic formula.

This formula is true if its two arguments are related by $\mathcal{R}_D$.

Proposition 10. *Given two stack trees s, t and an operation D, $t \in D(t)$ if and only if $\Psi_D(X_s, X_t)$ is true.*

Proof. We show it by induction on the structure of D:

- If $D = \square$, $\Psi_D(X_s, X_t)$ if and only if $X_s = X_t$, which is true if and only if $s = t$.
- $D = (F \cdot_{1,1} D_\theta) \cdot_{1,1} G$, with $\theta \in Ops_{n-1} \cup \mathcal{T}_n$. Suppose $t \in D(s)$, there exists i such that $t = D_{(i)}(t)$. By definition, $t = G_{(i)}(\theta_{(i)}(F_{(i)}(s)))$. We call $r = F_{(i)}(s)$. By induction hypothesis, we have $\Psi_F(X_s, X_r)$. By definition, we have, for all $j < i$, $\text{Code}(s, u_j) = \text{Code}(r, u_j)$, and for all $j > i$, $\text{Code}(s, u_{j+|I_F|-1}) = \text{Code}(r, u_j)$, thus $X_s \backslash \{\text{Code}(s, u_j) \mid i \leq j \leq |I_F|-1\} = X_r \backslash \{\text{Code}(r, u_i)\}$. We call $r' = \theta_{(i)}(r)$. We have $X_{r'} = X_r \backslash \{\text{Code}(r, u_i)\} \cup \{\theta(\text{Code}(r, u_i))\}$. And by definition, we have $\psi_\theta(\text{Code}(r, u_i), \theta(\text{Code}(r, u_i)))$. We have $t = G_{(i)}(r')$, thus, by induction hypothesis, $\Psi_G(X_{r'}, X_t)$ is true. Moreover, by definition, $X_t \backslash \{\text{Code}(t, u_j) \mid i \leq j \leq |O_G| - 1\} = X_{r'} \backslash \{\text{Code}(r', u_i)\} = X_r \backslash \{\text{Code}(r, u_i)\}$. Thus, $\Psi_D(X_s, X_t)$ is true, with $Z = X_r$, $z = \text{Code}(r, u_i)$, $z' = \text{Code}(r', u_i)$, $X' = \{\text{Code}(s, u_j) \mid i \leq j \leq |I_D| - 1\}$ and $Y' = \{\text{Code}(t, u_j) \mid i \leq j \leq |O_D| - 1\}$.

Suppose that $\Psi_D(X_s, X_t)$ is true. We call r the tree such that $X_r = Z$. By induction hypothesis, we have $r \in F(s)$. Moreover, we have $z = \mathrm{Code}(r, u_i)$ such that $X_r \backslash \{z\} = X_s \backslash X'$. Thus, by definition, $r = F_{(i)}(s)$, and $X' = \{\mathrm{Code}(s, u_j) \mid i \leq |I_F| - 1\}$. We have $z' = \theta(z)$, as $\psi_\theta(z, z')$ is true. We call $r' = \theta_{(i)}(r)$, and we have $X_{r'} = X_r \backslash \{z\} \cup \{z'\}$. As we have $\Psi_G(X_{r'}, Y)$, by induction, we have $t \in G(r')$. As we moreover have $Y \backslash Y' = Z \backslash \{z\}$, we thus have $t = G_{(i)}(r')$. Thus, we have $t = G_{(i)}(\theta_{(i)}(F_{(i)}(s))) = D_{(i)}(s)$.

The other cases are similar and left to the reader.

C.4 The formula ϕ associated with an automaton

Let us now explain $\phi(X, Y)$, which can be written as $\exists Z_{q_1}, \cdots, Z_{q_{|Q|}}, \phi'(X, Y, \boldsymbol{Z})$ with $\phi'(X, Y, \boldsymbol{Z}) = \mathrm{Init}(X, Y, \boldsymbol{Z}) \wedge \mathrm{Diff}(\boldsymbol{Z}) \wedge \mathrm{Trans}(\boldsymbol{Z})$. We detail each of the three subformulas Init, Diff and Trans below:

$$\mathrm{Init}(X, Y, \boldsymbol{Z}) = (\bigcup_{q_i \in I} Z_{q_i}) \subseteq X \wedge (\bigcup_{q_i \in F} Z_{q_i}) \subseteq Y \wedge X \setminus (\bigcup_{q_i \in I} Z_{q_i}) = Y \setminus (\bigcup_{q_i \in F} Z_{q_i})$$

