Theory Comput Syst
DOI 10.1007/s00224-017-9769-6

Rewriting Higher-Order Stack Trees

Vincent Penelle[1,2]

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.

Keywords Automata theory · Rewriting · Infinite graphs · Stack trees

1 Introduction

Since Rabin's proof of the decidability of monadic second order logic (MSO) over the full infinite binary tree Δ_2 [18], there has been an effort to characterise increasingly general classes of structures with decidable MSO theories. This can be achieved for

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

✉ Vincent Penelle
vincent.penelle@u-pem.fr

1 Institute of Informatics and Mathematics, University of Warsaw, Ulica Banacha 2, 02-097, Warszawa, Poland

2 LIGM (CNRS UMR 8049), UPEM, CNRS, Université Paris-Est, F-77454 Marne-la-Vallée, France

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 [7, 9] 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 [15]. We will henceforth call this the *pushdown hierarchy* and the graphs at its n-th order *n-pushdown graphs* for simplicity.

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* [11], a generalisation of MSO interpretations mapping structures with a decidable weak-MSO theory (WMSO) 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 [11] 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 [13] to have decidable 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)*

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

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 FO[$\xrightarrow{*}$] theories. A preliminary version of this work has been presented in [17].

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 FO[$\xrightarrow{*}$] model checking over ground stack tree rewriting graphs. Finally, Section 7 presents some further perspectives.

2 Definitions and Notations

Relations Given an arbitrary set Γ, a binary relation over it is a subset of $\Gamma \times \Gamma$. A pair of elements (γ_1, γ_2) of Γ related by a relation R will be denoted as $(\gamma_1, \gamma_2) \in R$. Given two relations R_1 and R_2 over Γ, we denote by $R_1 \circ R_2 = \{(\gamma_1, \gamma_2) \mid \exists \gamma_3, (\gamma_1, \gamma_3) \in R_1 \wedge (\gamma_3, \gamma_2) \in R_2\}$. Notice that this notation is not the one we encounter in the case of function, but will be easier to consider in our case as it will be more consistent with the other notations encountered throughout this document.

Trees Given an arbitrary set Γ, an ordered Γ-labelled tree t of arity at most $d \in \mathbb{N}$ is a *partial function* from $\{1, \ldots, d\}^*$ to Γ 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 over $\{1, \ldots, d\}^*$. By abuse of notation, given a symbol $\alpha \in \Gamma$, we simply denote by α the tree $\{\epsilon \mapsto \alpha\}$ reduced to a unique α-labelled node. The frontier of t is the set $\mathrm{fr}(t) = \{u \in \mathrm{dom}(t) \mid u1 \notin \mathrm{dom}(t)\}$. Its elements are called the *leaves* of t. Trees will always be drawn in such a way that the left-to-right placement of leaves respects the lexicographic order. The set of finite trees labelled by Γ is denoted by $\mathcal{T}(\Gamma)$. 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 $w \in \{1, \cdots, d\}^*$, such that $v = uw$. 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, and $v_{\geq i}$ its suffix of length i. For any $u \in \mathrm{dom}(t)$, $t(u)$ is called the *label* of node u in t. 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 $t\mathbf{u}s_1, \ldots, s_k$ 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 $s_i(v)$.

Directed Graphs A *directed graph* G with edge labels in Σ is a pair (V_G, E_G) where V_G is a set of vertices and $E_G \subseteq (V_G \times \Sigma \times V_G)$ is a set of edges. Given two vertices x and y, we write $x \xrightarrow{a}_G y$ if $(x, a, y) \in E_G$, $x \to_G y$ if there exists $a \in \Sigma$ such that $x \xrightarrow{a}_G y$, and $x \xrightarrow{\Sigma'}_G y$ if there exists $a \in \Sigma'$ such that $x \xrightarrow{a}_G y$. There is a *directed path* in G from x to y labelled by $w = w_1 \dots w_k \in \Sigma^*$, written $x \xrightarrow{w}_G y$, if there are vertices $x_0, \dots, x_k$ such that $x = x_0$, $x_k = y$ and for all $i \in \{1, \cdots, k\}$, $x_{i-1} \xrightarrow{w_i}_G x_i$. 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 $(x, y) \in (\to_G \cup \to_G^{-1})^*$. 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 $\Box$. 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 Γ and Σ be finite alphabets. A Σ-labelled *ground tree rewriting system* (GTRS) is a finite set R of triples (ℓ, a, r) called *rewrite rules*, with ℓ and r finite Γ-labelled trees and $a \in \Sigma$ a label. The rewriting graph of R is $\mathcal{G}_R = (V, E)$, where $V = \mathcal{T}(\Gamma)$ and $E = \{(c(i)\ell, a, c(i)r) \mid (\ell, a, r) \in R,\ c \in \mathcal{T}(\Gamma),\ i \text{ is a leaf of } c\}$. The *rewriting relation* associated to R is $\to_R = \to_{\mathcal{G}_R}$, 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*.

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 order 0 and uses a different set of basic operations.

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

For the sake of simplicity, we will denote as $[\alpha]_n$ the n-stack containing only the symbol α, i.e. $[[\dots[\alpha]_1 \dots]_{n-1}]_n$.

Basic Stack Operations Given two letters $\alpha, \beta \in \Gamma$, we define the operation $\mathrm{rew}_{\alpha,\beta}$, and its associated binary relation over $Stacks_0$, $r_{\mathrm{rew}_{\alpha,\beta}} = \{(\alpha, \beta)\}$. For $n \geq 1$, the operation copy_n defines the relation

$$r_{\mathrm{copy}_n} = \{([s_1 \dots s_k]_n, [s_1 \dots s_k s_k]_n) \mid [s_1 \dots s_k]_n \in Stacks_n\}.$$

We finally consider the operation $\overline{\mathrm{copy}}_n$ which is the inverse of copy_n, in the sense that its associated relation is $r_{\overline{\mathrm{copy}}_n} = r^{-1}_{\mathrm{copy}_n}$.

For any order-ℓ operation θ, we extend the relation r_θ to any order $n > \ell$ by considering

$$r_\theta = \{([s_1 \dots s_{k-1} s_k]_n, [s_1 \dots s_{k-1} s'_k]_n) \mid (s_k, s'_k) \in r_\theta\}.$$

The set Ops_n of basic operations of order n is defined as: $Ops_0 = \{\mathrm{rew}_{\alpha,\beta} \mid \alpha, \beta \in \Gamma\}$, and for $n \geq 1$, $Ops_n = Ops_{n-1} \cup \{\mathrm{copy}_n, \overline{\mathrm{copy}}_n\}$.

We consider sequences of operation of Ops_n. Given a sequence of operations $\theta = \theta_1 \cdots \theta_k \in Ops_n^*$, we define its associated relation $r_\theta = r_{\theta_1} \circ \cdots \circ r_{\theta_k}$. We as well define $\overline{\theta} = \overline{\theta_k} \cdots \overline{\theta_1}$, where for every $\theta_i \in Ops_n$, we take $\overline{\theta_i}$ such that $r_{\overline{\theta_i}} = r^{-1}_{\theta_i}$, i.e. $\overline{\mathrm{rew}_{\alpha,\beta}} = \mathrm{rew}_{\beta}\alpha$ and $\overline{\overline{\mathrm{copy}}_n} = \mathrm{copy}_n$. Observe that we have $r_{\overline{\theta}} = r^{-1}_{\theta}$. Finally, observe that all relations defined by sequences of operations are functions. So, by abuse of notation, given a stack s and a sequence of operations θ, we may in the following denote by $\theta(s)$ the unique stack s' such that $(s, s') \in r_\theta$.

3.2 Stack Trees

We introduce the set $ST_n(\Gamma) = \mathcal{T}(Stacks_{n-1}(\Gamma))$ (or simply ST_n when Γ is understood) of *n-stack trees*. Observe that an n-stack tree of degree 1 is isomorphic to an n-stack, and that $ST_1 = \mathcal{T}(\Gamma)$. 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 present the basic operations we consider over n-stack trees. As we wish that our model extends stack operations, and namely that unary stack tree operations coincide with stack operations, we consider the extension of stack operations to stack trees, which will be applied to one of the leaves of the n-stack tree, and we define two new order-n operations allowing to respectively create and destroy several leaves. We first define the relation associated with a basic stack tree operation. As there are in general several positions of a stack tree where a given

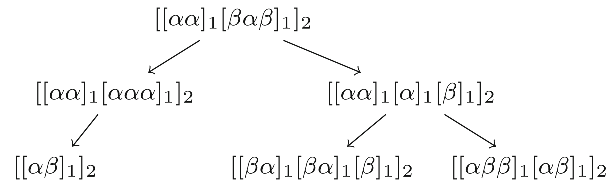

$[[\alpha\alpha]_1[\beta\alpha\beta]_1]_2$
$[[\alpha\alpha]_1[\alpha\alpha\alpha]_1]_2$
$[[\alpha\alpha]_1[\alpha]_1[\beta]_1]_2$
$[[\alpha\beta]_1]_2$
$[[\beta\alpha]_1[\beta\alpha]_1[\beta]_1]_2$
$[[\alpha\beta\beta]_1[\alpha\beta]_1]_2$

Fig. 1 A 3-stack tree

stack tree operation can be applied, we define the relation defined by the *localised application* of an operation to a given leaf of a stack tree, given by the index of that leaf in the lexicographic order over the leaves.

Definition 1 We consider the set of basic operations over ST_n, $TOps_n = Ops_{n-1} \cup \left\{\mathrm{copy}_n^k, \overline{\mathrm{copy}}_n^k \mid 1 \leq k \leq d\right\}$. Given an integer i, the relation defined by the application localised in i of a basic operation θ is defined as follows:

- $r_\theta^i = \{(t, t') \mid \mathrm{dom}(t) = \mathrm{dom}(t') \wedge \forall v \neq u_i, t(v) = t'(v) \wedge (t(u_i), t'(u_i)) \in r_\theta\}$, for $\theta \in Ops_{n-1}$.
- $r^i_{\mathrm{copy}_n^k} = \{(t, t') \mid t' = t \cup \{u_i j \mapsto t(u_i) \mid 1 \leq j \leq k\}$.
- $r^i_{\overline{\mathrm{copy}}_n^k} = \left(r^i_{\mathrm{copy}_n^k}\right)^{-1}$.

For $\theta \in TOps_n$, the relation defined by θ is thus $r_\theta = \bigcup_{i \in \mathbb{N}} r_\theta^i$, i.e. the union of all the possible localised applications of θ.

In the previous definition, we consider that $\mathrm{fr}(t) = \{u_1, \cdots, u_{|\mathrm{fr}(t)|}\}$, and that the leaves are ordered in respect to the lexicographic order over $\{1, \cdots, d\}^*$. If $i > |\mathrm{fr}(t)|$, there is no stack tree t' such that $(t, t') \in r_\theta^i$. 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.

3.3 Stack Tree Rewriting

As already mentioned, ST_1 is the set of trees labelled by Γ. In contrast with basic stack tree operations, a tree rewrite rule (ℓ, a, r) expresses the replacement of an arbitrarily large ground subtree ℓ of some tree $s = c[\ell]$ into r, yielding the tree $c[r]$. Contrarily 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 order 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 order-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 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.

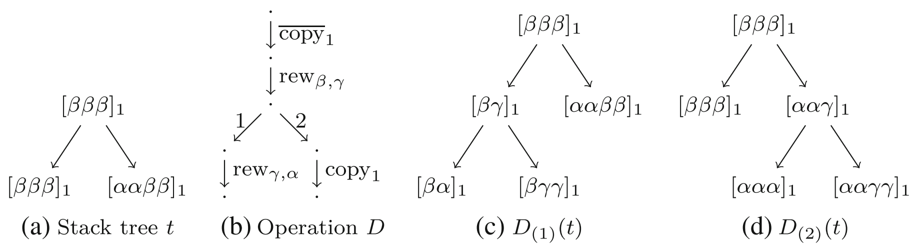


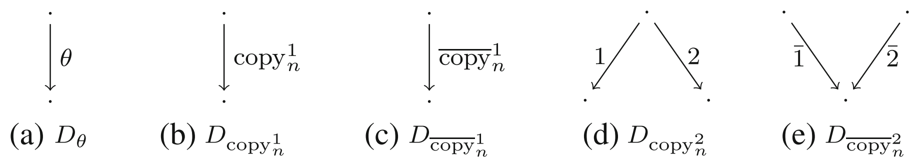
Fig. 2 The application of an operation D to a stack tree t

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 $i \in \{1, \cdots, \ell\}$ and $j \in \{1, \cdots, k'\}$ such that $\min(i, j) = 1$, 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 $\{(f(x), a, f(x')) \mid (x, a, x') \in E_D\}$, and $f(x) = b_{i+m}$ if $x = a'_{j+m}$ for some $m \in \{0, \dots, d-1\}$, 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. Remark that for copy and anticopy of arity 2, we label the edges representing them by 1 and 2 (respectively $\bar{1}$ and $\bar{2}$), for the sake of simplicity. If we were considering trees of arity higher than 2, it would be necessary to distinguish the copies of different arity. Here, there is no ambiguity, as a 1 edge will always have the same source as a 2 edge and vice versa, by definition.

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



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

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

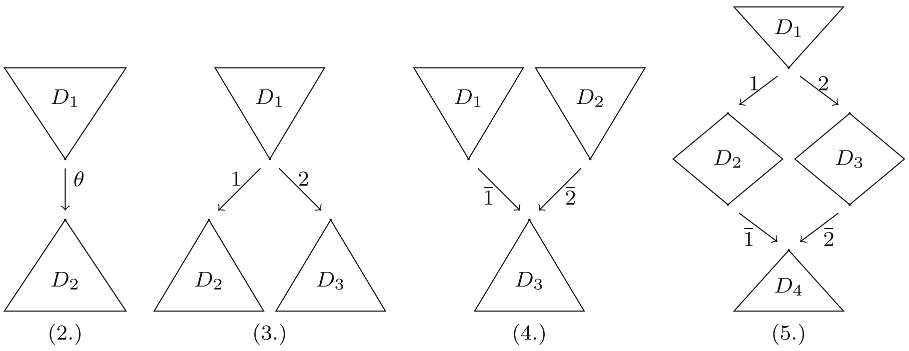


Fig. 4 Possible decompositions of a compound operation, numbered according to the items in Definition 2

4. $D = (D_1 \cdot_{1,1} (D_2 \cdot_{1,2} D_{\overline{\mathrm{copy}}_n^2})) \cdot_{1,1} D_3$ with $|O_{D_1}| = |O_{D_2}| = |I_{D_3}| = 1$;
5. $D = ((((D_1 \cdot_{1,1} D_{\mathrm{copy}_n^2}) \cdot_{2,1} D_3) \cdot_{1,1} D_2) \cdot_{1,1} D_{\overline{\mathrm{copy}}_n^2}) \cdot_{1,1} D_4$, 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 trivial one. This induces in particular an order over the input vertices of D, and one over its output vertices.

Definition 3 Given a compound operation D, we define r_D^i the relation associated to its *localised application* starting in the i^{th} leaf of a stack tree t as follows:

1. If $D = \square$, then $r_D^i = \mathrm{id}$ (where id denotes the identity relation).
2. If $D = (D_1 \cdot_{1,1} D_\theta) \cdot_{1,1} D_2$ with $\theta \in Ops_{n-1} \cup \{\mathrm{copy}_n^1, \overline{\mathrm{copy}}_n^1\}$,
   then $r_D^i = r_{D_1}^i \circ r_\theta^i \circ r_{D_2}^i$.
3. If $D = ((D_1 \cdot_{1,1} D_{\mathrm{copy}_n^2}) \cdot_{2,1} D_3) \cdot_{1,1} D_2$, then $r_D^i = r_{D_1}^i \circ r_{\mathrm{copy}_n^2}^i \circ r_{D_3}^{i+1} \circ r_{D_2}^i$.
4. If $D = (D_1 \cdot_{1,1} (D_2 \cdot_{2,1} D_{\overline{\mathrm{copy}}_n^2})) \cdot_{1,1} D_3$, then $r_D^i = r_{D_1}^i \circ r_{D_2}^{i+1} \circ r_{\overline{\mathrm{copy}}_n^2}^i \circ r_{D_3}^i$.
5. If $D = ((((D_1 \cdot_{1,1} D_{\mathrm{copy}_n^2}) \cdot_{2,1} D_3) \cdot_{1,1} D_2) \cdot_{1,1} D_{\overline{\mathrm{copy}}_n^2}) \cdot_{1,1} D_4$,
   then $r_D^i = r_{D_1}^i \circ r_{\mathrm{copy}_n^2}^i \circ r_{D_3}^{i+1} \circ r_{D_2}^i \circ r_{\overline{\mathrm{copy}}_n^2}^i \circ r_{D_4}^i$.

The relation defined by D is $r_D = \bigcup_i r_D^i$, i.e. the union of all its localised applications.

Remark 1 An operation may admit several different decompositions with respect to Definition 2. However, its application is well-defined, as one can show that this process is locally confluent. This result can be obtained through a case analysis of every possible decomposition of a DAG. This observation being easy but fastidious to write, we leave it to the reader.

Figure 2 shows an example of the application of a compound operation. Given a compound operation D, we define $\overline{D}$ the operation obtained by reversing the edges of D and swapping the input and output vertices. Formally, for $\theta \in Ops_{n-1}(\Gamma)$, we define $\overline{D_\theta} = D_{\overline{\theta}}$, $\overline{D_{\mathrm{copy}_n^d}} = D_{\overline{\mathrm{copy}_n^d}}$, $\overline{D_{\overline{\mathrm{copy}_n^d}}} = D_{\mathrm{copy}_n^d}$, and for every compound operations D_1, D_2, $\overline{D_1 \cdot_{i,j} D_2} = \overline{D_2} \cdot_{j,i} \overline{D_1}$. For every operation D, we have $r_{\overline{D}} = r_D^{-1}$. Finally, given a k-tuple of operations $\mathbf{D} = (D_1, \ldots, D_k)$ 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 $j \in \{1, \cdots, k\}$, we denote by $r_{\mathbf{D}}^{\mathbf{i}} = r_{D_k}^{i_k} \circ \cdots \circ r_{D_1}^{i_1}$ the relation defined by the parallel application of each D_j to the position i_j, and by $r_{\mathbf{D}}$ the union of all these relations (we apply the rightmost DAGs first to not have to deal with the effect of the application of the leftmost DAGs to the numbering of the leaves).

