PREFIX REWRITING FOR NESTED-WORDS AND COLLAPSIBLE PUSHDOWN AUTOMATA

C.H. BROADBENT

LIAFA, Université Paris Diderot-Paris 7 and CNRS, Case 7014 F-75205 Paris Cedex 13, France
e-mail address: broadbent@liafa.jussieu.fr

Abstract. We introduce two natural variants of prefix rewriting on nested-words. One captures precisely the transition graphs of order-2 pushdown automata and the other precisely those of order-2 *collapsible* pushdown automata (2-CPDA). To our knowledge this is the first precise 'external' characterisation of 2-CPDA graphs and demonstrates that the class is robust and hence interesting in its own right. The comparison with our characterisation for 2-PDA graphs also gives an idea of what 'collapse means' in terms outside of higher-order automata theory. A related construction, which is a kind of automaticity with respect to nested-trees, gives us a decidability result for first-order logic on a natural subclass of 3-CPDA graphs, which in some sense is optimal.

1. Introduction

Higher-order recursion schemes are systems of rewrite rules on typed non-terminal symbols. They provide an excellent basis for modelling higher-order functional programs and so have enticing prospects for software verification. Whilst a conventional *'order-1'* pushdown automaton (1-PDA) has access to a stack of atomic symbols, an order-n pushdown automaton (n-PDA) employs a stack that itself contains other stacks—an order-$(n+1)$ stack is a stack of order-n stacks. Unfortunately such devices share expressivity with only those recursion schemes satisfying a constraint called *safety* [19]. For expressive parity with unrestricted schemes, the structure of higher-order stacks must be enriched with 'links' and an associated *'collapse'* operation, giving the order-n *collapsible pushdown automata* (n-CPDA) [20, 16].

This paper focuses on the *configuration graphs* of these automata rather than the trees that they generate. Roughly speaking, such graphs consist of a set of stack configurations with a directed edge connecting one configuration to another whenever the underlying automaton would be able to transition between them. The ϵ-*closure* of such a graph 'glues together' nodes linked by an arbitrarily large number of ϵ-transitions. The ϵ-closures of

1998 ACM Subject Classification: F.1.1 Models of Computation, F.4.1 Mathematical Logic.

Key words and phrases: Collapsible Pushdown Automata, First-Order Logic, Automatic Structures, Prefix Rewrite Systems.

Some of the work presented here is based on part of the author's DPhil dissertation [6] and was summarised in ICALP 2012 [8].

Work supported by La Fondation Sciences Mathématiques de Paris.

LOGICAL METHODS
IN COMPUTER SCIENCE

DOI:10.2168/LMCS-???

configuration graphs of n-PDA are well understood, coinciding precisely [11] with the nth level of the *Caucal Hierarchy* [13], a robust class of graphs with decidable MSO theories. By contrast there exists a 2-CPDA yielding a transition graph with undecidable MSO theory [16]. Progress on the question of *first-order logic* and CPDA graphs remained open until recently when Kartzow demonstrated first-order logic is decidable on 2-CPDA graphs [17].

Kartzow's proof shows that 2-CPDA graphs are *tree automatic*. A relational structure is called *automatic* [18, 5] with respect to some class of objects (such as finite words, ω-words or trees) if its domain can be represented as a set of such objects recognised by a finite automaton, together with finite automata recognising relations by acting synchronously on tuples of objects. Since such automata on words or trees have good closure properties, automatic structures enjoy decidable first-order theories.

Other than decidability, Kartzow's work is interesting in that it provides an *external* description of the relations between nodes in the CPDA graph using a natural presentation that is *prima facie* quite different to the internal workings of a CPDA. However, whilst all 2-CPDA graphs are tree automatic, not all tree automatic graphs are 2-CPDA. We build on Kartzow's work and give a *precise* characterisation of 2-CPDA graphs.

Our First Contribution: Seeking inspiration from conventional pushdown automata (order-1), we recall that their transition graphs can be precisely characterised by *prefix rewriting* on words [12, 21]. We generalise prefix rewriting to *nested-words* [2, 3], which are strings endowed with pointers between positions arranged in a well-nested manner. In fact for nested-words there are two ways of defining a set of prefixes; one which ignores the pointer structure (a *'rational prefix set'*) and another which makes use of it (a *'summary prefix set'*). Rewrite systems mapping rational prefix sets to rational prefix sets characterise *precisely* the ϵ-closures of 2-PDA graphs, whilst those mapping summary prefix sets to rational sets characterise *precisely* the ϵ-closures of 2-CPDA graphs.