This formula is here to ensure that only leaves of X are labelled by initial states, only leaves of Y are labelled by final states and outside of their labelled leaves, X and Y are equal (i.e. not modified).

$$\begin{aligned}\mathrm{Diff}(\boldsymbol{Z}) = \big(\bigwedge_{q,q' \in Q_{T,c}} Z_q \cap Z_{q'} = \emptyset\big) \wedge \big(\bigwedge_{q,q' \in Q_{C,c}} Z_q \cap Z_{q'} = \emptyset\big) \\ \wedge \big(\bigwedge_{q,q' \in Q_{T,d}} Z_q \cap Z_{q'} = \emptyset\big) \wedge \big(\bigwedge_{q,q' \in Q_{C,d}} Z_q \cap Z_{q'} = \emptyset\big)\end{aligned}$$

This formula is here to ensure that a given stack (and thus a given leaf in a tree of the run) is labelled by at most a state of each subpart of Q: $Q_{T,d}, Q_{C,d}, Q_{T,c}, Q_{C,c}$. So if we have a non deterministic choice to do we will only choose one possibility.

$$\mathrm{Trans}(\boldsymbol{Z}) = \forall s, \bigwedge_{q \in Q} ((s \in Z_q) \Rightarrow (\bigvee_{K \in \Delta} \mathrm{Trans}_K(s, \boldsymbol{Z}) \vee \rho_q))$$

where ρ_q is true if and only if q is a final state, and

$$\begin{aligned}
\mathrm{Trans}_{(q,\mathrm{copy}^1_n,q')}(s, \boldsymbol{Z}) &= \exists t, \psi_{\mathrm{copy}^1_n,1}(s, t) \wedge t \in Z_{q'}, \\
\mathrm{Trans}_{(q,\overline{\mathrm{copy}}^1_n,q')}(s, \boldsymbol{Z}) &= \exists t, \psi_{\mathrm{copy}^1_n,1}(t, s) \wedge t \in Z_{q'}, \\
\mathrm{Trans}_{(q,\theta,q')}(s, \boldsymbol{Z}) &= \exists t, \psi_\theta(s, t) \wedge t \in Z_{q'}, \text{for } \theta \in Ops_{n-1} \cup \mathcal{T}_{n-1}, \\
\mathrm{Trans}_{(q,(q',q''))}(s, \boldsymbol{Z}) &= \exists t, t', \psi_{\mathrm{copy}^2_n,1}(s, t) \wedge \psi_{\mathrm{copy}^2_n,2}(s, t') \wedge t \in Z_{q'} \wedge t' \in Z_{q''}, \\
\mathrm{Trans}_{((q,q'),q'')}(s, \boldsymbol{Z}) &= \exists t, t', \psi_{\mathrm{copy}^2_n,1}(t', s) \wedge \psi_{\mathrm{copy}^2_n,2}(t', t) \wedge t \in Z_{q'} \wedge t' \in Z_{q''}, \\
\mathrm{Trans}_{((q',q),q'')}(s, \boldsymbol{Z}) &= \exists t, t', \psi_{\mathrm{copy}^2_n,1}(t', t) \wedge \psi_{\mathrm{copy}^2_n,2}(t', s) \wedge t \in Z_{q'} \wedge t' \in Z_{q''}.
\end{aligned}$$

This formula ensures that the labelling respects the rules of the automaton, and that for every stack labelled by q, if there is a rule starting by q, there is at least a stack which is the result of the stack by one of those rules. And also that it is possible for a final state to have no successor.

Proposition 11. *Given s, t two stack trees, $\phi(s,t)$ if and only if there are some operations $D_1, \cdots, D_k$ recognised by A such that t is obtained by applying $D_1, \cdots, D_k$ at disjoint positions of s.*

Proof. First suppose there exist such $D_1, \cdots, D_k$. We construct a labelling of $Stacks_n(\Sigma \cup \{1,2\})$ which satisfies $\phi(X_s, X_t)$. We take a labelling of the D_i by A. We will label the $Stacks_n$ according to this labelling. If we obtain a tree t' at any step in the run of the application of D_i to s, we label $\mathrm{Code}(t', u)$ by the labelling of the node of D_i appended to the leaf at position u of t'. Notice that this does not depend on the order we apply the D_i to s nor the order of the leaves we choose to apply the operations first.