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 Definition 2. 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 Definition 2, any operation can be decomposed into a concatenation of DAGs of $\mathcal{D}_n$. □

Ground Stack Tree Rewriting Systems By analogy with order-1 trees, given some finite alphabet of labels Σ, we call labelled *ground stack tree rewriting system* (GSTRS) a set $R = \{R_a \mid a \in \Sigma\}$, where each R_a is a finite set of operations. 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 $\overline{2}$) are isomorphic to the configuration graphs of order-n pushdown automata performing a finite sequence of operations at each transition.

4 Operation Automata

Our main goal is to prove the decidability of the $\mathsf{FO}[\xrightarrow{*}]$-theory of the graphs generated by GSTRS, and to do so, we need to be able to calculate the reachability relation

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

of a GSTRS. In the case of trees, this is achieved through the definition of ground tree transducers, which are pairs of tree automata [12, 14]. However, as for higher-order stacks, there is no natural notion of recognisable sets of stack trees which could allow to directly define an equivalent of GTT for stack trees. In [5], Carayol defines a notion of recognisability over sets of operations and derives from it a notion of recognisability over higher-order stacks as the image of a given stack by a recognisable set of operations. This definition ensures that the reachability set of a higher-order pushdown automaton is recognisable. Following this idea, we introduce a notion of recognisability over stack trees operations, through a notion of operation automaton, and derive from it a notion of recognisability over stack tree. We thus show that the recognisable sets of operations are closed under union, intersection. Then, we associate two notions of relation associated with an automaton, one leading to recognisable ground stack tree rewriting systems (RGTRS), and the other to ground stack tree transducers (GSTT), following the ideas of [12, 14]. However, this is not sufficient to be able to describe the reachability relation of a GSTRS. Indeed, the extension of the construction on word automaton allowing to recognise the star of a language will fail (we will in general recognise more operations that the iteration of the initial set). To contourn that difficulty, we introduce a normalisation notion over automata, and show that every relation defined by an automaton can be defined by a normalised automaton. We then show that for every normalised and distinguished automaton (similarly to the notion over words automata, it means that initial states are target of no transition and final states are source of no transition), we can construct an automaton which recognises its iteration. We use that property to show that the reachability relations of GSTRS are captured by GSTT. Finally, let us precise that the normalisation property is also crucial in the proof of the decidability of the $\mathsf{FO}[\xrightarrow{*}]$-theory of a graph generated by a GSTRS presented in the next section.

Definition 4 An automaton over $\mathcal{D}_n^*$ is a tuple $A = (Q, \Gamma, I, F, \Delta)$, where

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

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

Such a labelling is called an *accepting labelling*.

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. We denote by $\mathrm{r}_A = \{(t, t') \mid \exists D \in \mathrm{Op}(A), \mathrm{r}_D(t, t')\}$ the union of the relations defined by the operations recognised by A.

A set of stack trees L is *recognisable* if there exists a recognisable set of operations R such that $L = \{t \mid \exists D \in R, (t_0, t) \in r_D\}$ where t_0 is a stack tree containing only one node labelled by $[\alpha]_{n-1}$, for a given $\alpha \in \Gamma$.

4.1 Properties of Operation Automata

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

Proposition 2 *Given two automata A_1 and A_2, there exists an automaton A such that $Op(A) = Op(A_1) \cap Op(A_2)$.*

Proof We construct an automaton which witnesses Proposition 2. We consider the product automaton of A_1 and A_2: we take $Q = Q_{A_1} \times Q_{A_2}$, $I = I_{A_1} \times I_{A_2}$, $F = F_{A_1} \times F_{A_2}$, and

$$\begin{aligned}\Delta = \;& \left\{((q_1, q_2), \theta, (q_1', q_2')) \mid (q_1, \theta, q_1') \in \Delta_{A_1} \wedge (q_2, \theta, q_2') \in \Delta_{A_2}\right\}\\ & \cup \left\{(((q_1, q_2), (q_1', q_2')), (q_1'', q_2'')) \mid ((q_1, q_1'), q_1'') \in \Delta_{A_1} \wedge ((q_2, q_2'), q_2'') \in \Delta_{A_2}\right\}\\ & \cup \left\{((q_1, q_2), ((q_1', q_2'), (q_1'', q_2''))) \mid (q_1, (q_1', q_1'')) \in \Delta_{A_1} \wedge (q_2, (q_2', q_2'')) \in \Delta_{A_2}\right\}.\end{aligned}$$

If an operation admits a valid labelling in A_1 and in A_2, then the labelling which labels each node of the operation 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 (respectively A_2), we have a valid labelling in A_1 (respectively A_2). □

Proposition 3 *Given two automata A_1 and A_2, there exists an automaton A such that $Op(A) = Op(A_1) \cup Op(A_2)$.*

Proof We take the disjoint union of A_1 and A_2, i.e. $Q = Q_{A_1} \uplus Q_{A_2}$, $I = I_{A_1} \uplus I_{A_2}$, $F = F_{A_1} \uplus F_{A_2}$ and $\Delta = \Delta_{A_1} \uplus \Delta_{A_2}$.

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 every operation is connected) and then the labelling is valid in A_1 or in A_2. □

Proposition 4 *Given an operation D, there exists an automaton A such that $Op(A) = \{D\}$.*

Proof If $D = (V, E)$, we take $Q = V$, I as the set of input vertices, F as the set of output vertices and

$$\begin{aligned}\Delta = \ & \{(q,\theta,q') \mid (q,\theta,q') \in E\}\\ & \cup\{(q,(q',q'')) \mid (q,1,q') \in E \wedge (q,2,q'') \in E\}\\ & \cup\{((q,q'),q'') \mid (q,1,q'') \in E \wedge (q',2,q'') \in E\}.\end{aligned}$$

The recognised connected part is D by construction. □

4.2 Recognisable Ground Stack Tree Rewriting Systems and Ground Stack Tree Transducers

It is possible to extend the notion of ground stack tree rewriting systems to recognisable sets of operations. Such a system is called recognisable.

Definition 5 A Σ-labelled recognisable ground stack tree rewriting system R is a set of $|\Sigma|$ operation automata A_a. Given $a \in \Sigma$ and two stack trees t, t', we have $t \underset{R}{\overset{a}{\rightarrow}} t'$ if there exists $D \in \mathrm{Op}(A_a)$ such that $(t, t') \in r_D$.

As in the case of ground tree rewriting systems, recognisable ground stack tree rewriting systems do not capture the iteration of the transition relation of any ground stack tree rewriting system, as such a system impose to rewrite only a given subtree, while the iteration of the transition relation may rewrite several subtrees in one step. For ground tree rewriting systems, this problem is lifted by the introduction of ground tree transducers which precisely allow to rewrite in parallel any number of subtrees with a set of rules in a recognisable set of rules (but without control on the number of subtrees rewritten that way). Following that idea, we define the relation defined by an automaton A as $\mathcal{R}(A) = \{(t, t') \mid \exists \mathbf{D} \in \mathrm{Op}(A), (t, t') \in r_{\mathbf{D}}\}$.

Definition 6 A Σ-labelled ground stack tree transducer Λ is a set of $|\Sigma|$ operation automata A_a. Given $a \in \Sigma$ and two stack trees t, t', we have $t \underset{\Lambda}{\overset{a}{\rightarrow}} t'$ if $(t, t') \in \mathcal{R}(A_a)$.

4.3 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 *bubbles*. 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 inspired from this work.

To define such a notion for our operations, 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. This idea is similar to the way ground tree rewriting is performed at order 1.

The goal of this section is to define a subset of operations called *reduced operations* analogously to the notion of reduced instructions with tests in [5], and prove that any recognisable relation can be defined by a recognisable set of reduced operations. However, as in [5], we won't have a unique reduced operation representing a given relation, due to the presence of tests, but it limits the number of times the same stack tree can be obtained during its application to a stack tree, which is exactly what we need in the proof of the next section. To that end, we will first formally define tests, define the notion of reduced operations with tests, and prove that any relation accepted by an automaton can be accepted by an automaton accepting reduced operation with tests (with some technical restrictions), called a *normalised automaton*. A key point in the proof of that fact will be to use the notion of loop-free sets of instructions presented in [5], and we will explain why we can apply it to our case, as we don't have the same set of basic operations as them.

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 $s \in Stacks_n$, $(s,s) \in r_{T_L}$ if and only if $s \in L$. 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 now define a notion of reduced sequences of stack operations that will be sufficient for limiting the number of time a given stack tree appears during the application of a reduced operation. Informally, we say that a sequence over $Ops_{n-1} \cup \mathcal{T}_{n-1}$ is reduced if and only if the only subsequences that define a relation that can leave a stack unchanged are single test operations.

[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].

Definition 7 We define the set of reduced operations over $Ops_{n-1} \cup \mathcal{T}_{n-1}$ by

$$\mathrm{Red} = \left\{ \theta_1 \cdots \theta_k \mid \begin{array}{c} \forall i \leq j, r_{\theta_i \cdots \theta_j} \cap \mathrm{id} \neq \emptyset \Rightarrow (i = j \wedge \theta_i \in \mathcal{T}_{n-1}) \wedge \\ \forall i \leq j, \left(\theta_i = \mathrm{rew}_{\mathrm{a,b}} \wedge \theta_j = \mathrm{rew}_c d\right) \Rightarrow \\ \exists z, i < z < j \wedge \left(\theta_z = \mathrm{copy}_k \vee \theta_z = \overline{\mathrm{copy}}_k\right) \end{array} \right\}.$$

If a sequence $\theta = \theta_1 \cdots \theta_k$ is reduced, then for every stack s such that there is a stack s' such that $(s, s') \in r_\theta$, the sequence $s_1, \cdots, s_{k-1}$ with $(s, s_1) \in r_{\theta_1}$, $(s_1, s_2) \in r_{\theta_2}, \cdots, (s_{k-1}, s') \in r_{\theta_k}$ is such that for any s_i, s_j, if $s_i = s_j$, then $j = i + 1$ and $\theta_i \in \mathcal{T}_{n-1}$. Notice, that this definition also implies that there are no two consecutive test operations in a reduced sequence. Thus, in the application of a reduced sequence, any stack will appear at most twice. Furthermore, we also ask that there are no two $\mathrm{rew}_{\mathrm{a,b}}$ operations which can apply successively on the same letter. This prevents useless rewriting (as $\mathrm{rew}_{\mathrm{a,b}}\mathrm{rew}_{\mathrm{b,c}}$ is equivalent to $\mathrm{rew}_{\mathrm{a,c}}$).

For example, the sequence $\mathrm{rew}_{\mathrm{a,a}}\mathrm{copy}_1\mathrm{rew}_{\mathrm{a,b}}$ is not reduced as $\mathrm{rew}_{\mathrm{a,a}}$ is not a test but $([\alpha]_1, [\alpha]_1) \in r_{\mathrm{rew}_{\mathrm{a,a}}}$. Yet, $\mathrm{copy}_1\mathrm{rew}_{\mathrm{a,b}}$ is reduced. The sequence $\theta = \overline{\mathrm{copy}}_2\mathrm{rew}_{\mathrm{a,b}}$ $\mathrm{copy}_1 T_L \overline{\mathrm{cop}}_1 \mathrm{rew}_{\mathrm{b,a}}\mathrm{copy}_2$ is not reduced as $([[\alpha]_1[\alpha]_1]_2, [[\alpha]_1[\alpha]_1]_2) \in r_\theta$. The sequence $\theta = \overline{\mathrm{copy}}_2\mathrm{rew}_{\mathrm{a,b}}\mathrm{copy}_2$ is reduced. $T_{L_1}\mathrm{rew}_{\mathrm{a,b}}T_{L_2}\mathrm{copy}_1 T_{L_3}$ is reduced, as even if it has three test operations, they are not consecutive, and $\mathrm{rew}_{\mathrm{a,b}}\mathrm{copy}_1$ is reduced.

We can now define our notion of reduced stack tree operations. Informally, a stack tree operation is reduced if there is no $\overline{\mathrm{copy}}_n^d$ operation "below" a copy_n^d operation, and if any suboperation containing only stack operations is reduced.

Definition 8 We define the set of reduced sequences over $Ops_{n-1} \cup \mathcal{T}_{n-1} \cup \{\bar{1}, \bar{2}, 1, 2, \overline{\mathrm{copy}}_n^1, \mathrm{copy}_n^1\}$ by

$$\mathrm{TRed} = \left\{ \theta_1 \cdots \theta_k \mid \begin{array}{c} \forall i \leq j, \theta_i \cdots \theta_j \in (Ops_{n-1} \cup \mathcal{T}_{n-1})^* \Rightarrow \theta_i \cdots \theta_j \in \mathrm{Red} \wedge \\ \forall i, j, \left(\theta_i \in \{\bar{1}, \bar{2}, \overline{\mathrm{copy}}_n^1\} \wedge \theta_j \in \{1, 2, \mathrm{copy}_n^1\}\right) \Rightarrow i < j \end{array} \right\}.$$

An operation D is *reduced* if for any vertices x, y of D if $x \xrightarrow{w} y$, then $w \in \mathrm{TRed}$.

Intuitively, a reduced operation will have a *destructive part*, which contains all the operations $\overline{\mathrm{copy}}_n^d$ and thus "goes up (and only up)" in the stack tree it is applied to, followed by a *constructive part*, which contains all the operations copy_n^d and thus "constructs a stack tree from the leaf it is applied to", and never removes a leaf. Moreover, any suboperation that does not contain tree operation is reduced, preventing it to "loop" on a stack tree if it is not a single test operation.

We are now ready to present the main result of this section: a notion of normalisation for operation automata. An automaton A is said to be *distinguished* if there is no transition ending in an initial state or starting in a final state.

Definition 9 An automaton A with state set Q is said to be normalised if it accepts only reduced operations, it is distinguished, and furthermore Q admits two partitions:

– into Q_T and Q_C such that all test transitions lead from Q_C to Q_T, while all other lead from Q_T to Q_C.

– into Q_c and Q_d such that there is no transition from Q_c to Q_d, states of Q_d are target of no copy_n^d transition, states of Q_c are source of no $\overline{\mathrm{copy}}_n^d$ transition, and all transition from Q_d to Q_c are copy_n^d transitions.

Theorem 1 *For every automaton A, there exists a normalised automaton with tests A_r such that* $\mathrm{r}_A = \mathrm{r}_{A_r}$.

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. This transformation can be performed without altering the accepted relation over stack trees. The end of this section is devoted to explain this construction.

Before moving to the main proof, we first have to explain how we can derive that for every recognisable set of stack operations there is a recognisable set of reduced stack operations defining the same relation, from [5]. Indeed, we will need that fact in our construction. In [5], the set of Ops_1 contains operations of the form push_α and pop_α, while we have the operations $\mathrm{rew}_{\alpha,\beta}$, copy_1 and $\overline{\mathrm{cop}}_1$. Notice as well that in [5], there is an empty stack that we don't consider in our model. Yet it is possible to simulate our operations with sets of sequences of push_α and pop_α, and vice-versa. For example, $\mathrm{push}_\alpha \sim \bigcup_{\beta\in\Gamma} \mathrm{copy}_1\mathrm{rew}_\beta\alpha$ (observe that the relations defined by a sequence in this union are disjoint), or $\mathrm{copy}_1 \sim \bigcup_{\alpha\in\Gamma} \mathrm{pop}_\alpha\mathrm{push}_\alpha\mathrm{push}_\alpha$. The remaining equivalences are similar and thus left to the reader.

Thus, as these are local transformations, any relation defined by a recognisable set for our notion is also defined by a recognisable set for the notion of [5], and vice-versa.

In [5], there is a notion of *loop-free* operations, which are operations without factors of the form $\mathrm{copy}_i\overline{\mathrm{copy}}_i$, $\overline{\mathrm{copy}}_i\mathrm{copy}_i$, $\mathrm{push}_a\mathrm{pop}_a$ or $\mathrm{pop}_a\mathrm{push}_a$. Observe that a loop-free operation (without test) cannot contain a suboperation θ such that $r_\theta \cap \mathrm{id} \neq \emptyset$, as an operation defining a test necessarily contains one of these factors. They extend this notion to operations with tests (an operation with test is loop-free if the one obtained by removing the tests is loop-free). Observe that a loop-free operation with tests is reduced in our sense (except that there may be several tests in a row – but one can find easily an equivalent operation reduced in our sense by shrinking these tests). Proposition 2.2 from [5] thus states that any set recognised by an automaton A over operations defines the same relation as a set recognised by an automaton A' over $Ops_k \cup \{T_{L_{q,q'}} \mid q, q' \in Q\}$ accepting only loop-free operations, where $L_{q,q'}$ is the set of stacks s, such that there is an operation θ recognised by A between states q and q' such that $(s,s) \in r_\theta$. Furthermore, they show that the sets $L_{q,q'}$ are regular, so it is possible to compute A'.

Finally, observe that for any loop-free operation with tests in the sense of [5], we can find a finite set of reduced operations in our sense recognising the same relation, by applying the transformation described earlier, replacing the $\mathrm{rew}_\alpha\alpha$ with a test operation and merging the possible $T_L T_{L'}$ factors introduced that way with $T_{L\cap L'}$. Observe that a factor $\mathrm{rew}_{\mathrm{a,b}}T_L\mathrm{rew}_c d$ cannot appear in this transformation. These transformations being local, we can compute the set corresponding to a loop-free operation. For the same reason, we can compute a reduced automaton with tests from a loop-free automaton with tests.