This gives an account of the difference between 2-PDA and 2-CPDA and also shows that the 2-CPDA graphs form a robust class of interest in its own right. (The Caucal hierarchy already told us this for 2-PDA). We also hope that the rewrite systems (and associated notions of automaticity) will turn out to be of more general use.

Our Second Contribution: Representing stacks of 2-CPDA using nested-words allows us to represent stacks of a restriction of 3-CPDA called 3_2-*CPDA* with *nested-trees*. Via Gaifman's Locality Theorem [15] and the Boolean closure of nested-tree automata acting on trees with a bounded number of branches, one can show that first-order logic is decidable on 3_2-CPDA graphs *without* ϵ-*closure*. This generalisation of Kartzow's decidability result is *optimal* in that adding *any one of* order-3 links, ϵ-closure *or* going to order-4, generates graphs with undecidable first-order theory [9, 7].

Related Work. Our work is heavily indebted to that of Kartzow [17]. Indeed there are close connections between nested-words and trees; each type of structure can be encoded as the other. However, whilst every structure generated by our prefix rewrite systems is tree automatic, the converse is not true; tree automaticity is too strong to give the requisite precise characterisation. Nevertheless, in order to convert 2-(C)PDA into prefix rewrite systems, we use a construction that is very similar to that of Kartzow [17]. However, we differ from Kartzow in the way that we show reachability to be definable; our method exploits the fact we are working with nested-words rather than trees.

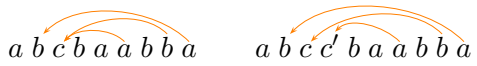


Figure 1: Semi-nested and nested-word

Our proof that a Kartzow-style construction works for nested-words is most neatly formulated using a notion of 'automaticity' that turns out to be equivalent to our prefix-rewrite systems. One challenge is developing a way in which nested-word automata can act synchronously on nested-words, even if their nesting structure differs. Arenas *et al.* provided a comprehensive solution to this problem [4]; we differ by dramatically reducing the power of synchronisation to the point where synchronisation may only occur on the initial segments that the nested-words have in common.

Carayol [10] also defined a generalisation of prefix rewriting for n-PDA (without collapse) which is quite different from our own in that it is described in terms of stack operations. Carayol's advantages include being able to deal with all orders (rather than just level-2) and obtaining some good logical characterisations for these graphs. Our strong points are being able to say something about collapse, as well as formulating our rewrite systems in terms naturally understood without referring to higher-order stacks.

2. Prefix and Suffix Rewriting on Nested-Words

2.1. **Nested-Words.** Recall that a Σ-labelled *word* is a map $w : \mathbf{dom}(w) \longrightarrow \Sigma$ where $\mathbf{dom}(w)$ is a downward closed subset of $\mathbb{N}$ (in particular $0 \in \mathbf{dom}(w)$). A *(semi-)nested-word* is a word endowed with 'back pointers' that are arranged in a well-nested manner.

Definition 2.1. A *semi-nested-word* over Σ is a pair $w^{\curvearrowleft E}$ where w is a Σ-labelled word and $E : \mathbf{dom}(w) \rightharpoonup \mathbf{dom}(w)$ is a partial map such that $\mathbf{dom}(E) \cap \mathbf{img}(E) = \emptyset$ and for all $x \in \mathbf{dom}(E)$, $E(x) < x$ and $E(x) \leq E(y)$ for every $y \in \mathbf{dom}(E)$ such that $E(x) < y < x$. We call it a *nested-word* if the last requirement is strengthened to $E(x) < y < x$ implying $E(x) \lneq E(y)$. We also define $succ(x) := x + 1$.

Graphically we represent E using pointers as in Figure 1. This inspires the terminology *source of a pointer* to refer to elements of $\mathbf{dom}(E)$ and *target of a pointer* for elements of $\mathbf{img}(E)$. The set of nested-words is denoted $\mathbf{NWord}(\Sigma)$.