We suppose that $t = D_{k i_k}(\cdots D_{1 i_1}(s) \cdots)$. Given a node x of an D_i, we call $l(x)$ its labelling.

Formally, we define the labelling inductively: the $(D_1, i_1, s_1), \cdots, (D_k, i_k, s_k)$ labelling of $Stacks_n(\Sigma \cup \{1,2\})$ is the following.

- The $\emptyset$ labelling is the empty labelling.
- The $(D_1, i_1, s_1), \cdots, (D_k, i_k, s_k)$ labelling is the union of the (D_1, i_1, s_1) labelling and the $(D_2, i_2, s_2), \cdots, (D_k, i_k, s_k)$ labelling.
- The $(\square, i, s)$ labelling is $\{\mathrm{Code}(s, u_i) \to l(x)\}$, where u_i is the i^{th} leaf of s and x is the unique node of $\square$.
- The $(F_1 \cdot_{1,1} D_\theta) \cdot_{1,1} F_2, i, s)$ labelling is the $(F_1, i, s), (F_2, i, \theta_{(i)}(F_{1(i)}(s)))$ labelling.
- The $((((F_1 \cdot_{1,1} D_{\mathrm{copy}^2_n}) \cdot_{2,1} F_3) \cdot_{1,1} F_2), i, s)$ labelling is the (F_1, i, s), $(F_2, i, \mathrm{copy}^2_{n(i)}(F_{1(i)}(s)))$, $(F_3, i+1, \mathrm{copy}^2_{n(i)}(F_{1(i)}(s)))$ labelling.
- The $((F_1 \cdot_{1,1} (F_2 \cdot_{2,1} \overline{\mathrm{copy}}^2_n)) \cdot_{1,1} F_3, i, s)$ labelling is the $(F_1, i, s), (F_2, i + |I_{F_1}|, s)$, $(F_3, i, \overline{\mathrm{copy}}^2_{n(i)}(F_{2(i+1)}(F_{1(i)}(s))))$ labelling.

Observe that this process terminates, as the sum of the edges and the nodes of all the DAGs strictly diminishes at every step.

We take $\boldsymbol{Z}$ the $(D_1, i_1, s), \cdots, (D_k, i_k, s)$ labelling of $Stacks_n(\Sigma \cup \{1,2\})$.

Lemma 9. *The labelling previously defined $\boldsymbol{Z}$ satisfies $\phi'(X_s, X_t, \boldsymbol{Z})$.*

Proof. Let us first cite a technical lemma which comes directly from the definition of the labelling:

Lemma 10. *Given a reduced operation D, a labelling of D, ρ_D, a stack tree t, a $i \in \mathbb{N}$ and a $j \leq |I_D|$, the label of $\mathrm{Code}(t, u_{i+j-1})$ (where u_i is the i^{th} leaf of t) in the (D, i, t) labelling is $\rho_D(x_j)$ (where x_j is the j^{th} input node of D).*

For the sake of simplicity, let us consider for this proof that D is a reduced operation (if it is a set of reduced operations, the proof is the same for every operations).

First, let us prove that Init is satisfied. From the previous lemma, all nodes of X_s are labelled with the labels of input nodes of D (or not labelled), thus they are labelled by initial states (as we considered an accepting labelling of D). Furthermore, as the automaton is distinguished, only these one can be labelled by initial states. Similarly, the nodes of X_t, and only them are labelled by final states (or not labelled).

We now show that Trans is satisfied. Let us suppose that a $\mathrm{Code}(t', u_i)$ is labelled by a q. By construction of the labelling, it has been obtained by a $(\square, i, t')$ labelling. If q is final, then we have nothing to verify, as ρ_q is true. If not, the node x labelled by q which is the unique node of the $\square$ which labelled $\mathrm{Code}(t', u_i)$ by q has at least one son in D. Suppose, for instance that $D = (F_1 \cdot_{1,1} D_\theta) \cdot_{1,1} F_2$ such that x is the output node of F_1. We call y the input node of F_2. As D is recognised by A, it is labelled by a q' such that $(q, \theta, q') \in \Delta_A$. By construction, we take the $(F_1, i, s), (F_2, i, \theta_{(i)}(t'))$ labelling, with $t' = F_{2(i)}(s)$. Thus we have $\mathrm{Code}(\theta_{(i)}(t'), u_i)$ labelled by q' (from Lemma 10), and thus $\mathrm{Trans}_{(q,\theta,q')}(\mathrm{Code}(t', u_i), \boldsymbol{Z})$ is true, as $\psi_\theta(\mathrm{Code}(t', u_i), \mathrm{Code}(\theta_{(i)}(t'), u_i)$ is true.