Thus, we deduce from [5] the following lemma:

Lemma 1 *Given an automaton A over Ops_{n-1}, there exists an automaton over $Ops_{n-1} \cup \mathcal{T}_{n-1}$ accepting only reduced operations recognising the same relation as A.*

We now give the proof of Theorem 1.

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 first prevent the presence of bubbles, i.e. suboperations containing a copy_n^d operation above a $\overline{\mathrm{copy}}_n^{d'}$ operation, and afterwards to replace all sequences of stack operations with reduced sequences, using Lemma 1. Finally, we will impose that between any two non-test operations there is a test operation, and there are no two consecutive test operations.

To forbid bubbles, we just have to prevent the automaton from recognising DAGs which contain

$$((D_{\mathrm{copy}_n^2} \cdot_{1,1} F_1) \cdot_{2,1} F_2) \cdot_{1,1} D_{\overline{\mathrm{copy}}_n^2}, \text{ or } (D_{\mathrm{copy}_n^1} \cdot_{1,1} F) \cdot_{1,1} D_{\overline{\mathrm{copy}}_n^1}$$

as a subDAG, where F_1, F_2 and F have 1 input node and 1 output node. 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 1), and then by ensuring that there won't be any $\overline{\mathrm{copy}}_n^i$ transition after the first copy_n^j transition (step 2).

Step 1: We consider an automaton $A = (Q, \Gamma, I, F, \Delta)$. Given two states q_1, q_2, we denote by $L_{A_{q_1,q_2}}$ the set $\{s \in Stacks_{n-1} \mid \exists D \in \mathcal{D}(A_{q_1,q_2}), (t_s, t_s) \in r_D^1\}$ where A_{q_1,q_2} is a copy of A in which we take q_1 as the unique initial state and q_2 as the unique final state, and t_s is the stack tree with a unique node labelled by s. 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 a stack of this set remain unchanged by an operation recognised by A_{q_1,q_2}. We define $A_1 = (Q, \Gamma, I, F, \Delta')$ with

$$\begin{aligned} \Delta' = {} & \Delta \\ & \cup \{(q, T_{L_{A_{r,r'}} \cap L_{A_{s,s'}}}, q') \mid (q, (r, s)), ((r', s'), q') \in \Delta\} \\ & \cup \left\{(q, T_{L_{A_{r,r'}}}, q'_s \mid \left(q, \mathrm{copy}_n^1, r\right), \left(r', \overline{\mathrm{copy}}_n^1, q'\right) \in \Delta\right\}, \end{aligned}$$

The idea of the construction is depicted in Fig. 5. Intuitively, we add, between every pair of states, a shortcut that replace a suboperation starting by a copy_n^d operation, ending by a $\overline{\mathrm{copy}}_n^d$ and not going upper in the stack tree it is applied to (i.e. a bubble) by a test operation which accepts the same $(n-1)$-stacks (indeed, such an operation cannot modify the stack tree it is applied to). However in this step, we do not formally remove these bubbles (that will be done in the next step).

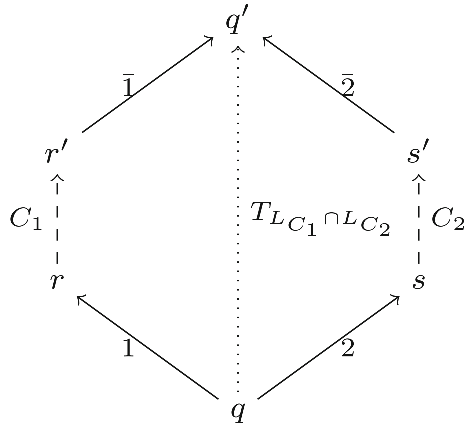


Fig. 5 Step 2: The added test transition to shortcut the bubble is depicted with a *dotted line*

We give a lemma which will allow us to prove that adding these test operations does not modify the relation recognised by the automaton.

Lemma 2 *Let $C = (Q, \Gamma, I, F, \Delta_C)$ be an automaton, and i_1, i_2, f_1, f_2 four states of Q (not necessarily distinct). For short, we denote $L_{C_{i_k,f_k}}$ by $L_{C,k}$, for $k \in \{1, 2\}$. We define the two automata $B_1 = (\{q_1, q_2\}, \Gamma\{q_1\}, \{q_2\}, \Delta_1)$ and $B_2 = (Q \cup \{q_1, q_2\}, \Gamma, \{q_1\}, \{q_2\}, \Delta_2)$, where q_1 and q_2 are two different states not in Q,*

$$\Delta_1 = \{(q_1, T_{L_{C,1} \cap L_{C,2}}, q_2)\}, \qquad \Delta_2 = \{(q_1, (i_1, i_2)), ((f_1, f_2), q_2)\} \cup \Delta_C.$$

The relation recognised by B_1 is included in the relation recognised by B_2.

We also define the two automata $B_3 = (\{q_1, q_2\}, \Gamma, \{q_1\}, \{q_2\}, \Delta_3)$ and $B_4 = (Q \cup \{q_1, q_2\}, \Gamma, \{q_1\}, \{q_2\}, \Delta_4)$, where,

$$\Delta_3 = \{(q_1, T_{L_{C,1}}, q_2)\}, \qquad \Delta_4 = \left\{\left(q_1, \mathrm{copy}_n^1, i_1\right), \left(f_1, \overline{\mathrm{copy}}_n^1, q_2\right)\right\} \cup \Delta_C.$$

The relation recognised by B_3 is included into the relation recognised by B_4.

Proof The two inclusions having very similar proof, we only detail the first one and leave the second to the reader.

Take $s \in L_{C,1} \cap L_{C,2}$. By definition there are $F_1 \in \mathcal{D}(C_{i_1,f_1})$ and $F_2 \in \mathcal{D}(C_{i_2,f_2})$ such that $(t_s, t_s) \in r_{F_1} \cap r_{F_2}$, where t_s is the stack tree with a unique node labelled by s. By extension, we get that any stack tree t with the stack s as the label of one of its leaves is such that $(t, t) \in r_{F_1} \cap r_{F_2}$. Conversely, by definition $r_{T_{L_{C,1} \cap L_{C,2}}}$ only accepts pairs of the form (t, t) where one of the leaf of t is labelled by a stack in $L_{C,1} \cap L_{C,2}$. We consider the DAG $D = D_{\mathrm{copy}_n^2} \cdot_{1,1} (F_1 \cdot_{1,1} (F_2 \cdot_{2,2} D_{\overline{\mathrm{copy}}_n^2}))$. By definition of its components, we get that $(t, t) \in r_D$. It is easy to see that D is recognised by B_2.

Thus, for any stack tree such that $(t, t) \in r_{T_{L_{C,1} \cap L_{C_2}}}$, we can find D recognised by B_2 such that $(t, t) \in r_D$, which concludes the proof. $\square$

The other direction of the inclusions is false. Indeed, in B_2 (resp. B_4), some operation that can modify node upper than the one they are applied to may be recognised (e.g. some operation starting by $\mathrm{copy}_n^d\overline{\mathrm{copy}}_n^d\overline{\mathrm{copy}}_n^{d'}$). As such an operation cannot accept a tree with a single node, it will not be shortcut by the test operation introduced in B_1. This will not be a problem in our construction, as we simply need that by adding the test operations, we do not modify the recognised relation.

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

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

Indeed A_1 is obtained from A by adding sub-automata of the form B_1 between two states forming the initial and final states of the form B_2. As the lemma shows that the relation recognised by an automaton of the form B_1 is included in the one recognised by the corresponding automaton of the form B_2, we do not add anything to the relation recognised by A_1. As A is furthermore included in A_1, we get that the two automata recognise the same relation.

Step 2: Suppose that $A_1 = (Q, \Gamma, I, F, \Delta)$ is the automaton obtained after step 1. We now want to really forbid bubbles. To do so, we split the control states of the 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 $A_2 = (Q', \Gamma, I', F', \Delta')$ with $Q' = \{q_d, q_c \mid q \in Q\}$, $I' = \{q_d, q_c \mid q \in I\}$, $F' = \{q_d, q_c \mid q \in F\}$ and

$$\begin{aligned}\Delta' = &\left\{\left(q_d, \theta, q'_d\right), \left(q_c, \theta, q'_c\right) \mid (q, \theta, q') \in \Delta, \theta \in Ops_{n-1} \cup \mathcal{T}_{n-1} \cup \{\mathrm{id}\}\right\}\\ &\cup \left\{\left(\left(q_d, q'_d\right), q''_d\right) \mid ((q, q'), q'') \in \Delta\right\}\\ &\cup \left\{\left(q_d, \overline{\mathrm{copy}}_n^1, q'_d\right) \mid \left(q, \overline{\mathrm{copy}}_n^1, q'\right) \in \Delta\right\}\\ &\cup \left\{\left(q_c, \left(q'_c, q''_c\right)\right), \left(q_d, \left(q'_c, q''_c\right)\right) \mid (q, (q', q'')) \in \Delta\right\}\\ &\cup \left\{\left(q_c, \mathrm{copy}_n^1, q'_c\right), \left(q_d, \mathrm{copy}_n^1, q'_c\right) \mid \left(q, \mathrm{copy}_n^1, q'\right) \in \Delta\right\}.\end{aligned}$$

Lemma 3 *A_1 and A_2 recognise the same relation.*

Proof A_2 recognises the operations recognised by A_1 which contain no bubble. Indeed, every labelling of such an operation in A_1 can be modified to be a labelling in A_2 by replacing q by q_d or q_c. Conversely, each operation recognised by A_2 is recognised by A_1.

Let us take D recognised by A_1 which contains at least one bubble. Suppose that D contains a bubble F and that $D = D'[F]_x$ where D' is a DAG such that we obtain D by replacing the node x by F in D', and F contains exactly one bubble (said otherwise, F is one of the innermost bubble of D).

Let us suppose $F = D_{\mathrm{copy}_n^2} \cdot_{1,1} (F_1 \cdot_{1,1} (F_2 \cdot_{1,2} D_{\overline{\mathrm{copy}}_n^2}))$, with F_1 and F_2 containing no tree operation (as F is an innermost bubble). We call x_k the input node of F_k and y_k its output node. As D is accepted by A_2, we can find an accepting labelling ρ such that there are states r, r', s, s' such that $\rho(x_1) = r$, $\rho(y_1) = r'$, $\rho(x_2) = s$, $\rho(y_2) = s'$. Thus, from step 1, $G = D'[T_{L_{A_{r,r'}} \cap L_{A_{s,s'}}}]_x$ is recognised by A_2. Then $r_D \subseteq r_G$, and G has one less bubble than D (as F was an innermost bubble).

The case where we consider $F = D_{\mathrm{copy}_n^1} \cdot_{1,1} (F_1 \cdot_{1,1} D_{\overline{\mathrm{copy}}_n^1})$, where F_1 does not contain any tree operation is similar, and thus omitted.

Iterating this process, we obtain an operation H without any bubble such that $r_D \subseteq r_H$ and H is recognised by A_1. As it contains no bubble, it is also recognised by A_2.

Thus every relation recognised by an operation with bubbles is already included in a relation recognised by an operation without bubbles. Therefore A_1 and A_2 recognise the same relation. □

We call the destructive part the restriction $A_{2,d}$ of A_2 to Q_d and the constructive part its restriction $A_{2,c}$ to Q_c. Notice that the two parts are separated by constructive operations (i.e. copy_n^d operations), thus the automaton labels all the nodes before the first copy_n^d operation with Q_d, and all the nodes after it with Q_c.

Step 3: We consider an automaton $A_2 = (Q, \Gamma, I, F, \Delta)$ obtained after the previous step. We now want every sequence of stack operations to be reduced. To do that, we will, as explained before, invoke the result of Lemma 1, but first we have to ensure that between two reduced sequences of stack operations, there is at least one tree operation, as the concatenation of two reduced sequences of stack operations is not necessarily reduced.

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

$$\begin{aligned}
Q' &= \{q_s, q_t \mid q \in Q\},\\
I' &= \{q_s, q_t \mid q \in I\},\\
F' &= \{q_s, q_t \mid q \in F\},\\
\Delta' &= \left\{(q_s, \theta, q'_s), (q_s, \theta, q'_t) \mid (q, \theta, q') \in \Delta, \theta \in Ops_{n-1} \cup \mathcal{T}_{n-1}\right\}\\
&\cup \left\{(q_t, (q'_t, q''_t)), (q_t, (q'_s, q''_t)), (q_t, (q'_t, q''_s)), (q_t, (q'_s, q''_s)) \mid (q, (q', q'')) \in \Delta\right\}\\
&\cup \left\{((q_t, q'_t), q''_t), ((q_t, q'_t), q''_s) \mid ((q, q'), q'') \in \Delta\right\}\\
&\cup \left\{\left(q_t, \mathrm{copy}_n^1, q'_t\right), \left(q_t, \mathrm{copy}_n^1, q'_s\right) \mid \left(q, \mathrm{copy}_n^1, q'\right) \in \Delta\right\}\\
&\cup \left\{\left(q_t, \overline{\mathrm{copy}}_n^1, q'_t\right), \left(q_t, \overline{\mathrm{copy}}_n^1, q'_s\right) \mid \left(q, \overline{\mathrm{copy}}_n^1, q'\right) \in \Delta\right\}.
\end{aligned}$$

Observe that in this automaton, states of Q_s are sources of all stack transitions and only them, and Q_t are sources of all tree transitions and only them.

Lemma 4 *A_2 and A_3 recognise the same relation.*

Proof If an operation is accepted by A_3 with the labelling ρ, we construct a labelling ρ' by $\rho'(x) = q$ if $\rho(x) = q_s$ or $\rho(x) = q_t$. By construction, $\rho(x)$ is initial (resp. final) if and only if $\rho'(x)$ is initial (resp. final), and ρ' respects the transitions of A_2. Thus the operation is accepted by A_2

Conversely, if an operation is accepted by A_2 with the labelling ρ, we construct a labelling ρ': if $\rho(x) = q$, we take $\rho'(x) = q_s$ if there exists y and $\theta \in Ops_{n-1} \cup \mathcal{T}_{n-1}$ such that $x \overset{\theta}{\to} y$, and $\rho'(x) = q_t$ otherwise. By construction of the automaton, ρ' respects the transitions of A_3, and state q_s and q_t are initial (resp. final) if and only if q is initial (resp. final). Thus the operation is accepted by A_3. □

Step 4: We consider an automaton $A_3 = (Q, \Gamma, I, F, \Delta)$ obtained after the previous step.

We now only have to replace every sequence of stack operations between two stack transitions (or initial and final states or mix of both cases) with a reduced one, and we know that such can only be labelled with q_s states except the last node which is labelled with a q_t node (in the case of a final node, it is not compulsory, but it always can). For each pair of states (q_s, q_t') we consider the subautomaton $A_{q_s,q_t'}$ with q_s as only initial state, q_t' as only final state, and which conserve only stack transitions. From Lemma 1, we consider the normalised automaton over stack operation $A'_{q_s,q_t'}$ which accepts the same language. Without loss of generality, we assume these automata to be distinguished, and to have disjoint sets of non-initial and non-final states. We also consider that $A'_{q_s,q_t'}$ has q_s as unique initial state and q_t' as unique final state.

We thus construct $A_4 = (Q', \Gamma, I', F', \Delta')$ with

$$Q' = Q \cup \bigcup_{q_s,q_t' \in Q} Q_{A'_{q_s,q_t'}}, \qquad I' = I, \qquad F' = F,$$

$$\Delta' = \Delta \backslash \{(q, \theta, q') \in \Delta \mid \theta \in Ops_{n-1} \cup \mathcal{T}_{n-1}\} \cup \bigcup_{q_s,q_t' \in Q} \Delta_{A'_{q_s,q_t'}}.$$

Informally, A_4 is simply A_3 in which we removed all stack transitions and we added between q_s and q_t' the normalised automaton accepting the same relation as $A_{q_s,q_t'}$. Figure 6 shows the idea of this transformation.

Lemma 5 *A_3 and A_4 recognise the same relation.*

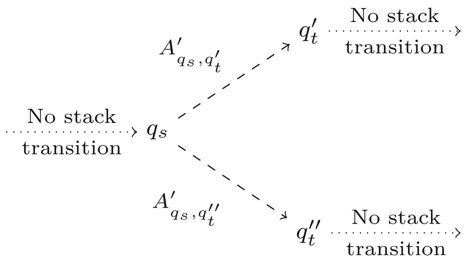


Fig. 6 Step 4: The normalisation of the stack part of the automaton

Proof As A_4 is obtained by normalising the parts of A_3 containing only operations of $Ops_{n-1} \cup \mathcal{T}_{n-1}$, the relation defined by the operations of $(Ops_{n-1} \cup \mathcal{T}_{n-1})^*$ recognised between two states q_s and q'_t is equal for A_3 and A_4. For every pair (t, t') in the relation defined by A_3, there exists an operation θ recognised by A_3 such that $(t, t') \in r_\theta$. As the remaining of the automaton is unchanged, it is possible to modify θ by replacing every maximal subDAG labelled with states between a q_s and a q'_s by an operation of $(Ops_{n-1} \cup \mathcal{T}_{n-1})^*$ in the set recognised by A_{q_s,q'_s} (belonging to Red) and thus obtain an operation θ' recognised by A_4 such that $(t, t') \in r_{\theta'}$. It follows that (t, t') is in the relation recognised by A_4. The other direction is proved the same way. □

Step 5: We now split the control states set into two parts Q_T and Q_C such that all test transitions lead from Q_C to Q_T, while all other transitions lead from Q_T to Q_C.