The definition is very similar to that given by Alur *et al.* [3] with the main difference that we disallow 'unmatched calls'—we cannot have a position in the word being the target of a pointer without the pointer having a corresponding source. This is important when considering a *prefix* of a nested-word $w^{\curvearrowleft E}$ which we define to be a nested-word $w\restriction_S{}^{\curvearrowleft E\restriction_S}$ for some downward closed subset $S \subseteq \mathbf{dom}(w)$. (We often abbreviate this to $u^{\curvearrowleft E\restriction_u}$ where u is a prefix of w.) Where $i \in \mathbf{dom}(w)$ is a *position* in a non-nested word w we write $w_{<i}$ for the prefix of w strictly before i and $w_{\leq i}$ when i is included. The requirement that a prefix of a nested-word be itself a nested-word removes information about pointers sourced in the suffix and targeted at the prefix. It is useful to have a notion of prefix for which such information is retained. For this reason we make a distinction with *visibly pushdown words* [2] which coincide with Alur *et al.*'s notion of nested-words but differ from our own. *Left* of Figure 2 illustrates this.

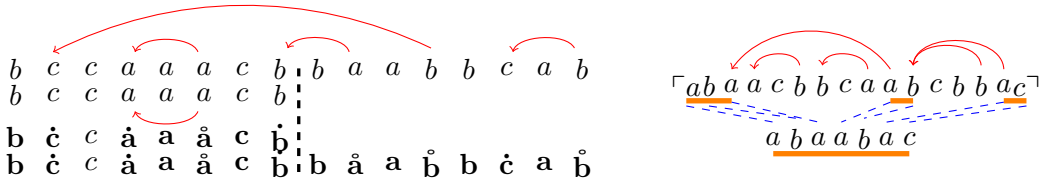


Figure 2: *Left*: Prefix of a nested-word *vs.* visibly pushdown word. *Right*: Summary

Definition 2.2. Given an alphabet Σ, the associated *visibly pushdown alphabet* $\mathfrak{V}(\Sigma)$ is given by $\mathfrak{V}(\Sigma) := \Sigma \cup \mathring{\Sigma} \cup \dot{\Sigma}$, where $\mathring{\Sigma} = \{ \mathring{a} \ : \ a \in \Sigma \}$ and $\dot{\Sigma} = \{ \dot{a} \ : \ a \in \Sigma \}$. Given $w^{\curvearrowleft E} \in \mathbf{NWord}(\Sigma)$ we define the *visibly pushdown word* $\mathfrak{V}(w^{\curvearrowleft E})$ to be the $\mathfrak{V}(\Sigma)$-word w' where $\mathbf{dom}(w') = \mathbf{dom}(w)$ and for each $x \in \mathbf{dom}(w')$ with $w(x) = a$:

$$w'(x) \ := \ \begin{cases} \dot{a} & \text{if } x \in \mathbf{img}(E) \\ a & \text{if } x \notin \mathbf{img}(E) \ \cup \ \mathbf{dom}(E) \\ \mathring{a} & \text{if } x \in \mathbf{dom}(E) \end{cases}$$

Note that $\mathfrak{V}$ does not make sense for semi-nested words as the pointer structure could not be uniquely recovered from the image of the map.

Another concept that we borrow from Alur *et al.* is the idea of the *summary* $\ulcorner w^{\curvearrowleft E} \urcorner$ of a nested-word $w^{\curvearrowleft E}$. This is the string that is obtained by reading the nested-word from left to right 'skipping over' the back edges as they are found. This concept applies equally well to semi-nested words.

Definition 2.3. The *summary* $\ulcorner w \urcorner$ of a (semi-)nested-word w is defined by:

$$\ulcorner w \ a \urcorner := \ulcorner w \urcorner \ a \text{ if } a \text{ sources no pointer} \quad \ulcorner w_0 \ a_0 \ w_1 \ a_1 \urcorner := \ulcorner w_0 \urcorner \ a_0 \ a_1$$

2.2. **(Semi)-Nested-Word Automata.** Formally a nested-word automaton (NWA) is a tuple:

$$\mathcal{A} \ = \ \langle \Sigma, Q, I, \delta_{\oplus}, \delta, \delta_{\ominus}, F \rangle$$

such that Σ is a finite alphabet, Q is a finite set of control-states, $I \subseteq Q$ is a set of initial states, $F \subseteq Q$ is a set of accepting states and $\delta_{\oplus}, \delta, \delta_{\ominus}$ are transition functions:

$$\delta_{\ominus} \ : \ \Sigma \ \times \ Q \ \longrightarrow \ 2^Q \qquad \delta \ : \ \Sigma \ \times \ Q \ \longrightarrow \ 2^Q \qquad \delta_{\oplus} \ : \ \Sigma \ \times \ Q \ \times Q \ \longrightarrow \ 2^Q$$