The other possible cases for decomposing D ($D = (((F_1 \cdot_{1,1} D_{\mathrm{copy}^1_n}) \cdot_{2,1} F_3) \cdot_{1,1} F_2$ or $D = ((F_1 \cdot_{1,1} (F_2 \cdot_{2,1} \overline{\mathrm{copy}}^2_n)) \cdot_{1,1} F_3)$ are very similar and are thus left to the reader. Observe that D may not be decomposable at the node x, in which case we decompose D and consider the part containing x until we can decompose the DAG at x, where the argument is the same.

Let us now prove that the labelling satisfies Diff. Given $q, q' \in Q_{C,d}$, suppose that there is a $\mathrm{Code}(t', u_i)$ which is labelled by q and q'. By construction, this labelling is obtained by a $(F_1, i, t'_1), (F_2, i, t'_2)$ labelling, where F_1 and F_2 are both $\square$, and $t'_1(u_i) = t'_1(u_i)$. We call x (resp. y) the unique node of F_1 (resp. F_2). x is labelled by q and y by q'.

Suppose that D can be decomposed as $(G \cdot_{1,1} D_\theta) \cdot_{1,1} H$ (or $((G \cdot_{1,1} D_{\mathrm{copy}^2_n}) \cdot_{2,1} K) \cdot_{1,1} H$, or $((G \cdot_{1,1} (H \cdot_{1,2} D_{\overline{\mathrm{copy}}^2_n}) \cdot_{1,1} K)$ such that y is the output node of G (if not, decompose D until you can obtain such a decomposition). Then, suppose you can decompose $G = G_1 \cdot_{1,1} D_\theta \cdot_{1,1} G_2$ (or $((G_1 \cdot_{1,1} (G_3 \cdot_{1,2} D_{\overline{\mathrm{copy}}^2_n}) \cdot_{1,1} G_2$. As we are considering states of $Q_{C,d}$, there is no other possible case) such that x is the input node of G_2. Thus, we have by construction $G_2(\mathrm{Code}(t', u_i)) = \mathrm{Code}(t', u_i)$. So G_2 defines a relation contained in the identity. As it is a part of D and thus labelled by states of A, with q and q' in $Q_{C,d}$, there is no copy^j_n nor $\overline{\mathrm{copy}}^j_n$ transitions in G_2. Moreover, as q and q' are in $Q_{C,d}$, G_2 is not a single test transition. Then it is a sequence of elements of $Ops_{n-1} \cup \mathcal{T}_{n-1}$ defining a relation included into the identity. As A is normalised, this is impossible, and then $\mathrm{Code}(t', u_i)$ cannot be labelled by both q and q'.

Taking two states in the other subsets of Q yields the same contradiction with few modifications and are thus left to the reader.

Then, as all its sub-formulæ are true, $\phi'(X_s, X_t, \boldsymbol{Z})$ is true with the described labelling $\boldsymbol{Z}$. And then $\phi(X_s, X_t)$ is true. $\square$

Suppose now that $\phi(X_s, X_t)$ is satisfied. We take a minimal labelling $\boldsymbol{Z}$ that satisfies the formula $\phi'(X_s, X_t, \boldsymbol{Z})$. We construct the following graph D :