To that end, we will add a test transition which accepts all stacks in order to serve as a link between two consecutive non-test transitions. Given automaton $A_4 = (Q, \Gamma, I, F, \Delta)$ obtained from the previous step, we define $A_5 = (Q', \Gamma, I', F', \Delta')$ with

$$Q' = \{q_T, q_C \mid q \in Q\},\ I' = \{q_C \mid q \in I\},\ F' = \{q_T, q_C \mid q \in F\},$$

$$\begin{aligned}\Delta' = &\left\{\left(q_T, \theta, q'_C\right) \mid (q, \theta, q') \in \Delta, \theta \in Ops_{n-1} \cup \left\{\mathrm{copy}^1_n, \overline{\mathrm{copy}}^1_n\right\}\right\}\\ &\cup \left\{\left(\left(q_T, q'_T\right), q''_C\right) \mid ((q, q'), q'') \in \Delta\right\}\\ &\cup \left\{\left(q_T, \left(q'_C, q''_C\right)\right) \mid (q, (q', q'')) \in \Delta\right\}\\ &\cup \left\{\left(q_C, T_L, q'_T\right) \mid (q, T_L, q') \in \Delta\right\} \cup \{(q_C, T_\top, q_T) \mid q \in Q\}.\end{aligned}$$

Lemma 6 *A_4 and A_5 recognise the same relation.*

Proof From step 4 it is not possible to have two successive test transitions. Suppose an operation D is accepted by A_4. We construct D' by adding $D_{T_\top}$ between every two consecutive non-test operations. For example we replace $D_\theta \cdot_{1,1} D_{\mathrm{copy}^2_n}$ with $D_\theta \cdot_{1,1} D_{T_\top} \cdot_{1,1} D_{\mathrm{copy}^2_n}$. By construction of the automaton, if there is a valid labelling for D, we can construct a valid labelling for D' by replacing q with q_C before a test and with q_T otherwise (by construction, any node has a test operation touching it, or it is input or output), and for the added $D_{T_\top}$, we labelled its extremities with q_C and q_T, where q was the label of the node joining the two operation it was included between. As we only added $D_{T_\top}$ operations to D without changing the order of the other operations, and $D_{T_\top}$ has no effect at all, D and D' define the same relation.

Suppose an operation D is accepted by A_5. We construct D' by removing all $D_{T_\top}$ operations whose both ends are labelled by q_C and q_T for a same q. From a labelling ρ of D, we construct a labelling ρ' by replacing q_T and q_C with q, and the node resulting after the removal of a $D_{T_\top}$ is labelled by q if the nodes of $D_{T_\top}$ were labelled by q_C and q_T. By construction, such a $D_{T_\top}$ cannot be labelled otherwise, as we suppose A_4 has no such tests. Thus, by construction, ρ is a valid labelling of

D'. As we only removed operations which have no effect, D and D' accept the same relation. □

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). Observe that A_5 satisfies all the requirements of Theorem 1. Thus A_5 is a normalised automaton with tests which recognises the same relation as the initial automaton A. In subsequent constructions, we will consider the subsets of states Q_T, Q_C, Q_d, Q_c as defined in steps 5 and 2, and $Q_{u,v} = Q_u \cap Q_v$ with $u \in \{T, C\}$ and $v \in \{d, c\}$. □

4.4 Iteration of Ground Stack Tree Rewriting Systems

With normalised automata, we can now show that the iteration of any relation defined by a ground stack tree rewriting system or a ground stack tree transducer is defined by a ground stack tree transducer.

Proposition 5 *Given a distinguished automaton A, there exists an automaton A' that recognises $Op(A)^*$.*

Proof We construct A'. We take $Q = Q_A \uplus \{q\}$, $I = I_A \cup \{q\}$, $F = F_A \cup \{q\}$. The set of transitions 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:

$$\begin{aligned}\Delta = \Delta_A \\ &\cup\{(q_1, \theta, q_i) \mid q_i \in I_A, \exists q_f \in F_A, (q_1, \theta, q_f) \in \Delta_A\} \\ &\cup\{((q_1, q_2), q_i) \mid q_i \in I_A, \exists q_f \in F_A, ((q_1, q_2), q_f) \in \Delta_A\} \\ &\cup\{(q_1, (q_2, q_i)) \mid q_i \in I_A, \exists q_f \in F_A, (q_1, (q_2, q_f)) \in \Delta_A\} \\ &\cup\{(q_1, (q_i, q_2)) \mid q_i \in I_A, \exists q_f \in F_A, (q_1, (q_f, q_2)) \in \Delta_A\} \\ &\cup \left\{(q_1, (q_i, q_i')) \mid q_i, q_i' \in I_A, \exists q_f, q_f' \in F_A, \left(q_1, \left(q_f, q_f'\right)\right) \in \Delta_A\right\}.\end{aligned}$$

We first prove for every $k \in \mathbb{N}$ that if $D \in \mathrm{Op}(A)^k$, it has a valid labelling in A': The operation □ 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 $\mathrm{Op}(A)^\ell$ and F of $\mathrm{Op}(A)^m$, for $\ell, m \leq k$ and $\ell + m = k + 1$, 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'.

We now consider an operation $D = (V, E)$ accepted by A' with the labelling ρ. We first cut D into a collection of connected DAGs $D_1, \cdots, D_k$ by cutting D at the nodes labelled by states of I_A. We call D' the disconnected DAG which is the union of all the D_i. Formally, we define $D' = (V', E')$ with $V' = V \cup \{(x, 1) \mid \rho(x) \in I_A\}$, and $E' = \{(x, a, y) \mid (x, a, y) \in E \wedge \rho(y) \notin I_A\} \cup \{(x, a, (y, 1)) \mid (x, a, y) \in E \wedge \rho(y) \in I_A\}$. We as well consider that we remove DAGs which have only one node (i.e. those which were created by duplicating an input node of D). Observe that we can obtain D by concatenating the $D_1, \cdots, D_k$ together.

By definition of D', each D_i can be labelled by $\rho \cup \{(x, 1) \to \rho(x)\}$ while respecting the transitions of A' such that all input nodes are labelled by I_A all output nodes are labelled by I_A and F_A, and no other nodes are labelled by I_A and F_A (as otherwise, they would have been cut, we only cut at nodes labelled by I_A, and F_A can only label output nodes as A is distinguished). By definition of A', we can modify the labels of the output nodes of D_i labelled by a state of I_A by labelling them by states of F_A such that this modified labelling is a valid labelling of A. Thus all D_i are accepted by A.

We have shown that D can be obtained by concatenating DAGs recognised by A. We now have to prove that the D_i are operations.

We say that a DAG is *well-formed* if for every nodes x, y with an edge between x and y, the subDAG containing x, y, the successors of x, and the predecessors of y, and the edges between them, is a DAG representing a basic operation. From Definition 2, all operations are well-formed. Indeed an operation is inductively formed by concatenating basic operations between output and input nodes of already existing operations. Thus if a node is already the input node of a basic operation, by Definition 2, it will never be appended to the input node of another basic operation in the construction of an operation as it is not the output node of an operation (similarly for output nodes of basic operations). However, the converse is not true, as for example the DAG $D_{\mathrm{copy}_n^2} \cdot_{2,1} D_{\overline{\mathrm{copy}_n^2}}$ is well-formed but not an operation.

As the D_i are obtained by cutting the operation D so that for every node we either conserve all its predecessors (resp. successors) or drop them, they are well-formed. To conclude, we only have to prove that if a well-formed DAG is accepted by a normalised automaton, it is an operation.

Lemma 7 *A connected well-formed DAG that can be labelled by a normalised automaton is an operation.*

Proof We consider a normalised automaton $A = (Q, \Gamma, I, F, \Delta)$, and we consider the set of constructive states Q_c and of destructive states Q_d as in Theorem 1. We prove that a connected well-formed DAG D that can be labelled by A by a labelling ρ is an operation by induction on the number of its nodes. We furthermore prove that if D has an input node x with $\rho(x) \in Q_c$, then x is the only input node of D, and that if it has an output node y with $\rho(y) \in Q_d$, then y is the only output node of D.

The only DAG with 1 node is $\square$. It can be labelled by A (by any state), and by definition, it is an operation. Furthermore, its only node is the only input node and the only output node, thus we can label it by Q_c or Q_d and satisfy the induction hypothesis.

Suppose now that any connected DAG with at most n nodes that can be labelled by A satisfies the induction hypothesis. We take a connected well-formed DAG D with $n + 1$ nodes that can be labelled by A. We call ρ this labelling. We consider the leftmost input node x of D, and call $q = \rho(x)$. As ρ is consistent with the transitions of A and D is well-formed, we have the following cases for the successors of x:

- x has a unique successor y and we have the edge (x, θ, y) with $\theta \in Ops_{n-1} \cup \{\mathrm{copy}_n^1, \overline{\mathrm{copy}}_n^1\}$. As ρ is consistent with the transitions of A, we have a transition (q, θ, q') in A and $\rho(y) = q'$. Thus, if we consider D' obtained by removing x from D, we obtain a DAG with n nodes that can be labelled by A (by ρ restricted to it). By hypothesis of induction, D' is an operation. Moreover, as D is well-formed, y has only x as a predecessor, and is thus the leftmost input node of D'. The subDAG of D obtained by keeping only x and y is D_θ. Thus, we have $D = D_\theta \cdot_{1,1} D'$. From Definition 2, D is an operation. Furthermore, if $\rho(x) \in Q_c$, then as there is no transition from Q_d to Q_c, $\rho(y) \in Q_c$. Thus, by hypothesis of induction, y is the only input node of D'. Thus, x is the only input node of D. If D has an output node z labelled by Q_d, then as x is not an output node, z is an output node of D'. By hypothesis of induction it is its only output node. Thus D has a unique output node.
- x has a unique successor y and we have the edge $(x, \bar{1}, y)$. As ρ is consistent with the transitions of A, there exists a transition $((q, q'), q'')$ in A, $\rho(y) = q''$ and there is a node z such that $(z, \bar{2}, y) \in E$ and $\rho(z) = q'$. Furthermore, as A is normalised, $q, q' \in Q_d$. As D is well-formed, y has no other predecessor than x and z, and z has no other successor than y. We consider the DAG D' obtained by removing x and the edge $(z, \bar{2}, y)$ from D. As we removed a basic operation, D' is well-formed, it can be labelled by A (by restricting ρ to it), y is an input node, and z an output node. If D' was connected, as z is labelled by Q_d, it would be the only output node of D', by hypothesis of induction. Thus, there would be a path from y to z, which contradicts the fact that D is a DAG. Thus, D' is a disconnected DAG. As D is a connected DAG, D' is the disjoint union of two connected DAGs: D_y having y as its leftmost input node (otherwise, x would not be the leftmost input node of D) and D_z having z as an output node. These DAGs are connected, well-formed, can be labelled by A and have less than n vertices, therefore, by hypothesis of induction, they are operations. Furthermore, as z is labelled by Q_d, it is the only output node of D_z. If we restrict D to x, y and z, we obtain the DAG $D_{\overline{\mathrm{copy}}_n^2}$. Thus we have $D = (D_z \cdot_{1,2} D_{\overline{\mathrm{copy}}_n^2}) \cdot_{1,1} D_y$. Therefore, it is an operation. Furthermore, if D has an output node labelled by Q_d, as z is the only output node of D_z, it is an output node of D_y. By hypothesis of induction, D_y has thus a unique output node, and thus D has a unique output node. As there are no transition from Q_c to Q_d, D_z cannot have any input node labelled by Q_c. If D_y has an input node labelled by Q_c, by hypothesis of induction, it has a single input node, which is y. Thus, as y is not an input node in D, D has no input node labelled by Q_c.
- x has two successors y and z such that $(x, 1, y) \in E$ and $(x, 2, z) \in E$. As ρ is consistent with the transitions of A, there exists a transition $(q, (q', q''))$ in A, $\rho(y) = q'$ and $\rho(z) = q''$. Furthermore, as A is normalised, $q', q'' \in Q_c$. We consider D' the DAG obtained by removing x of D. As D is well-formed, y and z have no other predecessor in D. Thus x and y are input nodes of D', both labelled by a state of Q_c. As D is well formed and we obtained D' by removing a basic operation, D' is well formed. Furthermore, it can be labelled by A (by restricting ρ to it). If D' was connected, then by hypothesis of induction, it would have a single input node, which is not the case. Therefore, D' is not connected.

As D is connected, D' is therefore the disjoint union of two connected DAGs: D_y having y as an input node and D_z having z as an input node. D_y and D_z are two connected well-formed DAGs with less than n nodes, that can be labelled by A and which have an input node labelled by Q_c. By hypothesis of induction, they are operations with a single input node. If we restrict D to x, y and z, we obtain the DAG $D_{\mathrm{copy}_n^2}$. Thus we have $D = (D_{\mathrm{copy}_n^2} \cdot_{2,1} D_z) \cdot_{1,1} D_y$. By Definition 2, D is therefore an operation. Furthermore, D has a unique input node: x. As there is no transition from Q_c to Q_d and ρ is consistent with transitions of A, no node of D' is labelled with Q_d, and in particular, no output node. Thus, the induction hypothesis is satisfied.

As we supposed that x is the leftmost input node, there is no y such that $(x, \bar{2}, y) \in E$, otherwise, there would be a z such that $(z, \bar{1}, y) \in E$, and thus we could find an input node at the left of x.

Finally, we get that any well-formed DAG labelled by A is an operation. □

Thus, every D_i is an operation accepted by A, and therefore $D \in \mathrm{Op}(A)^k$.
Finally, we get $\mathrm{Op}(A') = \bigcup_{k \geq 0} \mathrm{Op}(A)^k$, and thus A' recognises $\mathrm{Op}(A)^*$. □

Proposition 6 *Given R a (finite or recognisable) ground stack tree rewriting system, there exists A an operation automaton such that for all stack trees t, t', $t \xrightarrow[R]{*} t' \Leftrightarrow (t, t') \in \mathcal{R}(A)$.*

Given Λ a ground stack tree transducer, there exists A an operation automaton such that for all stack trees t, t', $t \xrightarrow[\Lambda]{*} t' \Leftrightarrow (t, t') \in \mathcal{R}(A)$.

Proof This proposition is a consequence of the closure properties defined earlier and the previous proposition. Indeed, every finite set of operations is recognised by an automaton, and recognisable sets of operations are closed under union. Thus, for every (finite or recognisable) ground stack tree rewriting system R, there exists A an operation automaton such that for all stack trees t, t', $t \xrightarrow[R]{} t' \Leftrightarrow (t, t') \in r_A$. Similarly, for every ground stack tree transducer Λ, there exists A an operation automaton such that $t \xrightarrow[\Lambda]{} t' \Leftrightarrow (t, t') \in \mathcal{R}(A)$. By Theorem 1, there thus exists a normalised and distinguished automaton A_r such that $r_{A_r} = r_A$. By Proposition 5, there exists A' an operation automaton such that $\mathrm{Op}(A') = \mathrm{Op}(A_r)^*$.

Let us now show that this implies $\mathcal{R}(A') = \mathcal{R}(A)^* = r_A^*$. The last equality is straightforward, because, as $\mathcal{R}(A)$ is the relation obtained by the disjoint application of operations recognised by A, we get $r_A \subseteq \mathcal{R}(A)$ and as the iteration of the relation allows disjoint application of operations recognised by the automaton, we get $\mathcal{R}(A) \subseteq r_A^*$.

Let us start by showing the proposition for ground stack tree rewriting systems. We show by induction on the lenght of the run in R that $t \xrightarrow[R]{*} t' \Rightarrow (t, t') \in \mathcal{R}(A')$. If $t = t'$, the proposition is trivial, as by definition, every pair (t, t) is in the relation recognised by A', as the empty DAG is recognised by A'. Suppose now that

$t \xrightarrow{\ell} t'$, with $\ell \geq 1$. We consider t'' such that $t \underset{R}{\rightarrow} t''$ and $t'' \xrightarrow[R]{\ell-1} t'$. By hypothesis of induction, we have $(t'', t') \in \mathcal{R}(A')$, and thus by definition, there exists $\mathbf{D} = (D_1, \cdots, D_{|\mathbf{D}|}) \in \mathrm{Op}(A') = \mathrm{Op}(A_r)^*$ and $\mathbf{i} = (i_1, \cdots, i_{|\mathbf{D}|}) \in \mathbb{N}^*$ such that $(t'', t') \in r_{\mathbf{D}}^{\mathbf{i}}$, and $D \in \mathrm{Op}(A_r)$ and $j \in \mathbb{N}$ such that $(t, t'') \in r_D^j$. For the sake of simplicity, we suppose that $i_1 < \cdots < i_{|D|}$. We have now two possible cases:

- D is applied separately from the elements of $\mathbf{D}$. As D is in $\mathrm{Op}(A_r)$, it is also in $\mathrm{Op}(A')$. Thus, $(t, t') \in r_{\mathbf{D}'}^{\mathbf{i}'}$ with $\mathbf{D}' = (D_1, \cdots, D_k, D, D_{k+1}, D_{|\mathbf{D}|})$ and $\mathbf{i}' = (i_1, \cdots, i_k, j, i'_{k+1}, \cdots, i'_{|\mathbf{D}|})$, with $i_k < j < i_{k+1}$, and for every ℓ, $i'_\ell = i_\ell + |I_D| - |O_D|$ (to adapt the indices of application of the DAGs, as the number of leaves may have been modified by the application of D to t).
- $\mathbf{D}$ is applied to leaves produced by D. Suppose that the operations $D_{j_1}, \cdots, D_{j_\ell}$ are applied to leaves produced by D (where $j_1, \cdots, j_\ell$ are consecutive numbers). By definition of the application, there are integers $k_1, \cdots, k_\ell, h_1, \cdots, h_\ell$ such that $r_D^j \circ r_{D_{j_\ell}}^{i_{j_\ell}} \circ \cdots \circ r_{D_{j_1}}^{i_{j_1}} = r_{D'}^{\min(j, j_1)}$, with $D' = (\cdots (D \cdot_{k_\ell, h_\ell} D_{j_\ell}) \cdots D_{j_2}) \cdot_{k_1, h_1} D_{j_1}$. It is easy to see that D' is an operation. Informally, we append to D the operations applied after it. The $\min(j, i_{j_1})$ is present to ensure we apply the obtained operation at the right place, i.e. j if the leftmost input vertex of D' is a vertex of D, or i_{j_1} if it is a vertex of D_{j_1}. By construction, D' is in $\mathrm{Op}(A)^*$, thus it is in $\mathrm{Op}(A')$, and thus the vector $\mathbf{D}'$ obtained by replacing all the D_{j_k} with D' is such that $r_{\mathbf{D}'}^{\mathbf{i}'} = r_D^j \circ r_{\mathbf{D}}^{\mathbf{i}}$, where $\mathbf{i}'$ is the vector obtained by replacing all the i_{j_k} with $\min(j, i_{j_1})$ and adding $|I_D| - |O_D|$ to the indices after i_{j_l}.