A *run* of $\mathcal{A}$ on $w^{\curvearrowleft E} \in \mathbf{NWord}(\Sigma)$ is a Q-labelled word $\mathcal{R}$ such that $\mathbf{dom}(\mathcal{R}) = \mathbf{dom}(w) \ \cup \ \{ \ |\mathbf{dom}(w)| \ \}$ and $\mathcal{R}(0) \in I$ and:

- If $i \in \mathbf{img}(E)$, then $\mathcal{R}(i+1) \in \delta_{\ominus}(w(i))$.
- If $i \in \mathbf{dom}(w) - (\mathbf{dom}(E) \cup \mathbf{img}(E))$, then $\mathcal{R}(i+1) \in \delta(w(i), \mathcal{R}(i))$.
- If $i \in \mathbf{dom}(w)$, then $\mathcal{R}(i+1) \in \delta_{\oplus}(w(i), \mathcal{R}(E(i)), \mathcal{R}(i))$.

A run $\mathcal{R}$ on $w^{\curvearrowleft E}$ is *accepting* if $\mathcal{R}(|\mathbf{dom}(w)|) \in F$. The *language recognised* by $\mathcal{A}$ is defined by: $\mathcal{L}(\mathcal{A}) = \{ \ w^{\curvearrowleft E} \ : \ \mathcal{A} \text{ has an accepting run on } w^{\curvearrowleft E} \ \}$. Languages so recognised are called *regular nested-word languages*.

A nested-word automaton is *deterministic* if the image of every transition function consists only of singleton sets and the set I of initial states is also a singleton. Nested-word automata can be *determinised* and are hence closed under Boolean operations [2, 3].

Remark 2.4. For $w^{\frown E} \in \mathbf{NWord}(\Sigma)$ let us write $w^{\frown E^{\leftarrow}} := w^{\leftarrow\frown E^{\leftarrow}}$ to denote $w^{\frown E}$ 'written backwards with pointers inverted'. To be more precise, $\mathbf{dom}(w^{\leftarrow}) := \mathbf{dom}(w)$ with $w^{\leftarrow}(i) := w(|\mathbf{dom}(w)| - i - 1)$ for every $i \in \mathbf{dom}(w)$; $\mathbf{dom}(E^{\leftarrow}) = \{\ |\mathbf{dom}(w)| - i - 1\ :\ i \in \mathbf{img}(E)\ \}$ and $\mathbf{img}(E^{\leftarrow}) = \{\ |\mathbf{dom}(w)| - i - 1\ :\ i \in \mathbf{dom}(E)\ \}$ with $i \in \mathbf{img}(E)$ implying $E^{\leftarrow}(|\mathbf{dom}(w)| - i - 1) := |\mathbf{dom}(w)| - j - 1$ where $E(j) = i$.

Let us further define $\mathcal{L}(\mathcal{A})^{\leftarrow} := \{\ w^{\frown E^{\leftarrow}}\ :\ w^{\frown E} \in \mathcal{L}(\mathcal{A})\ \}$. It is not difficult to see that if $\mathcal{L}$ is regular then $\mathcal{L}^{\leftarrow}$ must also be regular. The transition functions of a nested-word automaton recognising $\mathcal{L}$ can be reversed, and the exchange between targets and sources of pointers handled by guessing at a source-turned-target the state of the run at the target-turned-source, a guess which can be verified when the latter is reached.

We will also find it helpful to consider automata acting on *semi*-nested words. These are defined in the same way as nested-word automata except that they may take semi-nested words as input. (Note that semi-nested words are intrinsically asymmetric as targets of a pointer may be shared but sources may not be. Thus the remark above does not apply to semi-nested words).

2.3. **Prefix and Suffix Rewriting.** In the literature it is conventional to consider prefix rewriting [12], and we wish to emphasise the fact that this work can be viewed as a generalisation of such apparatus. At the same time, we find it technically convenient to work with *suffix* rewrite systems since this orientation fits more naturally with the various constructions we employ. However, as observed in remark 2.4, regularity for nested-words is symmetric, as indeed is the case with regularity for standard non-nested-words. Therefore whilst we will only define suffix rewrite systems here, all of the definitions can be converted into a 'prefix rewriting' variant.