$$
\begin{aligned}
V_D &= \{(x,q) \mid x \in \mathit{Stacks}_n(\Sigma \cup \{1,2\}) \wedge x \in Z_q\} \\
E_D &= \{((x,q),\theta,(y,q')) \mid (\exists\theta,(q,\theta,q') \in \Delta \wedge \psi_\theta(x,y))\} \\
&\cup \{((x,q),1,(y,q')),((x,q),2,(z,q'')) \mid (q,(q',q'')) \in \Delta \\
&\quad \wedge \psi_{\mathrm{copy}^2_n,1}(x,y) \wedge \psi_{\mathrm{copy}^2_n,2}(x,z)\} \\
&\cup \{((x,q),\bar{1},(z,q'')),((y,q'),\bar{2},(z,q'')) \mid ((q,q'),q'') \in \Delta \\
&\quad \wedge \psi_{\mathrm{copy}^2_n,1}(z,x) \wedge \psi_{\mathrm{copy}^2_n,2}(z,y)\} \\
&\cup \{((x,q),1,(y,q')) \mid (q,\mathrm{copy}^1_n,q') \in \Delta \wedge \psi_{\mathrm{copy}^1_n,1}(x,y)\} \\
&\cup \{((x,q),\bar{1},(y,q')) \mid (q,\overline{\mathrm{copy}}^1_n,q') \in \Delta \wedge \psi_{\mathrm{copy}^1_n,1}(y,x)\}
\end{aligned}
$$

Lemma 11. *D is a disjoint union of operations $D_1, \cdots, D_k$.*

Proof. Suppose that D is not a DAG, then there exists $(x,q) \in V$ such that $(x,q) \xrightarrow{+} (x,q)$, then there exists a sequence of operations in A_d (for A_c it is symmetric, and there is no transition from A_c to A_d, thus a cycle cannot have states of the both parts) which is the identity (and thus it is an sequence of operations of $\mathit{Ops}_{n-1} \cup \mathcal{T}_{n-1}$). As A_d is normalised, it is not possible to have such a sequence. Then, there is no cycle in D which is therefore a DAG.

By definition of E_D, it is labelled by $\mathit{Ops}_{n-1} \cup \mathcal{T}_{n-1} \cup \{1, \bar{1}, 2, \bar{2}\}$.

We choose an D_i. Suppose that it is not an operation. Thus, there exists a node (x,q) of D_i such that D_i cannot be decomposed at this node (i.e, in the inducted decomposition, there will be no case which can be applied to cut either D_i or one of its subDAG to obtain (x,q) as the output node of a subDAG obtained (or the input node). Let us consider the following cases for the neighbourhood of (x,q):

- (x,q) has a unique son (y,q'), which has no other father such that $(x,q) \xrightarrow{2} (y,q')$. By definition of Trans, we have that $\psi_{\mathrm{copy}^2_n,2}(x,y)$, and thus we have a $(q,(q'',q')) \in \Delta$ and a z such that $\psi_{\mathrm{copy}^2_n,1}(x,z)$ which is in $Z_{q''}$. This contradicts that (x,q) has a unique son in D_i. If $(x,q) \xrightarrow{\bar{2}} (y,q')$, the case is similar. For every other $\theta \in \mathit{Ops}_{n-1} \cup \mathcal{T}_{n-1} \cup \{1,\bar{1}\}$, we can decompose the subDAG $\{(x,q) \xrightarrow{\theta} (y,q')\}$ as $(\Box \cdot_{1,1} D_\theta) \cdot_{1,1} \Box$.
- Suppose that (x,q) has at least three sons $(y_1,q_1),(y_2,q_2),(y_3,q_3)$. There is no subformula of Trans which impose to label three nodes which can be obtained from x, so this contradicts the minimality of the labelling.
  For a similar reason, (x,q) has at most two fathers.
- Suppose that (x,q) has two sons (y_1,q_1) and (y_2,q_2). By definition of Trans and by minimality, we have that $\psi_{\mathrm{copy}^2_n,1}(x,y_1)$, $\psi_{\mathrm{copy}^2_n,2}(x,y_2)$, and $(q,(q_1,q_2)) \in \Delta$ (otherwise, the labelling would not be minimal, as it is the only subformula imposing to label two sons of a node). Thus we have $(x,q) \xrightarrow{1} (y_1,q_1)$ and $(x,q) \xrightarrow{2} (y_2,q_2)$. By minimality again, (y_1,q_1) and (y_2,q_2) have no other father than (x,q). In this case, the subDAG $\{(x,q) \xrightarrow{1} (y_1,q_1),(x,q) \xrightarrow{2} (y_2,q_2)\}$ can be decomposed as $((\Box \cdot_{1,1} D_{\mathrm{copy}^2_n}) \cdot_{2,1} \Box) \cdot_{1,1} \Box$.