Thus, in every case, we have $(t, t') \in \mathrm{R}(A')$.

The other direction is obtained similarly, showing that every vector of operations recognised by A' can be decomposed into operations of A_r applied at the positions defined by the vector $\mathbf{i}$. As we have $\mathrm{Op}(A') = \mathrm{Op}(A_r)^*$, we get that this decomposition is always possible.

Thus, $\mathrm{R}(A')$ is the transitive closure of the relation defined by R.

For a ground stack tree transducer, the demonstration is similar, differing only by the replacement of the operation D with a vector of operations. Its writing is more fastidious but the idea is unchanged. □

A consequence of this proposition is that the set of stack trees reachable in a GSTRS or a GSTT from a recognisable set of stack trees is recognisable.

Proposition 7 *Given R a GSTRS or a GSTT and L a recognisable of stack trees, the set* $L' = \left\{ t \mid \exists t_0 \in L, t_0 \xrightarrow[R]{*} t \right\}$ *is a recognisable set of stack trees.*

This property is obtained by appending the automaton recognising the transitive closure of the relation defined by the GSTRS (or the GSTT) to the automaton recognising the operation producing L from a given stack tree, as in the previous proof. It is then sufficient to observe that the resulting automaton recognises the set of operations producing L' from a given stack tree.

5 Rewriting Graphs of Stack Trees

We have now the elements to prove our main technical result: the decidability of the $\mathsf{FO}[\xrightarrow{*}]$-theory of graphs generated by GSTRS. We first formally define the graphs we study. We will consider the restriction of these graphs to sets of vertices reachable from a recognisable set of vertices, as we want to have the decidability of the reachability predicate. Notice that although we will be able to express the reachability predicate, it will not be possible to express that a stack tree is in a recognisable set.

Given R a GSTRS, we define $\mathcal{G}_R = (V, E)$ its associated graph with:

- $V = ST_n(\Gamma)$,
- $E = \{(t, a, t') \mid \exists \theta \in R_a, (t, t') \in r_\theta\}$.

We consider the restriction of $\mathcal{G}_R$ to the set of vertices reachable in R from a recognisable set of vertices L by replacing V with $\{t \mid \exists t_0 \in L, t_0 \xrightarrow[R]{*} t\}$. The graph obtained that way is denoted $\mathcal{G}_{R,L}$. We denote by GSTR$_n$ the set of the restrictions to reachable sets of vertices of graphs generated by finite GSTRS of order n, and RGSTR$_n$ those generated by recognisable GSTRS of order n.

Given Λ a GSTT, we define $\mathcal{G}_\Lambda = (V, E)$ its associated graph with:

- $V = ST_n(\Gamma)$.
- $E = \{(t, a, t') \mid (t, t') \in \mathcal{R}(\mathcal{A}_a)\}$.

As previously, we consider the restriction of this graph to a reachable set of vertices in Λ from a recognisable set of vertices. The graph obtained that way is denoted $\mathcal{G}_{\Lambda,L}$. We denote by GSTT$_n$ the set of the restrictions to reachable sets of vertices of graphs generated by GSTT of order n.

From the previous section, the transitive closure of the relation defined by a GSTRS is defined by an automaton, as well as the transitive closure of the relation defined by a GSTT. Thus, the transitive closure of a graph of GSTR$_n$ is a graph of GSTT$_n$, and this is also true for graphs of RGSTR$_n$ and of GSTT$_n$.

Theorem 2 *Graphs of GSTR$_n$, RGSTR$_n$ and of GSTT$_n$ have a decidable* $\mathsf{FO}[\xrightarrow{*}]$-*theory.*

We prove the theorem for the logic $\mathsf{FO}[\xrightarrow{*}]$. Notice that we could, without modifying the proof much, add a predicate $\xrightarrow{B^*}$ which is the restriction of the reachability predicate to a smaller alphabet B. We could as well fix an initial stack tree, thus allowing to express if a given stack tree belongs to a recognisable set (in the case of recognisable GSTRS or GSTT). Let us remark as well that it is impossible to add control state to GSTRS or to add a predicate stating that a stack tree is reachable by a word from a fixed regular language (as both are already undecidable on GTRS). It is as well impossible to consider higher-order stacks with collapse links, as the graphs generated by these stacks do not have a decidable WMSO-theory, which is crucial in our proof.

To prove this theorem, we use a remark of Colcombet and Löding in [11] stating that a graph obtained by a finite set interpretation (FSI) from a graph with a decidable MSO-theory has a decidable FO-theory. We use that remark by exhibiting a finite set interpretation of a graph of GSTR$_n$ (resp, RGSTR$_n$, GSTT$_n$) in which we added the reachability relation from a graph defined by an n-pushdown automaton, namely the graph obtained by the application of n operations of treegraph to a given finite graph. Thus, Theorem 2 is a direct consequence of the following property.

In this proposition, we use the graph $\Delta^n_{\Gamma\times\{\varepsilon,11,21,22\}} = (V, E)$ with $V = Stacks_n(\Gamma \times\{\varepsilon, 11, 21, 22\})$ and $E = \{(x, \theta, y) \mid \theta \in Ops_n(\Gamma \times \{\varepsilon, 11, 21, 22\})$. This graph is the configuration graph of the n-pushdown automaton with one state q and the transition (q, θ, q) for every basic operation θ. From [7], this graph has a decidable MSO-theory.

We recall that a *finite set interpretation* $\mathcal{I}$ of a graph $\mathcal{G}' = (V', E')$ in a graph $\mathcal{G} = (V, E)$ is a tuple of MSO-formulæ $(\delta, \phi_{a_1}, \cdots, \phi_{a_k})$ where δ is a formula with a single free order-2 variable such that $X \in V'$ if and only if $\mathcal{G} \models \delta(X)$, and for every letter a_i labelling an edge of $\mathcal{G}'$, ϕ_{a_i} is a formula with two free order-2 variables such that $X \underset{\mathcal{G}'}{\xrightarrow{a_i}} Y$ if and only if $\mathcal{G} \models \phi_{a_i}(X, Y)$.

Proposition 8 *Given R a recognisable GSTRS of order n, and L a recognisable set of n-stack trees, there exists a finite set interpretation $(\delta, \phi_{a_1}, \cdots, \phi_{a_k}, \phi_\tau)$ of the graph $\mathcal{G}_{R,L}$ in which we added the reachability relation of R, in $\Delta^n_{\Gamma\times\{\varepsilon,11,21,22\}}$.*

Given Λ an order n GSTT and L a recognisable set of n-stack trees, there exists a finite set interpretation $(\delta, \phi_{a_1}, \cdots, \phi_{a_k}, \phi_\tau)$ of the graph $\mathcal{G}_{\Lambda,L}$ in which we added the reachability relation of Λ, in $\Delta^n_{\Gamma\times\{\varepsilon,11,21,22\}}$.

To prove this proposition, we first define a coding of an n-stack tree in $ST_n(\Gamma)$ as a set of n-stacks of $Stacks_n(\Gamma \times \{\varepsilon, 11, 21, 22\})$, which is the set of paths from the root of the tree to its leaves where we encode the position of each node in the path with the 11, 21, 22. We then define three MSO-formulæ over $\Delta^n_{\Gamma\times\{\varepsilon,11,21,22\}}$: $\delta(X)$ which holds if X is a valid coding of an n-stack tree, and for every automaton A, $\psi_A(X, Y)$ and $\phi_A(X, Y)$, the first holding if there is an operation D recognised by A, if $(t, t') \in r_D$ where t and t' are the trees coded by X and Y, and the second holding if $(t, t') \in \mathcal{R}(A)$. These formulæ require that the automaton A is a normalised automaton produced by the proof of Theorem 1, as we will precise it in the proof. The finite set interpretation proving the proposition for the graph $\mathcal{G}_{R,L}$ is

$$((\delta(X) \wedge \exists Y, \psi_B([\alpha_1]_n, Y) \wedge \phi_C(Y, X)), \psi_{A_{a_1}}(X, Y), \ldots, \psi_{A_{a_k}}(X, Y), \phi_C(X, Y)),$$

where B is the automaton recognising the set of operations generating L from the n-stack tree with one node labelled with $[\alpha_1]_{n-1}$, and C is the automaton recognising the reachability relation of R.

The finite set interpretation proving the proposition for the graph $\mathcal{G}_{\Lambda,L}$ is

$$((\delta(X) \wedge \exists Y, \psi_B([\alpha_1]_n, Y) \wedge \phi_C(Y, X)), \phi_{A_{a_1}}(X, Y), \ldots, \phi_{A_{a_k}}(X, Y), \phi_C(X, Y)),$$

where B is the automaton recognising the set of operations generating L from the n-stack tree with one node labelled with $[\alpha_1]_{n-1}$, and C is the automaton recognising the reachability relation of Λ.

Let us start by defining the coding of an n-stack tree of $ST_n(\Gamma)$ as a set of n-stacks of $Stacks_n(\Gamma \times \{\varepsilon, 11, 21, 22\})$. For the sake of simplicity, in this section, the pair (α, ε) will be denoted by α.

Definition 10 Given t an n-stack tree and u a position in $\mathrm{dom}(t)$, we define

$$\mathrm{Code}(t, u) = [\mu_{t,u,1}(t(\varepsilon))\mu_{t,u,2}(t(u_{\leq 1})) \ldots \mu_{t,u,|u|}(t(u_{\leq |u|-1}))t(u)]_n,$$

where for every p an $(n-1)$-stack, t an n-stack tree, u a node of t and $i \leq |u|$, $\mu_{t,u,i}(p)$ is the n-stack p' such that $\mathrm{rew}_\alpha(\alpha, ju_i)(p, p')$ where α is the topmost element of p in Γ, j is the number of children of the node $u_1 \cdots u_{i-1}$ of t, and u_i is the i^{th} letter of u. An n-stack tree t is thus coded by the finite set of n-stacks $X_t = \{\mathrm{Code}(t, u) \mid u \in \mathrm{fr}(t)\}$, i.e. the set of codes of its leaves.

Informally, $\mathrm{Code}(t, u)$ is the n-stack read in the path from the root of t to u, in which we added to the label of each node the number of its children and the number of the son which leads to u.

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

$$\begin{aligned} X_t = \{&[[[\alpha\alpha]_1[\beta\alpha(\beta, 21)]_1]_2[[\alpha\alpha]_1[\alpha\alpha(\alpha, 11)]_1]_2[[\alpha\beta]_1]_2]_3,\\ &[[[\alpha\alpha]_1[\beta\alpha(\beta, 22)]_1]_2[[\alpha\alpha]_1[\alpha]_1[(\beta, 21)]_1]_2[[\beta\alpha]_1][\beta\alpha]_1[\beta]_1]_2]_3,\\ &[[[\alpha\alpha]_1[\beta\alpha(\beta, 22)]_1]_2[[\alpha\alpha]_1[\alpha]_1[(\beta, 22)]_1]_2[[\alpha\beta\beta]_1[\alpha\beta]_1]_2]_3\}. \end{aligned}$$

We now detail the formulæ defining the finite set interpretations. After giving useful technical formulæ, we present the formulæ δ, ϕ_A, and finally ψ_A.

5.1 Notations and Technical Formulæ

We define technical formulæ over $\Delta^n_{\Gamma\times\{\varepsilon,11,21,22\}}$ which are useful to define the set of stacks used to represent stack trees as sets of vertices of $\Delta^n_{\Gamma\times\{\varepsilon,11,21,22\}}$. To avoid long notations, in this section, we will shortcut $Ops_{n-1}(\Gamma \times \{\varepsilon\})$ by $Ops_{n-1}(\Gamma)$.

For $\alpha, \beta \in \Gamma \times \{\varepsilon, 11, 21, 22\}$, we define $\psi_{\mathrm{rew}_{\alpha,\beta}}(x, y) = x \xrightarrow{\mathrm{rew}_{\alpha,\beta}} y$. For every $k < n$, we define $\psi_{\mathrm{copy}_k}(x, y) = x \xrightarrow{\mathrm{copy}_k} y$ and $\psi_{\overline{\mathrm{copy}}_k}(x, y) = y \xrightarrow{\mathrm{copy}_k} x$. Finally, for $k \in \{1, 2\}$ and $d \leq k$, we define $\psi_{\mathrm{copy}^k_n,d}(x, y) = \exists z_1, z_2, \bigvee_{\alpha\in\Gamma} \left(x \xrightarrow{\mathrm{rew}_\alpha(\alpha,kd)} z_1 \wedge z_1 \xrightarrow{\mathrm{copy}_n} z_2 \wedge z_2 \xrightarrow{\mathrm{rew}_{(\alpha,kd)}\alpha} y\right)$. Finally, for $T_L \in \mathcal{T}_{n-1}$, we define $\psi_{T_L}(x, y) = (x = y) \wedge \psi'_L(x)$, where $\psi'_L(x)$ is a MSO-formula such that $\psi'_L(x)$ holds if and only if the topmost $(n-1)$-stack of x is in L. From Proposition 2.6 of [5], such a formula exists for any regular set L of $(n-1)$-stacks.

For $\theta \in Ops_{n-1}(\Gamma)$, $\psi_\theta(x, y)$ is satisfied if y is obtained by applying θ to x. $\psi_{\text{copy}^k_n,d}(x, y)$ is satisfied if y is obtained by adding kd to the topmost letter of x, then copying its topmost $(n-1)$-stack, and finally removing kd of its topmost letter. $\psi_{\overline{\text{copy}}^k_n,d}(x, y)$ is satisfied if $\psi_{\text{copy}^k_n,d}(y, x)$ is satisfied. These formulæ allow us to simulate the application of a stack tree operation to a stack tree over the elements of their coding.

We now give a formula checking that a given n-stack y is obtained from an n-stack x by using only the previous formulæ,

$$\text{Reach}(x, y) = \forall X, \left(\left(x \in X \wedge \forall z, z', \left(z \in X \wedge \left(\bigvee_{\theta \in Ops_{n-1}(\Gamma) \cup \mathcal{T}_{n-1}} \psi_\theta(z, z') \vee \bigvee_{k \in \{1,2\}} \bigvee_{d \le k} \psi_{\text{copy}^k_n,d}(z, z')\right)\right) \Rightarrow z' \in X\right) \Rightarrow y \in X\right).$$

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

Lemma 8 *For every n-stacks $x = [x_1 \ldots x_m]_n$ and $y = [y_1 \ldots y_{m'}]_n$, Reach$(x, y)$ is satisfied if and only if*

- *For every $i < m$, $y_i = x_i$,*
- *For every $i \in [m, m'-1]$, there is $\theta \in Ops_{n-1}(\Gamma)^*$, $\alpha \in \Gamma$ and $h \in \{11, 21, 22\}$ such that $(x_m, y_i) \in r_{rew_\alpha(\alpha,h)} \circ r_\theta$,*
- *There exists $\theta \in Ops_{n-1}(\Gamma)^*$ such that $(x_m, y_{m'}) \in r_\theta$.*

Proof Suppose that Reach(x, y) is satisfied, then we can obtain y by applying to x a sequence of operations of $Ops_{n-1}(\Gamma)$ or of the form $\text{rew}_\alpha(\alpha, h)\text{copy}_n\text{rew}_{(\alpha,h)}\alpha$, with $h \in \{11, 21, 22\}$. Thus y is of the form given in the lemma.

Suppose that y has the form given in the lemma, then y can be obtained by applying to x a sequence of operations of $Ops_{n-1}(\Gamma)$ or an operation of the form $\text{rew}_\alpha(\alpha, h)\text{copy}_n\text{rew}_{(\alpha,h)}\alpha$, with $h \in \{11, 21, 22\}$. Which proves the satisfaction of Reach(x, y). □

Corollary 2 *For every n-stack x and $\alpha \in \Gamma$, Reach$([\alpha]_n, x)$ is satisfied if and only if there exists an n-stack tree t and u one of its leaves such that $x =$ Code(t, u).*

Proof Suppose there exists an n-stack tree t and u one of its leaves such that $x =$ Code(t, u). Then

$$x = [\mu_{t,u,1}(t(\varepsilon)), \mu_{t,u,2}(t(u_{\le 1})), \cdots, \mu_{t,u,|u|}(t(u_{\le |u|-1})), t(u)]_n$$

As for every j, $t(u_{\le j})$ is in $Stacks_{n-1}(\Gamma)$, there exists $\rho_j \in Ops_{n-1}(\Gamma)^*$ such that $\rho_j([\alpha]_{n-1}, t(u_{\le j}))$. Thus, by the previous lemma, Reach$([\alpha]_{n-1}, x)$ is satisfied.

Conversely, suppose $\mathrm{Reach}([\alpha]_{n-1}, x)$ is satisfied. From Lemma 8, for every $j < |x|$, there exists $\theta \in Ops_{n-1}(\Gamma)^*$, $\beta_j \in \Gamma$ and $h_j \in \{11, 21, 22\}$ such that

$$([\alpha]_{n-1}, x_j) \in r_{\mathrm{rew}_{\beta_j}(\beta_j, h_j)} \circ r_\theta,$$

and there exists $\theta \in Ops_{n-1}(\Gamma)^*$ such that $([\alpha]_{n-1}, x_{|x|}) \in r_\theta$.