A *suffix-rewrite system* would traditionally consist of a regular language $\mathcal{L}$ of standard (non-nested) words together with pairs of regular languages $(\mathcal{L}_i, \mathcal{L}'_i)$ each assigned a label e_i. The *suffix recognisable graph* generated by such a system has $\mathcal{L}$ as its node-set and an e_i labelled edge from a word of the form uv to a word of the form uv' whenever $v \in \mathcal{L}_i$ and $v' \in \mathcal{L}'_i$. Call the resulting class of structures $\boldsymbol{\mathcal{RW}}$.

All of our generalisations have the form:

$$\mathcal{RW} = \left\langle\ \mathcal{L}, \mathcal{L}_1 \xrightarrow{e_1} \mathcal{L}'_1, \ldots, \mathcal{L}_k \xrightarrow{e_k} \mathcal{L}'_k\ \right\rangle$$

where $\mathcal{L}$ is a regular *nested*-word language over some alphabet Σ and $\mathcal{L}_i$, $\mathcal{L}'_i$ are regular languages of *non-nested words* over that same alphabet. The nodes of the graph generated are always the elements of $\mathcal{L}$.

The 'dangling pointers' targeting the prefix whose sources disappear during the rewriting of a suffix must be 'preserved'. The language $\mathcal{L}$ will constrain where their new sources in the new suffix may be located. In order to express this, we use $\mathfrak{V}$ to mark the prefix in order to track the targets of such dangling pointers. Given a nested-word of the form $(uv)^{\frown E}$, we abuse notation and write $\mathfrak{V}(u^{\frown E})$ to mean the prefix of $\mathfrak{V}((uv)^{\frown E})$ corresponding to u (noting E defines all positions in u that were originally sources.) A

- *rat-rat* system has an e_i-labelled edge from $(uv)^{\frown E}$ to $(uv')^{\frown E'}$ if $v \in \mathcal{L}_i$, $v' \in \mathcal{L}'_i$ and $\mathfrak{V}(u^{\frown E}) = \mathfrak{V}(u^{\frown E'})$. Denote this class of graphs by $\boldsymbol{\mathcal{RW}_{rr}}$.
- *sum-rat* system has an e_i-labelled edge from $(uv)^{\frown E}$ to $(uv')^{\frown E'}$ if $\ulcorner v^{\frown E\restriction_v} \urcorner \in \mathcal{L}_i$, $v' \in \mathcal{L}'_i$ and $\mathfrak{V}(u^{\frown E}) = \mathfrak{V}(u^{\frown E'})$. Denote this class of graphs by $\boldsymbol{\mathcal{RW}_{sr}}$.

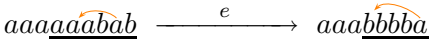
Note that sum-rat rewrite systems subsume their rat-rat counterparts. It is always possible to enrich the words in the domain so that each node $u \in \mathbf{dom}(E)$ with corresponding $u' \in \mathbf{img}(E)$ is decorated with a function mapping each state q of a given regular automaton to the sets of states that it could be in at node u had it started reading the word from u' in state q. This is essentially the construction used in the determinisation of nested-word automata [2, 3] and so we do not spell it out here. The subsumption of rat-rat by sum-rat will, in any case, follow from our results establishing their correspondence with 2-PDA and 2-CPDA respectively.

Example 2.5. The following is an example of a traditional rational suffix rewrite system (the suffix analogue of [12]). Consider $\mathcal{L} = (a+b)^*$ and a rule: $e : a^*b \longrightarrow b^*a$. Then we have the following edges in the graph:

$$abaaa\underline{aaab} \xrightarrow{\quad e \quad} abaaa\underline{bba} \quad \text{and} \quad aaa\underline{aaaaaaab} \xrightarrow{\quad e \quad} aaa\underline{bbbbbba}$$

Example 2.6. We now give an example of a rat-rat system. Consider an $\mathcal{L}$ based on $(a+b)^*$ that allows arbitrary well-nested pointers. The rule: $e : a^*b \longrightarrow b^*a$ interpreted as a rat-rat rule would now give the following edges in the graph:

To give an example of a sum-rat system, let us interpret the rule as a sum-rat rule. This now gives the following edge (which would not belong to the rat-rat interpretation).