– Suppose that (x, q) has a unique son (y_1, q_1) which has an other father (y_2, q_2). By definition of Trans and by minimality of the labelling, we have that $\psi_{\mathrm{copy}^2_n,1}(y_1, x)$, $\psi_{\mathrm{copy}^2_n,2}(y_1, y_2)$, and $((q, q_2), q_1) \in \Delta$. Thus we have $(x, q) \xrightarrow{\bar{1}} (y_1, q_1)$ and $(y_2, q_2) \xrightarrow{\bar{2}} (y_1, q_1)$. By minimality again, (y_2, q_2) has no other son than (y_1, q_1). In this case, the subDAG $\{(x, q) \xrightarrow{\bar{1}} (y_1, q_1), (y_2, q_2) \xrightarrow{\bar{2}} (y_1, q_1)\}$ can be decomposed as $(\Box \cdot_{1,1} (\Box \cdot_{1,2} D_{\overline{\mathrm{copy}^2_n}})) \cdot_{1,1} \Box$.

In all the cases we considered, or the case is impossible, or the DAG is decomposable at the node (x, q). Thus, the DAG D_i is always decomposable and is thus an operation. $\Box$

Lemma 12. *Each D_i is recognised by A*

Proof. By construction, for every node (x, q), if $x \in X_s$, q is an initial state (because init is satisfied), and (x, q) is then an input node, as A is distinguished. And as init is satisfied, only these nodes are labelled by initial states.

Also, for every node (x, q), if $x \in X_t$, q is a final state (because init is satisfied) and (x, q) is then an output node, as A is distinguished. And as init in satisfied, only these nodes are labelled by final states.

By construction, the edges are always transitions present in Δ, and then we label each node (x, q) by q.

As the formula Trans is satisfied, we have that given any node (x, q), either q is final (and then (x, q) is an output node), or there exists one of the following:

– a node (y, q') and θ such that $\psi_\theta(x, y)$ and $(q, \theta, q') \in \Delta$
– two nodes (y, q') and (z, q'') such that $\psi_{\mathrm{copy}^2_n,1}(x, y)$, $\psi_{\mathrm{copy}^2_n,2}(x, z)$ and $(q, (q', q'')) \in \Delta$
– two nodes (y, q') and (z, q'') such that $\psi_{\mathrm{copy}^2_n,1}(z, x)$, $\psi_{\mathrm{copy}^2_n,2}(z, y)$ and $((q, q'), q'') \in \Delta$

Then, only nodes (x, q) with q final are childless and are those labelled with final states. As well, only (x, q) with q initial are fatherless.

Then each D_i is recognised by A with this labelling. $\Box$

Lemma 13. *t is obtained by applying the D_i to disjoint positions of s.*

Proof. We show by induction that $t' = D_{(j)}(s)$ if and only if $X_{t'} = X_s \cup \{x \mid (x, q) \in O_D\} \backslash \{x \mid (x, q) \in I_D\}$:

– If $D = \Box$, it is true, as $X_{t'} = X_s$ and $t' = s$.
– If $D = (F \cdot_{1,1} D_\theta) \cdot_{1,1} G$, by induction hypothesis, we consider r such that $r = F_{(j)}(s)$, we then have $X_r = X_s \cup \{y\} \backslash \{x \mid (x, q) \in I_F\}$, where (y, q') is the only output node of F. By construction, the input node of G, (z, q'') is such that $\psi_\theta(y, z)$, and thus we have $r' = \theta_{(j)}(r)$ such that $X_{r'} = X_r \backslash \{y\} \cup \{z\}$. By induction hypothesis, we have $X_{t'} = X_{r'} \cup \{x \mid (x, q) \in O_G\} \backslash \{z\}$, as $t' = G_{(j)}(\theta_{(j)}(F_{(j)}(s))) = G_{(j)}(r')$. Thus, $X_{t'} = X_s \cup \{x \mid (x, q) \in O_G\} \backslash \{x \mid (x, q) \in I_F\} = X_s \cup \{x \mid (x, q) \in O_D\} \backslash \{x \mid (x, q) \in I_D\}$.