For every j, we consider that $h_j = k_j d_j$. We choose a tree domain U such that $d_1 \cdots d_{|x|-1} \in U$. We define t a tree of domain U such that for every j, every node $d_1 \cdots d_j$ has k_j children, $(t(d_1 \cdots d_{j-1}), x_j) \in r_{\mathrm{rew}_{\beta_j}(\beta_j, h_j)}$, and for every $u \in U$ which is not a $d_1 \cdots d_j$, $t(u) = [\alpha]_{n-1}$. We thus have $x = \mathrm{Code}(t, d_1 \cdots d_{|x|-1})$. $\square$

5.2 The Formula δ

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

$$\mathrm{OnlyLeaves}(X) = \forall x, x \in X \Rightarrow \mathrm{Reach}([\alpha]_n, x),$$

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

where α is a fixed letter of Γ.

Formula OnlyLeaves ensures that every element x in X encodes a node in some stack tree. TreeDom ensures that the prefix closure of the set of words u such that there exists a tree t such that $\mathrm{Code}(t, u) \in X$ is a tree domain, that the set of words $k_0 \cdots k_{m-1}$, where for every j, k_j is the arity of the node $u_1 \cdots u_j$ of t is included in this domain, and that there is no j for which $k_1 \cdots k_{j-1}(k_j + 1)$ is included in this domain (in other words, that the arity announced by the k_j is respected). Finally UniqueLabel ensures that for any two elements $x = \mathrm{Code}(t, u)$ and $y = \mathrm{Code}(t', v)$ of X, there exists an index $j \in \{1, \cdots, \min(|u|, |v|)\}$ such that for every $i < j$, $u_i = v_i$, $t(u_i) = t'(u_i)$ and the node u_i has the same arity in t and t', and $u_j \neq v_j$, i.e. for every pair of elements, the $(n - 1)$-stacks labelling their common ancestors are equal, and x and y cannot code the same leaf (because $u \neq v$). Moreover, this prevents x to code a node on the path from the root to y.

Lemma 9 *For every X finite subset of $Stacks_n(\Gamma \times \{\varepsilon, 11, 21, 22\})$, $\delta(X) \iff \exists t \in ST_n, X = X_t$.*

Proof We first show that for every n-stack tree t, $\delta(X_t)$ holds over $\Delta^n_{\Gamma \times \{\varepsilon, 11, 21, 22\}}$. By definition, for every $x \in X_t$, $\exists u \in \mathrm{fr}(t)$, $x = \mathrm{Code}(t, u)$, and then $\mathrm{Reach}([\alpha]_n, x)$ holds (by Corollary 2). Thus OnlyLeaves holds.

Let us take $x \in X_t$ such that $x = \text{Code}(t, u)$ 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 $v \in \text{fr}(t)$ such that $\forall j \leq i, v_j = u_j, v_{i+1} = 1$, and $\text{Code}(t, v) \in X_t$. Let us now take $x \in X_t$ such that $x = \text{Code}(t, u)$ with $u = u_0 \cdots u_i 1 u_{i+2} \cdots u_{|u|}$ and the node $u_0 \cdots u_i$ has 2 children in t, 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 $\text{Code}(t, v) \in X_t$. Thus TreeDom holds.

Let x and y in X_t such that $x \neq y$, $x = \text{Code}(t, u)$ and $y = \text{Code}(t, v)$, 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 $z = \text{Code}(t, u_0 \cdots u_{i-1})$, and take z' and z'' such that $\psi_{\text{copy}^2_n,1}(z, z')$ and $\psi_{\text{copy}^2_n,2}(z, z'')$. We have then $\text{Reach}(z', x)$ and $\text{Reach}(z'', y)$. And thus UniqueLabel holds. Therefore, for every stack tree t, $\delta(X_t)$ holds.

Let us now show that for every $X \subseteq Stacks_n(\Gamma \times \{\varepsilon, 11, 21, 22\})$ such that $\delta(X)$ holds, there exists $t \in ST_n$, such that $X = X_t$. As OnlyLeaves holds, for every $x = [x_1 \cdots x_{|x|}]_n \in X$, there exists t_x an n-stack tree and $u^x = u^x_1 \cdots u^x_{|u^x|}$ a node of t_x such that $x = \text{Code}(t_x, u^x)$. We define the set of words $U = \{u \mid \exists x \in X, u \sqsubseteq u^x\}$. By definition, U is closed under prefix. 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 an x such that $u1 \sqsubseteq u^x$ and the node u has 2 children in t_x, 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 thus the node $u^x_1 \cdots u^x_k$ has as many children and the same label in t_x and in t_y. 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 children 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. □

5.3 The Formula ϕ_A Associated with a Normalised Automaton Satisfying Theorem 1

First, let us recall that from Theorem 1 we can construct from any automaton an equivalent automaton satisfying the conditions of the theorem. In this subsection, we consider that A is such a normalised automaton.

Let us now explain $\phi_A(X, Y)$, which can be written as

$$\begin{aligned}\phi_A(X, Y) &= \exists Z_{q_1} \cdots Z_{q_{|Q|}}, \phi'_A(X, Y, \mathbf{Z}), \text{ with}\\ \phi'_A(X, Y, \mathbf{Z}) &= \text{Init}(X, Y, \mathbf{Z}) \wedge \text{WellFormed}(\mathbf{Z})\\ &\quad \wedge \text{Trans}(\mathbf{Z}) \wedge \text{RTrans}(\mathbf{Z}) \wedge \text{NoCycle}(\mathbf{Z})\end{aligned}$$

We detail each of the subformulæ below:

$$\text{Init}(X, Y, \mathbf{Z}) = \left(\bigcup_{q_i \in I} Z_{q_i}\right) \subseteq X \wedge \left(\bigcup_{q_i \in F} Z_{q_i}\right) \subseteq Y \wedge X \setminus \left(\bigcup_{q_i \in I} Z_{q_i}\right) = Y \setminus \left(\bigcup_{q_i \in F} Z_{q_i}\right).$$

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).

$$\mathrm{Trans}(\mathbf{Z}) = \forall s, \bigwedge_{q \in Q} \left((s \in Z_q) \Rightarrow \left(\bigvee_{K \in \Delta | q \in \mathrm{left}(K)} \exists t, t', \mathrm{Trans}_K(s, t, t', \mathbf{Z}) \vee \rho_q \right) \right),$$

where ρ_q is true if and only if q is a final state, $q \in \mathrm{left}(K)$ means that q appears in the left member of K (as in the cases below), and

$$\begin{aligned}
\mathrm{Trans}_{(q,\theta,q')}(s, t, t', \mathbf{Z}) &= (t = t') \wedge \psi_\theta(s, t) \wedge s \in Z_q \wedge t \in Z_{q'}, \\
\mathrm{Trans}_{(q,(q',q''))}(s, t, t', \mathbf{Z}) &= \psi_{\mathrm{copy}_n^2,1}(s, t) \wedge \psi_{\mathrm{copy}_n^2,2}(s, t') \wedge s \in Z_q \\
&\quad \wedge t \in Z_{q'} \wedge t' \in Z_{q''}, \\
\mathrm{Trans}_{((q,q'),q'')}(s, t, t', \mathbf{Z}) &= (t = t') \wedge \psi_{\overline{\mathrm{copy}_n^2},1}(s, t) \wedge s \in Z_q \wedge t \in Z_{q''}, \\
\mathrm{Trans}_{((q',q),q'')}(s, t, t', \mathbf{Z}) &= (t = t') \wedge \psi_{\overline{\mathrm{copy}_n^2},2}(s, t) \wedge s \in Z_q \wedge t \in Z_{q''}.
\end{aligned}$$

This formula ensures that the labelling respects the rules of the automaton in the downward direction, and that for every stack s labelled by q, if there is a rule starting by q, there is a labelled stack s' which is the result of s by at least one of those rules. And also that it is possible for a stack labelled by a final state to have no successor labelled by a state of the automaton (and as the automaton is distinguished, it cannot have a successor).

$$\mathrm{RTrans}(\mathbf{Z}) = \forall s, \bigwedge_{q \in Q} \left((s \in Z_q) \Rightarrow \left(\bigvee_{K \in \Delta | q \in \mathrm{right}(K)} \exists t, t', \mathrm{RTrans}_K(s, t, t', \mathbf{Z}) \vee \sigma_q \right) \right)$$

where σ_q is true if and only if q is an initial state, $q \in \mathrm{right}(K)$ means that q appears in the right member of K (as in the cases below), and

$$\begin{aligned}
\mathrm{RTrans}_{(q',\theta,q)}(s, t, t', \mathbf{Z}) &= (t = t') \wedge \psi_\theta(t, s) \wedge s \in Z_{q'} \wedge t \in Z_q, \\
\mathrm{RTrans}_{(q',(q,q''))}(s, t, t', \mathbf{Z}) &= (t = t') \wedge \psi_{\mathrm{copy}_n^2,1}(t, s) \wedge s \in Z_{q'} \wedge t \in Z_q, \\
\mathrm{RTrans}_{(q',(q'',q))}(s, t, t', \mathbf{Z}) &= (t = t') \wedge \psi_{\mathrm{copy}_n^2,2}(t, s) \wedge s \in Z_q \wedge t \in Z_{q'}, \\
\mathrm{RTrans}_{((q',q''),q)}(s, t, t', \mathbf{Z}) &= \psi_{\overline{\mathrm{copy}_n^2},1}(t, s) \wedge \psi_{\overline{\mathrm{copy}_n^2},2}(t', s) \wedge s \in Z_q \\
&\quad \wedge t \in Z_{q'} \wedge t \in Z_{q''}
\end{aligned}$$

This formula ensures that the labelling respects the rules of the automaton in the upward direction, and that for every stack s labelled by q, if there is a rule ending by q, there is a labelled stack s' such that s is the result of s' by at least one of those rules. And also that it is possible for a stack labelled by an initial state to have no predecessor labelled by a state of the automaton (and as the automaton is distinguished, it cannot have a predecessor).

$$\begin{aligned}\mathrm{WellFormed}(\mathbf{Z}) = \forall s, t_1, t_2, t_3, t_4,&\\ \bigwedge_{\substack{K \neq K' \in \Delta \\ q \in \mathrm{left}(K) \cap \mathrm{left}(K')}} &\neg(\mathrm{Trans}_K(s, t_1, t_2, \mathbf{Z}) \wedge \mathrm{Trans}_{K'}(s, t_3, t_4, \mathbf{Z})) \wedge \\ \bigwedge_{\substack{K \neq K' \in \Delta \\ q \in \mathrm{right}(K) \cap \mathrm{right}(K')}} &\neg(\mathrm{RTrans}_K(s, t_1, t_2, \mathbf{Z}) \wedge \mathrm{RTrans}_{K'}(s, t_3, t_4, \mathbf{Z})) \\ \wedge \bigwedge_{q,q',q''} &\forall s, t, [\mathrm{RTrans}_{(q,(q',q''))}(t, s, s, \mathbf{Z}) \Rightarrow \exists t', \mathrm{Trans}_{(q,(q',q''))}(s, t, t', \mathbf{Z})] \\ &\wedge[\mathrm{Trans}_{((q,q''),q')}(s, t, t, \mathbf{Z}) \Rightarrow \exists t', \mathrm{RTrans}_{((q,q''),q')}(t, s, t', \mathbf{Z})] \\ &\wedge[\mathrm{Trans}_{((q'',q),q')}(s, t, t, \mathbf{Z}) \Rightarrow \exists t', \mathrm{RTrans}_{((q'',q),q')}(t, t', s, \mathbf{Z})]\end{aligned}$$

This formula ensures that you never use two different transitions to label stacks from a single stack labelled with a state q (in direct and reverse direction). This forces that whenever the automaton has a non-deterministic choice, it only takes one. It furthermore ensures that it is impossible to have two nodes linked by a copy_n^2 or $\overline{\mathrm{copy}}_n^2$ transition labelled without having the third node linked with that transition labelled.

$$\begin{aligned}\mathrm{NoCycle}(\mathbf{Z}) = \forall s, &\bigvee_{q \in Q} (s \in Z_q) \\ &\Rightarrow \forall X, \left(s \in X \wedge \forall s', t, t', \left(s' \in X \wedge \left(\bigvee_K \mathrm{Trans}_K(s', t, t', \mathbf{Z})\right)\right)\right. \\ &\left.\Rightarrow \left(t \in X \wedge t' \in X\right)\right) \Rightarrow \bigcup_{q \in F} Z_q \cap X \neq \emptyset \\ &\wedge \left(s \in X \wedge \forall s', t, t', \left(s' \in X \wedge \left(\bigvee_K \mathrm{RTrans}_K(s', t, t', \mathbf{Z})\right)\right)\right. \\ &\left.\Rightarrow \left(t \in X \wedge t' \in X\right)\right) \Rightarrow \bigcup_{q \in I} Z_q \cap X \neq \emptyset\end{aligned}$$

This formula ensures that for any labelled stack, we can find a path of labelled stacks related by the transitions of the automaton which starts in an initial state, and another which ends in a final state.

Proposition 9 *Given s, t two stack trees, $\phi_A(X_s, X_t)$ holds if and only if there exists a tuple of operations $\mathbf{D} = (D_1, \cdots, D_k)$ recognised by A and a tuple of positions $\mathbf{i} = (i_1, \cdots, i_k)$ such that $(s, t) \in r_{\mathbf{D}}^{\mathbf{i}}$.*

Proof First, suppose there exist such $\mathbf{D}$ and $\mathbf{i}$. We construct a labelling of $Stacks_n(\Gamma \times \{\varepsilon, 11, 21, 22\})$ which satisfies $\phi_A(X_s, X_t)$. We take an accepting labelling ρ of the D_i by A. We will label the $Stacks_n$ according to this labelling. The idea is to cut the application of a D_i to s basic operation by basic operation and label the codes of the leaves appearing during this process by the label of the output node(s) of the basic operation considered. For example, if we consider a suboperation D_θ whose output node is y, and we obtain a tree t'' after applying D_θ at the position i to a tree t', we label Code(t'', u_i) by $\rho(y)$, where u_i is the i^{th} leaf of t''. Notice that this does not depend on the order we apply the D_j to s nor the order of the leaves we choose to apply the operations first.

We suppose that $(s, t) \in r_{\mathbf{D}}^{\mathbf{i}}$. Given a node x of an D_j, we call $\rho(x)$ its labelling.

We define the $(D_1, i_1, s_1), \cdots, (D_k, i_k, s_k)$ labelling of $Stacks_n(\Gamma \times \{\varepsilon, 11, 21, 22\})$ inductively. Let us precise that in this labelling, a single stack may have several labels.

– 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 $\{(\text{Code}(s, u_i), \rho(x))\}$, 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 $(F_1, i, s), (F_2, i, s')$ labelling, with $(s, s') \in r_{F_1}^i \circ r_\theta^i$.
– The $((((F_1 \cdot_{1,1} D_{\text{copy}_n^2}) \cdot_{2,1} F_3) \cdot_{1,1} F_2), i, s)$ labelling is the $(F_1, i, s), (F_2, i, s'), (F_3, i+1, s')$ labelling, with $(s, s') \in r_{F_1}^i \circ r_{\text{copy}_n^2}^i$.
– The $\big(\big(F_1 \cdot_{1,1} \big(F_2 \cdot_{2,1} \overline{\text{copy}_n^2}\big)\big) \cdot_{1,1} F_3, i, s\big)$ labelling is the $(F_1, i, s), (F_2, i + |I_{F_1}|, s), (F_3, i, s')$ labelling, with $(s, s') \in r_{F_1}^i \circ r_{F_2}^{i+1} \circ r_{\overline{\text{copy}_n^2}}^i$.

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 $\mathbf{Z}$ the $(D_1, i_1, s), \cdots, (D_k, i_k, s)$ labelling of $Stacks_n(\Gamma \times \{\varepsilon, 11, 21, 22\})$.

Lemma 10 *The previously defined labelling $\mathbf{Z}$ satisfies $\phi'_A(X_s, X_t, \mathbf{Z})$.*

Proof Let us first state a technical lemma which follows directly from the definition of the labelling:

Lemma 11 *Given D a reduced operation, ρ_D an accepting labelling of D, t a stack tree, $i \in \mathbb{N}$ and $1 \leq j \leq |I_D|$, one of the labels of 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).*

Furthermore, if t' is a stack tree such that $(t, t') \in r_D^i$, and $1 \leq j \leq |O_D|$, one of the labels of Code(t', u_{i+j-1}) is $\rho_D(y_j)$ (where y_j is the j^{th} output node of D).

Finally if q labels a stack s in the (D, i, t) labelling, then there is a node x of D such that $\rho_D(x) = q$.

For the sake of simplicity, let us consider for this proof that $\mathbf{D}$ is a unique reduced operation D (if it is a tuple of reduced operations, the proof is the same for every operation).

Init 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 ones can be labelled by initial states. Similarly, the nodes of X_t, and only them are labelled by final states (or not labelled).

Trans and RTrans We show by induction on the size of the operation that Trans and RTrans are satisfied. For the sake of this proof, we will consider that ρ_q (resp. σ_q) are replaced by predicates that are true if and only if the nodes considered are output (resp. input). As Init is satisfied, for an operation accepted by the automaton, ρ_q will only be true for output nodes and σ_q for input nodes.

For $D = \Box$, the formulæ are true by vacuity.