3. (C)PDA Graphs

3.1. **Higher-Order Stacks.** Let us fix a stack-alphabet Γ. For higher-order automata this alphabet must be finite, but it is convenient for definitions to allow it to be infinite. An *order*-1 stack over Γ is just a string of the form $[\ \gamma\ ]$ where $\gamma \in \Gamma^*$. Let us refer to the set of order-1 stacks over Γ as $stack_1(\Gamma)$. For $n \in \mathbb{N}$ the set of *order*-$(n+1)$ stacks is: $stack_{n+1}(\Gamma) := stack_1(stack_n(\Gamma))$. We allow the following operations on an order-1 stack s for every $a \in \Gamma$: $push_1^a([\ a_1 \cdots a_m\ ]) := [\ a_1 \cdots a_m a\ ]$, $pop_1([\ a_1 \cdots a_m a_{m+1}\ ]) := [\ a_1 \cdots a_m\ ]$, $nop(s) := s$. We allow the following *order*-$(n+1)$ *operations* on an order-$(n+1)$ stack s, where θ is any *order-n operation*: $push_{n+1}([\ s_1 \cdots s_m\ ]) := [\ s_1 \cdots s_m s_m\ ]$, $pop_{n+1}([\ s_1 \cdots s_m s_{m+1}\ ]) := [\ s_1 \cdots s_m\ ]$, $\theta([\ s_1 \cdots s_{m-1} s_m\ ]) := [\ s_1 \cdots s_{m-1} \theta(s_m)\ ]$. Where $s = [\ s_1\ \cdots\ s_m\ ]$ is an $(n+1)$-stack for $n \geq 1$, we define $top_{n+1}(s) := s_m$ and $top_k(s) := top_k(s_m)$ for $1 \leq k \leq n$. We abuse notation in defining $top_{n+1}(t) := t$ when t is an n-stack. Define $|s| := m$ for $s = [\ s_1 \cdots s_m\ ]$.

3.2. **Collapsible Pushdown Stacks.** We first informally describe how CPDA stacks extend those introduced above. CPDA include a new $push_1^{a,k}$ operation, which attaches a *k-link* from the newly created a element that points to the k-stack below. The targets of these links are preserved during higher-order *push* operations in the manner illustrated by:

Example 3.1.

$$[[[abca]\ [ab]]] \xrightarrow{push_1^{a,2};push_2;push_3;push_1^{c,3};push_3} [[[abca]\ [aba]\ [aba]]\ [[abca]\ [aba]\ [abac]]\ [[abca]\ [aba]\ [abac]]]$$

We include the orders of links a stack may use as a subscript, so an order-n_S stack is an order-n stack equipped with order-i links for each $i \in S$. Formally the *S-collapsible pushdown alphabet* $\Gamma^{[S]}$ (for $S \subseteq \mathbb{N}$) induced by an alphabet Γ is the set $\Gamma \times S \times \mathbb{N}$. The set of order-$n_S$ collapsible stacks $stack_{n_S}^{\mathcal{C}}(\Gamma)$ is defined by: $stack_{n_S}^{\mathcal{C}}(\Gamma) := stack_n(\Gamma^{[S]})$. An atomic element $(a, l, p) \in \Gamma^{[S]}$ has label a with a link of order l. If $l < n$ the target of a link is the pth $(l-1)$-stack of the l-stack in which the element resides.

We thus introduce the operations $push_1^{a,l}(s) := push_1^{(a,l,|top_{l+1}(s)|-1)}(s)$ and $collapse(s) := pop_l^{|top_{l+1}(s)|-p}$ where $top_1(s) = (a, l, p)$ and θ^m represents the operation θ iterated m times. From now on we will abuse notation and consider $top_1(s) := a$. We define $\boldsymbol{l_o}(a) := l$ and $\boldsymbol{l_a}(a) := p$. Write Θ_{n_S} to denote the set of order-n_S collapsible stack operations. Note that a *collapse* on a 1-link can always be simulated by a pop_1 operation; 1-links are therefore 'equivalent' to there being no link. We thus always implicitly assume that an n_S-stack allows 1-links (even if $1 \neq S$) and write $push_1^a$ (pushing a 'without a link') to mean $push_1^{a,1}$.