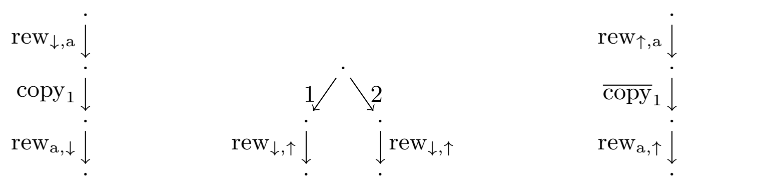
The other cases are similar and are thus left to the reader. It then suffices to construct this way successively $t_1 = D_{1(i_1)}(s)$, $t_2 = D_{2(i_2)}(t_1)$, etc, to obtain t and prove the lemma. $\square$

We have proved both directions: for every n-stack trees s and t, there exists a set of operations D_i recognised by A such that t is obtained by applying the D_i to disjoint positions of s if and only if $\phi(X_s, X_t)$. $\square$

We then have a monadic interpretation with finite sets (all sets are finite), and then, the graph has a decidable FO theory, which concludes the proof.

D Example of a language

We can see a rewriting graph as a language acceptor in a classical way by defining some initial and final states and labelling the edges. We present here an example of a language recognised by a stack tree rewriting system. The recognised language is $\{u \sqcup u \mid u \in \Sigma\}$. Fix an alphabet Σ and two special symbols $\uparrow$ and $\downarrow$. We consider $ST_2(\Sigma \cup \{\uparrow, \downarrow\})$. We now define a rewriting system R, whose rules are given in Fig. 8.



(a) P_a, for every $a \in \Sigma$ (b) *Dupl* (c) D_a, for every $a \in \Sigma$

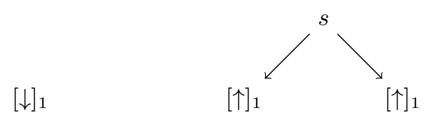
Fig. 8: The rules of the rewriting system

To recognise a language with this system, we have to fix an initial set of stack trees and a final set of stack trees. We will have a unique initial tree and a recognisable set of final trees. They are depicted on Fig. 9.

A word $w \in R^*$ is accepted by this rewriting system if there is a path from the initial tree to a final tree labelled by w. The trace language recognised is

$$\{P_{a_1} \cdots P_{a_n} \cdot Dupl \cdot ((D_{a_n} \cdots D_{a_1}) \sqcup (D_{a_n} \cdots D_{a_1})) \mid a_1, \cdots, a_n \in \Sigma\}.$$

Let us informally explain why. We start on the initial tree, which has only a leaf labelled by a stack whose topmost symbol is $\downarrow$. So we cannot apply a D_a to it. If we apply a P_a to it, we remain in the same situation, but we added an a to the stack labelling the unique node. So we can read a sequence $P_{a_1} \cdots P_{a_n}$. From this situation, we can also apply a *Dupl*, which yields a tree with three nodes



(a) The initial tree.

(b) A final tree. s is an arbitrary 1-stack

Fig. 9: The initial and final trees.

whose two leaves are labelled by $[a_1 \cdots a_n \uparrow]_1$, if we first read $P_{a_1} \cdots P_{a_n}$. From this new situation, we can only apply D_a rules. If the two leaves are labelled by $[b_1 \cdots b_m \uparrow]_1$ and $[c_1 \cdots c_\ell \uparrow]_1$, we can apply D_{b_m} or D_{c_ℓ}, yielding the same tree in which we removed b_m or c_ℓ from the adequate leaf. We can do this until a final tree remains. So, on each leaf, we will read $D_{a_n} \cdots D_{a_1}$ in this order, but we have no constraint on the order we will read these two sequences. So we effectively can read any word in $(D_{a_n} \cdots D_{a_1}) \sqcup\!\sqcup (D_{a_n} \cdots D_{a_1})$. And this is the only way to reach a final tree.

To obtain the language we announced at the start, we just have to define a labelling λ of each operation of R as follows: $\lambda(\mathit{Dupl}) = \varepsilon$, for every $a \in \Sigma$, $\lambda(P_a) = \varepsilon$ and $\lambda(D_a) = a$, and remark that if w is of the previous form, then $\lambda(w) = (a_1 \cdots a_n) \sqcup\!\sqcup (a_1 \cdots a_n)$, and we indeed recognise $\{u \sqcup\!\sqcup u \mid u \in \Sigma\}$.