Suppose now that the formulæ are true for any operation of at most n nodes. Consider D with $n+1$ nodes and suppose $D = D_1 \cdot_{1,1} D_\theta \cdot_{1,1} D_2$. We call x the output node of D_1 and y the input node of D_2. We take an integer i and a stack tree t and consider the (D, i, t) labelling. By definition it is the $(D_1, i, t)(D_2, i, t')$ labelling with t' is a stack tree such that $(t, t') \in r^i_{D_1} \circ r^i_{D_\theta}$. We call t'' a stack tree such that $(t, t'') \in r^i_{D_1}$. By definition, we thus have $(t'', t') \in r^i_{D_\theta}$. By Lemma 11, we have $s_1 = \text{Code}(t'', u_i)$ labelled by $\rho(x)$ and $s_2 = \text{Code}(t', u_i)$ labelled by $\rho(y)$. As furthermore, we have $(s_1, s_2) \in r_\theta$, $\text{Trans}_{(\rho(x),\theta,\rho(y))}(s_1, s_2, s_2, \mathbf{Z})$ and $\text{RTrans}_{(\rho(x),\theta,\rho(y))}(s_2, s_1, s_1)$ are true. Thus Trans and RTrans are true for the (D, i, t) labelling, as by hypothesis of induction they hold for all the nodes of the (D_1, i, t) and (D_2, i, t') labellings, x is the only output node of D_1 and y the only input node of D_2.

The other cases for the decomposition of D are similar, and thus omitted. Thus, Trans and RTrans are satisfied for any (D, i, t) labelling.

WellFormed Let us now show that the labelling satisfies WellFormed. We show it by induction. We furthermore show that if a stack s is labelled by an input node x, there is no $K \in \Delta$ with $\rho(x) \in \text{right}(K)$ and no t, t' such that $\text{RTrans}_K(s, t, t', \mathbf{Z})$, and that if a stack s is labelled by an output node x, there is no $K \in \Delta$ with $\rho(x) \in \text{left}(K)$ and no t, t' such that $\text{Trans}_K(s, t, t', \mathbf{Z})$.

If $D = \Box$ or $D = D_{T_L}$, the result is immediate.

Suppose that $D = (D_1 \cdot_{1,1} (D_{\text{copy}^2_n} \cdot_{2,1} D_3)) \cdot_{1,1} D_2$. By induction, WellFormed holds for the (D_1, i, t), (D_2, i, t') and $(D_3, i+1, t')$ labellings with $(t, t') \in r^i_{D_1} \circ r^i_{\text{copy}^2_n}$, and we consider t'' such that $(t'', t') \in r^i_{\text{copy}^2_n}$. We call x the only output node of D_1, y and z the only input nodes of D_2 and D_3, $s_1 = \text{Code}(t'', u_i)$, $s_2 = \text{Code}(t', u_i)$ and $s_3 = \text{Code}(t', u_{i+1})$. First, observe that we have $\text{Trans}_{(\rho(x),(\rho(y),\rho(z)))}(s_1, s_2, s_3, \mathbf{Z})$, $\text{RTrans}_{(\rho(x),(\rho(y),\rho(z)))}(s_2, s_1, s_1, \mathbf{Z})$ and $\text{RTrans}_{(\rho(x),(\rho(y),\rho(z)))}(s_3, s_1, s_1, \mathbf{Z})$, by definition of the labelling. By hypothesis of induction, there are no stacks s, s' and no transition K with $\rho(x) \in \text{left}(K)$ (resp. $\rho(y) \in \text{right}(K)$, $\rho(z) \in \text{right}(K)$) such that $\text{Trans}_K(s_1, s, s', \mathbf{Z})$ (resp.

$\mathrm{RTrans}_K(s_2, s, s', \mathbf{Z})$, $\mathrm{RTrans}_K(s_3, s, s')$) holds in the (D_1, i, t) labelling (resp. the (D_2, i, t') or $(D_3, i+1, t')$ labellings). Suppose there are stacks s, s', s'' labelled in the (D, i, t) labelling such that $\mathrm{Trans}_K(s, s', s'', \mathbf{Z})$ holds, such that s, s', s'' are not all labelled in the same D_i. As A is normalised, all stacks labelled in D_2 and D_3 are labelled with states of Q_c, and have at least 1 $(n-1)$-stack more than stack labelled with states of Q_c in D_1. Moreover, as s_2 is obtained as the left copy of s_1 and s_2 as the right copy of s_1, it is not possible to produce stacks labelled in D_2 from stacks labelled in D_3 without producing first s_1 (and vice-versa), and as A is normalised, this is impossible. Thus, if all three s, s', s'' are labelled in Q_c, then s is labelled in D_1 and K is a copy_n^d transition. If s is labelled in Q_d, then the only possibility for K is also to be a copy_n^d transition. Suppose it is a copy_n^1 transition, then $s' = s''$. Suppose s' is labelled in D_2 (the case for D_3 being symmetric). It is thus obtained from s by a copy_n^1 operation. But it is also obtained from s_2 which has more $(n-1)$-stacks than s and is obtained from s_1 as the left copy of a copy_n^2. Thus s_2 have at least as many $(n-1)$-stacks as s', and to obtain s' from s_2 we must first annul that binary copy with a $\overline{\mathrm{copy}}_n^2$, which is impossible as A is normalised. Thus we have K is a copy_n^2 transition, s is labelled in D_1, s' in D_2 and s'' in D_3. They are thus θ_1, θ_2, θ_3 such that $(s, s_1) \in r_{\theta_1}$, $(s_2, s') \in r_{\theta_2}$ and $(s_3, s'') \in r_{\theta_3}$. Thus we have $D_{\theta_1} \cdot_{1,1} ((D_{\mathrm{copy}_n^2} \cdot_{2,1} D_{\theta_3}) \cdot_{1,1} D_{\theta_2})$ equivalent to $D_{\mathrm{copy}_n^2}$. Thus as A is normalised, θ_1, θ_2 and θ_3 can only be single tests or empty operations, but as there are no state that can be at the left (resp. right) of both a test and a non-test transition, we get that θ_1, θ_2 and θ_3 are empty operation, and thus that $s = s_1$, $s' = s_2$, $s'' = s_3$ and $K = (\rho(x), (\rho(y), \rho(z)))$. Similarly, if $\mathrm{RTrans}_K(s, s', s'')$ for s, s', s'' not labelled in the same DAG, then $K = (\rho(x), (\rho(y), \rho(z)))$ and $s = s_2$ or $s = s_3$ and $s' = s'' = s_1$. Observe as well that as we added a full transition, we did not add any x, y such that there is a K with $\mathrm{RTrans}_K(x, y, y, \mathbf{Z})$ but no z such that $\mathrm{Trans}_K(y, z, x)$ or $\mathrm{Trans}_K(y, x, z)$ (or the converse). Thus WellFormed holds for the (D, i, t) labelling. Finally, as input nodes of D were input nodes of D_1 (resp. output nodes of D were output nodes of D_2 and D_3), and we did not add them any incoming edge (resp. output edge), the same reasoning apply to show there is no RTrans_K (resp. Trans_K)) true with them as first variable, and thus the induction hypothesis holds.

For $D = D_2 \cdot_{1,2} (D_1 \cdot_{1,1} D_{\overline{\mathrm{copy}}_n^2})) \cdot_{1,1} D_3$, the case is similar (by exchanging roles of copy_n^d and $\overline{\mathrm{copy}}_n^d$ and so on).

Suppose that $D = D_1 \cdot_{1,1} D_\theta \cdot_{1,1} D_2$, and call x the output node of D_1 and y the input node of D_2. We take t'' such that $(t, t'') \in r_{D_1}^i$ and t' such that $(t'', t') \in r_\theta^i$, and call $s_1 = \mathrm{Code}(t'', u_i)$ and $s_2 = \mathrm{Code}(t', u_i)$. By definition of the labelling, $\mathrm{Trans}_{(\rho(x),\theta,\rho(y))}(s_1, s_2, s_2, \mathbf{Z})$ and $\mathrm{RTrans}_{(\rho(x),\theta,\rho(y))}(s_2, s_1, s_1, \mathbf{Z})$ hold. By hypothesis of induction, WellFormed holds for the (D_1, i, t) and (D_2, i, t') labellings, and there are no K, and no s, s' labelled in the (D_1, i, t) labelling (resp. the (D_2, i, t') labelling) such that $\mathrm{Trans}_K(s_1, s, s', \mathbf{Z})$ holds (resp. $\mathrm{RTrans}_K(s_2, s, s', \mathbf{Z})$). Suppose there are stacks s, s', s'' labelled in the (D, i, t) labelling such that $\mathrm{Trans}_K(s, s', s'', \mathbf{Z})$ holds and s, s', s'' are not all labelled in the same D_i. Suppose that s is labelled by z_2 in D_2 and s' by z_1 in D_1. Then, there is a path from z_1 to z_2 labelled by w which contains θ. But as from $\rho(z_2)$ it is possible to take K, it is thus possible to label a DAG containing w followed by K, which is a non-single test word that can leave s' unchanged. As A is normalised, this is impossible. Thus, s is labelled by z_1 in D_1 and s', s'' by z_2 and z_3 in D_2. Call θ' the operation of K. There is a path from z_1 to z_2 labelled by an operation w that contains θ, and such that $r_w = r_{\theta'}$. We thus have $w = w_1 \cdot \theta \cdot w_2$ for some operation words w_1, w_2. Suppose θ' is a copy_n^d operation. Then s' has exactly one $(n-1)$-stack more than s and their topmost are equal. As θ is a non-test and non-tree operation, it must be inverted in w to get s' from s. Thus w contains a non-single test operation containing the identity, which is impossible as A is normalised. If θ' is a $\overline{\mathrm{copy}}_n^d$ operation we get a similar contradiction. Thus θ' is a stack operation. Suppose w contains tree operations. As A is normalised, you only can have $\overline{\mathrm{copy}}_n^d$ followed by the same number of copy_n^d, as θ' does not modify the number of $(n-1)$-stacks. Thus $\rho(z_1)$ is in Q_d and $\rho(z_2)$ is in Q_c. But the only transition between Q_d and Q_c are labelled with copy_n^d operations, so this is impossible. Thus w does not contain any tree operation and is thus in Red (as A is normalised). If $\theta \neq \theta'$ and $\theta = \mathrm{copy}_i$ (resp. $\overline{\mathrm{copy}}_j$) or $\theta' = \mathrm{copy}_j$ (resp. $\overline{\mathrm{copy}}_j$), we get a similar contradiction than for the copy_n^d case. If $\theta = \theta' = \mathrm{copy}_i$, similarly to the above case for D, we get that w_1 and w_2 can only be empty operations, thus $z_1 = x$, $z_2 = y$, and $K = (\rho(x), \theta, \rho(y))$. Suppose $\theta = \mathrm{rew}_{\alpha,\beta}$ and $\theta' = \mathrm{rew}_{\gamma}\delta$. As w is in Red, it cannot contain non-single test factors containing the identity, thus it only contains tests and $\mathrm{rew}_{\alpha'}\beta'$ operations. But in a word of Red, there cannot be two $\mathrm{rew}_{\alpha'}\beta'$ operations not separated by a copy_i or a $\overline{\mathrm{copy}}_i$. Moreover as no state can be at the left of the right of both a test and a non-test transition, we get that $w = \mathrm{rew}_{\alpha,\beta}$ and is equivalent to $\mathrm{rew}_{\gamma}\delta$. Thus $z_1 = x$, $z_2 = y$, and $K = (\rho(x), \theta, \rho(y))$. Thus in every case, we have that if $\mathrm{Trans}_K(s, s', s'')$ with s, s', s'' labelled in different D_i, $s = s_1$, $s' = s'' = s_2$ and $K = (\rho(x), \theta, \rho(y))$. Similarly, if $\mathrm{RTrans}_K(s, s', s'')$ with s, s', s'' labelled in different D_i, we get $s = s_2$, $s' = s'' = s_1$ and $K = (\rho(x), \theta, \rho(y))$. Observe as well that as we added a full transition, we did not add any x, y such that there is a K with $\mathrm{RTrans}_K(x, y, y, \mathbf{Z})$ but no z such that $\mathrm{Trans}_K(y, z, x)$ or $\mathrm{Trans}_K(y, x, z)$ (or the converse). Thus WellFormed holds for the (D, i, t) labelling. Finally, as input nodes of D were input nodes of D_1 (resp. output nodes of D were output nodes of D_2 and D_3), and we did not add them any incoming edge (resp. output edge), the same reasoning apply to show there is no RTrans_K (resp. Trans_K)) true with them as first variable, and thus the induction hypothesis holds.

Thus WellFormed holds for the (D, i, t) labelling.

NoCycle Finally, let us show that the labelling satisfies NoCycle. Take s a stack labelled in the (D, i, t) labelling of $Stacks_{n-1}$ with the label of a node x. We show by induction on the size of D that there exists a path of stacks related by basic operations from a stack s' labelled with a state labelling an input node of D to s and a path of stacks related by basic operations from s to a stack s'' labelled with a state labelling an output node of D.

If $D = \square$, the result is immediate, both paths being empty.

Suppose $D = D_1 \cdot_{1,1} D_\theta \cdot_{1,1} D_2$, and x is a node of D_1 (the case x is a node of D_2 is symmetric and thus omitted). By hypothesis of induction, there exists s' labelled by an input node of D_1 such that there is a path from s' to s, and s'' labelled by the only output node of D_1 such that there is a path from s to s''. By hypothesis of induction, for s_1 labelled by the only input node of D_2, there exists s_2 labelled by an output node of D_2 such that there is a path from s_1 to s_2. Furthermore, by definition of the labelling, we have $(s'', s_1) \in r_\theta$, and thus, there is a path from s to s_2.

The other cases for the decomposition of D are similar and left to the reader.

Thus, as all its sub-formulæ are true, $\phi'_A(X_s, X_t, \mathbf{Z})$ is true with the described labelling $\mathbf{Z}$. And thus $\phi_A(X_s, X_t)$ is true. $\square$

Suppose now that $\phi_A(X_s, X_t)$ is satisfied. We take a labelling $\mathbf{Z}$ that satisfies the formula $\phi'_A(X_s, X_t, \mathbf{Z})$, minimal in the number of nodes labelled. We construct the following graph D:

$$\begin{aligned}
V_D &= \{(x,q) \mid x \in Stacks_n(\Gamma \times \{\varepsilon, 11, 21, 22\}) \wedge x \in Z_q\}\\
E_D &= \{((x,q), \theta, (y,q')) \mid (\exists\theta, (q,\theta,q') \in \Delta \wedge \psi_\theta(x,y))\}\\
&\quad \cup\{((x,q),1,(y,q')), ((x,q),2,(z,q'')) \mid (q,(q',q'')) \in \Delta\\
&\quad \wedge\psi_{\mathrm{copy}^2_n,1}(x,y) \wedge \psi_{\mathrm{copy}^2_n,2}(x,z)\}\\
&\quad \cup\{((x,q),\bar{1},(z,q'')), ((y,q'),\bar{2},(z,q'')) \mid ((q,q'),q'') \in \Delta\\
&\quad \wedge\psi_{\mathrm{copy}^2_n,1}(z,x) \wedge \psi_{\mathrm{copy}^2_n,2}(z,y)\}\\
&\quad \cup\left\{\left((x,q), \mathrm{copy}^1_n, (y,q')\right) \mid \left(q, \mathrm{copy}^1_n, q'\right) \in \Delta \wedge \psi_{\mathrm{copy}^1_n,1}(x,y)\right\}\\
&\quad \cup\left\{\left((x,q), \overline{\mathrm{copy}}^1_n, (y,q')\right) \mid \left(q, \overline{\mathrm{copy}}^1_n, q'\right) \in \Delta \wedge \psi_{\mathrm{copy}^1_n,1}(y,x)\right\}
\end{aligned}$$

Lemma 12 *D is a disjoint union of connected well-formed DAGs $D_1, \cdots, D_k$.*

We recall that a well-formed DAG is a DAG where for every nodes x, y, if there is an edge between x and y, then the subDAG obtained by restricting the DAG to x, y, the successors of x and the predecessors of y is a DAG representing a basic operation.

Proof First, we show that D is a DAG. If not, there exists $(x,q) \in V$ such that $(x,q) \xrightarrow{w} (x,q)$, with w a non-empty sequence of operations.

Suppose that w is composed of $Ops_{n-1} \cup \mathcal{T}_{n-1}$ operations. As WellFormed is satisfied, we thus have a cycle $(x,q) \xrightarrow{w_1} (x_1,q_1) \cdots (x_{k-1},q_{k-1}) \xrightarrow{w_k} (x,q)$ without any other edge touching the (x_i,q_i). As NoCycle is satisfied, there is a q_i initial and a q_j final. As the automaton is distinguished, there is no transition from q_j and no transition ending in q_i, thus we have a contradiction, and such a cycle is impossible.

Thus, there is a tree operation in w. Suppose that the first tree operation appearing in w is a $\overline{\mathrm{copy}}^d_n$. As this operation diminishes the number of $(n-1)$-stacks in x, and w starts from (x,q) and ends in (x,q), it has to increase again the number of $(n-1)$-stacks. This can only be done by a $\mathrm{copy}^{d'}_n$ operation. As A is normalised, $\overline{\mathrm{copy}}^d_n$ transition starts in Q_d and $\mathrm{copy}^{d'}_n$ transition ends in Q_c. Thus states of both Q_c and Q_d appears on the cycle, and in particular, we have an edge from a state of Q_c to a state of Q_d, and thus a transition from Q_c to Q_d in A. As A is normalised, this is impossible. If w starts by a copy^d_n operations, we reach the same contradiction.

Thus, D is a DAG.

Now, let us suppose that there are (x, q) and (y, q') nodes such that there is an edge $(x, q) \xrightarrow{\theta} (y, q')$ and the restriction of D to (x, q), its successors, (y, q') and its predecessors is not a basic operation. Suppose for example that there is a node (z, q'') distinct from (y, q') such that $(x, q) \xrightarrow{\theta'} (z, q'')$ and θ and θ' are operations in $Ops_{n-1} \cup \mathcal{T}_{n-1} \cup \{\mathrm{copy}_n^1, \overline{\mathrm{copy}}_n^1\}$. Thus, by definition of D, we have $\psi_\theta(x, y)$, $\psi_{\theta'}(x, z)$, $y \in Z_{q'}$ and $z \in Z_{q''}$. Thus, we have $\mathrm{Trans}_{(q,\theta,q')}(x, y, y, \mathbf{Z})$ and $\mathrm{Trans}_{(q,\theta',q'')}(x, z, z, \mathbf{Z})$ which hold. As WellFormed holds, this is impossible.

Suppose there is a node (z, q'') such that $(x, q) \xrightarrow{1} (z, q'')$. By construction of D, there is a transition $(q, (q'', q'''))$ in the automaton, and thus as WellFormed is satisfied, there exists a node (z', q''') such that $\mathrm{Trans}_{(q,(q'',q'''))}(x, z, z', \mathbf{Z})$ holds. But as we have $\mathrm{Trans}_{(q,\theta,q')}(x, y, y, \mathbf{Z})$, this is impossible (by WellFormed).

The other cases for the neighbourhood of x and y which gives a non-basic operation wield similar contradictions and are left to the reader.

Thus, the restriction of D to (x, q), its successors, (y, q') and its predecessors is a basic operation. Therefore, D is well-formed. □

Lemma 13 *Each D_i is recognised by A*

Proof If a node (x, q) of D_i is an output node, by definition, there is no y and q' such that there is a basic operation θ such that $(q, \theta, q') \in \Delta_A$, $(x, y) \in r_\theta$, and y is labelled by q' (or $(q, (q', q'')) \in \Delta$ or $((q, q'), q'')) \in \Delta$, and the corresponding nodes). As Trans is satisfied, if q is not final, there exist such y and q'. Thus q is final. Similarly (using RTrans instead of Trans), we get that if a node (x, q) is an input node then q is initial.

Furthermore, by construction of the edges, and because Trans is satisfied, by labelling a node (x, q) by q we obtain a labelling respecting the transitions of A.

Therefore D_i is accepted by A. □

Thus, every D_i is a well-formed DAG accepted by A. From Lemma 7, we get that D_i is an operation.

Lemma 14 *t is obtained by applying the D_i to disjoint positions of s.*

Proof We show by induction on the size of D that $(s, t') \in r_D^j$ if and only if $X_{t'} = X_s \cup \{x \mid (x, q) \in O_D\} \backslash \{x \mid (x, q) \in I_D\}$, where j is the number of the leaf coded by the leftmost $(x, q) \in I_D$ in s:

– 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 $(s, r) \in r_F^j$, 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 $(r, r') \in r_\theta^j$ 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 $(s, t') \in$

$r^j_F \circ r^j_\theta \circ r^j_G$ and thus $(r', t') \in r^j_G$. Thus, $X_{t'} = X_s \cup \{x \mid (x,q) \in O_G\} \backslash \{x \mid (x,q) \in I_F\} = X_s \cup \{x \mid (x,q) \in O_D\} \backslash \{x \mid (x,q) \in I_D\}$.

Conversely, suppose we have $X_{t'} = X_s \cup \{x \mid (x,q) \in O_D\} \backslash \{x \mid (x,q) \in I_D\}$. Thus $X_{t'} = ((X_s \cup \{y\} \backslash \{x \mid (x,q) \in I_F\}) \backslash \{y\} \cup \{z\}) \cup \{x \mid (x,q) \in O_G\} \backslash \{z\}$, with (y, q') the only output node of F and (z, q'') the only input node of G. We call $X_r = X_s \cup \{y\} \backslash \{x \mid (x,q) \in I_F\}$, and $X_{r'} = X_r \backslash \{y\} \cup \{z\}$. By induction hypothesis, X_r and $X_{r'}$ code stack trees r and r' such that $(s, r) \in r^j_F$ and $(r', t') \in r^j_G$. As we have $D_\theta = \{((x, q'), \theta, (y, q''))\}$, we have $\psi_\theta(x, y)$, by construction, and thus $(r, r') \in r^j_\theta$. Thus $(s, t') \in r^j_D$.

The other cases are similar and are thus left to the reader. We then construct this way successively t_1, t_2, and so on such that $(s, t_1) \in r^{i_1}_{D_1}$, $(t_1, t_2) \in r^{i_2}_{D_2}$, etc, where i_k is the number of the leaf coded by the leftmost $(x, q) \in D_k$ in t_{k-1}. As the D_i are disjoint, and in particular their input nodes are disjoint, we get that the D_k are applied disjointly as the leaf coded by the input node of a D_k is not coded by an input node of any other $D_{k'}$. Finally, we obtain t as we have $X_{t_{|\mathbf{D}|}} = X_s \backslash (\bigcup_{k \le |\mathbf{D}|} I_D) \cup \bigcup_{k \le |\mathbf{D}|} O_D = X_t$ and thus prove the lemma. □

We have proved both directions: for every n-stack trees s and t, there exists a tuple of operations $\mathbf{D}$ recognised by A and a tuple of positions $\mathbf{i}$ such that $(s, t) \in r^{\mathbf{i}}_{\mathbf{D}}$ if and only if $\phi_A(X_s, X_t)$. □

5.4 The Formula ψ_A Associated with a Normalised Automaton A

Given two stack trees s, t, the formula $\psi_A(X_s, X_t)$ holds if there exists an operation D recognised by A such that $(s, t) \in r_D$. Thus, it differs from ϕ_A only by the fact that D is a unique operation, as $\phi_A(X_s, X_t)$ holds if there exists a tuple of operations $\mathbf{D}$ recognised by A such that $(s, t) \in r_{\mathbf{D}}$. The formula is defined as follows:

$$\psi_A(X, Y) = \exists Z_{q_1}, \cdots, Z_{q_{|Q|}}, \phi'_A(X, Y, \mathbf{Z}) \wedge \text{Connected}(\mathbf{Z}), \text{ with}$$

$$\begin{aligned}\text{Connected}(\mathbf{Z}) = \forall x, y, x \in \bigcup_q Z_q \wedge y \in \bigcup_q Z_q \Rightarrow \forall W, \Bigg(x \in W \wedge \Bigg(\forall z, z', z'', z \in W \\ \wedge \left(\bigcup_K \text{Trans}_K(z, z', z'', \mathbf{Z}) \vee \text{RTrans}_K(z, z', z'', \mathbf{Z}) \right) \\ \Rightarrow (z' \in W \wedge z'' \in W) \Bigg) \Bigg) \Rightarrow y \in W\end{aligned}$$

Informally, this formula states that between any two nodes labelled, one can find a path of labelled nodes related by transitions of the automaton.

Proposition 10 *Given s and t two stack trees, the formula $\psi_A(X_s, X_t)$ is satisfied if and only if there exists an operation D recognised by A such that $(s, t) \in r_D$.*

Proof As ψ_A is the conjunction of ϕ_A and Connected, we have as in the previous section that $\psi_A(X_s, X_t)$ is satisfied if and only if there exists $\mathbf{D}$ a tuple of operations recognised by A and a tuple of positions $\mathbf{i}$ such that $(s,t) \in r_{\mathbf{D}}^{\mathbf{i}}$. To obtain our property, it is sufficient to show that ψ_A is satisfied if and only if we moreover have that $\mathbf{D}$ is a singleton.

Suppose first that there exists D recognised by A such that $(s,t) \in r_D^i$. With the labelling defined in the previous subsection, Lemma 10 holds. It suffices to add the following lemma to conclude that $\psi_A(X_s, X_t)$ is satisfied:

Lemma 15 *If* $\mathbf{D}$ *is a single operation* D*, the labelling* $\mathbf{Z}$ *defined in the previous subsection satisfies* Conneted$(X_s, X_t, \mathbf{Z})$.

Proof We show the result by induction on the size of D.

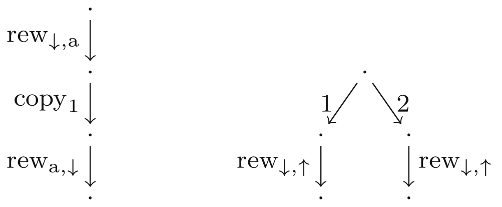
If $D = \Box$, then there is only one node labelled in the (D, i, t) labelling and thus Connected holds.

Suppose $D = D_1 \cdot_{1,1} D_\theta \cdot_{1,1} D_2$ has n nodes, has a labelling ρ consistent with A, and that the result holds for any operation with less than n nodes. Take s, s' two stacks labelled in the (D, i, t) labelling, and suppose they have been labelled by the nodes x and y of D. If x and y are in the same subDAG, e.g. D_1, then by definition, as the (D, i, t) labelling is the (D_1, i, t), (D_2, i, t') labelling with $(t, t') \in r_{D_1}^i \circ r_{D_\theta}^i$, s and s' are in the (D_1, i, t) labelling and thus by hypothesis of induction, there is an undirected path between them. If x is in D_1, and y is in D_2, we consider z_1 the output node of D_1 and z_2 the input node of D_2, t' such that $(t, t') \in r_{D_1}^i \circ r_{D_\theta}^i$ and t'' such that $(t, t'') \in r_{D_1}^i$. Observe we have $(t'', t') \in r_{D_\theta}^i$. By Lemma 11, Code$(t'', u_i)$ is labelled by $\rho(z_1)$ in the (D_1, i, t) labelling and Code(t', u_i) is labelled by $\rho(z_2)$ in the (D_2, i, t') labelling. Thus, by hypothesis of induction, there is an undirected path of labelled nodes between s and Code(t'', u_i) and one between Code(t', u_i) and s'. As furthermore, there is a transition $K = (\rho(z_1), \theta, \rho(z_2))$ in A, Trans$_K(z_1, z_2, z_2, \mathbf{Z})$ holds, and thus there is an undirected path between s and s'. Thus Connected holds.

The other cases are similar and left to the reader. $\Box$

Suppose now that $\psi_A(X_s, X_t)$ holds. As previously, we construct the graph D from a minimal labelling satisfying the formula. Lemmas 12, 13 and 14 hold. It is sufficient to modify the proof of Lemma 12 to show that D is furthermore a connected DAG. We chose the vertices (x, q) and (y, q') in D. By definition of D, we have $x \in Z_q$ and $y \in Z_{q'}$. As Connected($\mathbf{Z}$) holds, there is a sequence $x = x_1, x_2, \cdots, x_k = y$ such that for any i, x_i is labelled with a state q_i, with $q_1 = q$ and $q_k = q'$, and there is a transition K and a node z_i such that Trans$_K(x_i, x_{i+1}, z_i, \mathbf{Z})$ or RTrans$_K(x_i, x_{i+1}, z_i, \mathbf{Z})$ holds. In the first case, by construction of D, we have an edge (x_i, θ, x_{i+1}) (or $(x_i, 1, x_{i+1})$ or other similar cases depending on the transition K), and in the second case we have an edge (x_{i+1}, θ, x_i). Thus, there is an undirected path between (x, q) and (x, q'). Therefore D is a connected DAG. $\Box$

The three formulæ defined in this section allow to define the finite set interpretations announced at the beginning of the section, which proves Theorem 2.



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

Fig. 7 Rules of the rewriting system

6 Example of a Language

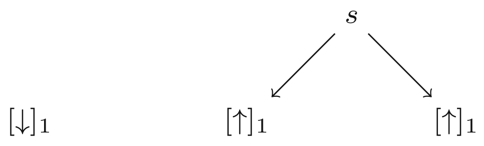
We can see a rewriting graph as a language acceptor in a classical way by defining some initial and final nodes and labelling the edges. We present here an example of a language recognised by a stack tree rewriting system. The recognised language is $\bigcup_{u\in\Sigma^*} u \quad u$. We recall that the *shuffle* of two words $u \quad v$ is the set of words inductively defined by $u \quad v = u_1(u_2 \cdots u_{|u|} \quad v) \cup v_1(u \quad v_2 \cdots v_{|v|})$, and $\varepsilon \quad \varepsilon = \{\varepsilon\}$. 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. 7. We consider, for every $a \in \Sigma$, $R_a = \{D_a\}$, and $R_\varepsilon = \{Dupl\} \cup \{P_a \mid a \in \Sigma\}$, where R_ε is a set of rules which can be taken without reading a letter of the recognised word.

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. 8.

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

$$\bigcup_{a_1\cdots a_n\in\Sigma} P_{a_1}\cdots P_{a_n}\cdot Dupl\cdot((D_{a_n}\cdots D_{a_1}) \quad (D_{a_n}\cdots D_{a_1})).$$

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 whose two leaves are labelled by $[a_1 \ldots a_n \uparrow]_1$, 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 $[b_1 \ldots b_m \uparrow]_1$ and



(a) The initial tree (b) A final tree; s is an arbitrary 1-stack

Fig. 8 The initial and final trees

$[c_1 \dots c_\ell \uparrow]_1$, 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 $(D_{a_n} \cdots D_{a_1}) \quad (D_{a_n} \cdots D_{a_1})$. And this is the only way to reach a final tree.

To obtain the language we announced at the start, we just have to remark that every P_a and *Dupl* operations are taken without reading a letter, and that D_a is taken while reading a. Thus, denoting $\lambda(w)$ the word read by a calculus of trace w, we get that, if w is an accepting trace (so is of the form described earlier), we get $\lambda(w) = (a_n \cdots a_1) \quad (a_n \cdots a_1)$, and we indeed recognise $\bigcup_{u \in \Sigma^*} u \quad u$.

7 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 orders with the corresponding orders of the pushdown and tree-automatic hierarchies (the later defined from the first in [11], and currently only considered there, to the knowledge of the author). 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 [19] and that those recognised by tree-automatic structures form the class ETIME [16]. However there is to our knowledge no characterisation of the languages recognised by ground tree rewriting systems. An other interesting point consists in finding other representation of the classes of graphs GSTR$_n$, RGSTR$_n$ and GSTT$_n$. At order 1, RGSTR$_1$ is the set of graphs defined by finite VRP (Vertex Replacement and Asynchronous product) systems of equations [10].[4] We conjecture that RGSTR$_n$ is the set of graphs defined by VRP systems of equations defined by a graph of the order $n-1$ of the prefix-recognisable hierarchy. 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 2 to this model.

Acknowledgements The author would like to address his thanks to Antoine Meyer for the help he provided in the realisation of this work, and for fruitful discussions. The author also thanks the anonymous reviewer for his invaluable comments.

[4]In the PhD of Colcombet, these VRP operations are known as VRA operations

References

1. Barthelmann, K.: When Can an Equational Simple Graph Be Generated by Hyperedge Replacement? In: MFCS, LNCS, vol. 1450, pp. 543–552. Springer (1998)
2. Blumensath, A., Grädel, E.: Finite presentations of infinite structures: Automata and interpretations. Theory Comput. Syst. **37**(6), 641–674 (2004)
3. Broadbent, C.H., Carayol, A., Hague, M., Serre, O.: A Saturation Method for Collapsible Pushdown Systems. In: ICALP (2), LNCS, vol. 7392, pp. 165–17. Springer (2012)
4. Broadbent, C.H., Carayol, A., Ong, C.H.L., Serre, O.: Recursion Schemes and Logical Reflection. In: LICS, LNCS, pp. 120–129. IEEE (2010)
5. Carayol, A.: Regular Sets of Higher-Order Pushdown Stacks. In: MFCS, LNCS, vol. 3618, pp. 168–179. Springer (2005)
6. Carayol, A., Serre, O.: Collapsible Pushdown Automata and Labeled Recursion Schemes: Equivalence, Safety and Effective Selection. In: LICS, LNCS, pp. 165–174. IEEE (2012)
7. Carayol, A., Wöhrle, S.: The Caucal Hierarchy of Infinite Graphs in Terms of Logic and Higher-Order Pushdown Automata. In: FSTTCS, LNCS, vol. 2914, pp. 112–123. Springer (2003)
8. Caucal, D.: On Infinite Transition Graphs Having a Decidable Monadic Theory. In: ICALP, LNCS, vol. 1099, pp. 194–205. Springer (1996)
9. Caucal, D.: On Infinite Terms Having a Decidable Monadic Theory. In: MFCS, LNCS, vol. 2420, pp. 165–176. Springer (2002)
10. Colcombet, T.: On Families of Graphs Having a Decidable First Order Theory with Reachability. In: ICALP, LNCS, vol. 2380, pp. 98–109. Springer (2002)
11. Colcombet, T., Löding, C.: Transforming structures by set interpretations. logical methods in computer science (LMCS) **3**(2) (2007)
12. Dauchet, M., Heuillard, T., Lescanne, P., Tison, S.: Decidability of the confluence of finite ground term rewrite systems and of other related term rewrite systems. Inf. Comput. **88**(2), 187–201 (1990)
13. Dauchet, M., Tison, S.: The Theory of Ground Rewrite Systems is Decidable. In: LICS, LNCS, pp. 242–248. IEEE Computer Society (1990)
14. Dauchet, M., Tison, S., Heuillard, T., Lescanne, P.: Decidability of the Confluence of Ground Term Rewriting Systems. In: LICS, LNCS, pp. 353–359. IEEE Computer Society (1987)
15. Knapik, T., Niwinski, D., Urzyczyn, P.: Higher-Order Pushdown Trees are Easy. In: FoSSaCS, LNCS, vol. 2303, pp. 205–222. Springer (2002)
16. Meyer, A.: Traces of Term-Automatic graphs. ITA **42**(3), 615–630 (2008)
17. Penelle, V.: Rewriting Higher-Order Stack Trees. In: CSR, LNCS, vol. 9139, pp. 364–397. Springer (2015)
18. Rabin, M.O.: Decidability of Second-Order theories and automata on infinite trees. Bull. Am. Math. Soc. **74**, 1025–1029 (1968)
19. Rispal, C.: The synchronized graphs trace the Context-Sensitive languages. Electr. Notes Theor. Comput. Sci. **68**(6), 55–70 (2002)