3.3. **The Automata, their Graphs and First-Order Logic.** Let $n \in \mathbb{N}$ and let $S \subseteq [2..n]$. An n_S-CPDA (order-n_S collapsible pushdown automaton) $\mathcal{A}$ is a tuple:

$$\langle\ \Sigma, \Pi, Q, q_0, \Gamma, R_{a_1}, R_{a_2}, \ldots, R_{a_r}, P_{b_1}, P_{b_2}, \ldots, P_{b_{r'}}\ \rangle$$

where Σ is a finite set of transition labels $\{a_1, a_2, \ldots, a_r\}$; Π is a finite set of configuration labels $\{b_1, b_2, \ldots, b_{r'}\}$; Q is a finite set of control-states; $q_0 \in Q$ is an initial control-state; Γ is a finite stack alphabet; each R_{a_i} is the a_i-labelled transition relation with $R_{a_i} \subseteq Q \times \Gamma \times \Theta_{n_S} \times Q$; each P_{b_i} is the b_i-labelled unary predicate specified by $P_{b_i} \subseteq Q \times \Gamma$. Remaining consistent with the definitions in the literature, an n-CPDA is an $n_{[n..2]}$-CPDA and an n-PDA is an $n_\emptyset$-CPDA. We write 3_2 and 3_3-CPDA to mean $3_{\{2\}}$ and $3_{\{3\}}$-CPDA.

A *configuration* of an n_S-CPDA $\mathcal{A}$ is a pair (q, s) where q is a control-state and s is an n_S-stack. Such a configuration *satisfies* the predicate $b_i \in \Pi$ just in case $(q, top_1(s)) \in P_{b_i}$. We say $\mathcal{A}$ can *a_i-transition from (q, s) to $(q', \theta(s))$*, written $(q, s) \xrightarrow{a_i} (q', \theta(a))$, iff $(q, top_1(s), \theta, q') \in R_{a_i}$. Let us further say that *(q', s') can be reached from (q, s)* in $\mathcal{A}$ *with path labeled in* $\mathcal{L}$ for some $\mathcal{L} \subseteq \Sigma^*$ iff $(q, s) \xrightarrow{a_{i_1}} (q_1, s_1) \xrightarrow{a_{i_2}} \cdots (q_{m-1}, s_{m-1}) \xrightarrow{a_{i_m}} (q', s')$ for some configurations $(q_1, s_1), \ldots, (q_{m-1}, s_{m-1})$ where $a_{i_1} a_{i_2} \cdots a_{i_m} \in \mathcal{L}$. We write $(q, s) \xrightarrow{\mathcal{L}} (q', s')$ to mean this. The set of *reachable configurations* of $\mathcal{A}$ is given by:

$$\boldsymbol{R}(\mathcal{A}) := \{\ (q, s)\ :\ (q_0, \bot_n) \xrightarrow{\Sigma^*} (q, s)\ \} \quad \text{with } \bot_n \text{ the empty } n\text{-stack}$$

Definition 3.2. The configuration graph of (graph generated by) $\mathcal{A}$ has domain $\boldsymbol{R}(\mathcal{A})$, unary predicates Π and directed edges Σ between configurations. We write $\mathcal{G}(\mathcal{A})$ to denote this graph. For a distinguished $\epsilon \in \Sigma$, the ϵ-closure $\mathcal{G}^\epsilon(\mathcal{A})$ is induced from $\mathcal{G}(\mathcal{A})$ by restricting

the domain to nodes reachable from the initial configuration by a path of the form $\xrightarrow{\epsilon^* b}$ for $b \neq \epsilon$, and defining a-labelled edges between vertices related by $\xrightarrow{\epsilon^* a}$ in $\mathcal{G}(\mathcal{A})$.

We also write $\mathbf{Pd}_{\boldsymbol{\epsilon} n}$ to denote the class of ϵ-closures of n-PDA and $\mathbf{Pd}_{\boldsymbol{\epsilon} n}^{\mathcal{C}}$ to denote the class of ϵ-closures of n-CPDA.

An n-(C)PDA is described as *slow* if no ϵ-transition ever performs a $push_n$, pop_n or a *collapse* on an n-link. In particular, if an edge in the ϵ-closure of the graph of a slow n-CPDA affects more than the top $(n-1)$-stack, then it is produced by just a single operation.

First-Order Logic $\boldsymbol{FO}$ on such a graph has the unary and binary relation labels as predicates, together with existential and universal quantification over nodes. The logic $\boldsymbol{FO^{\infty}}$ adds a quantifier $\exists^{\infty}$ asserting that there are infinitely many witnesses for the variable it binds. A *sentence* is a formula with no free variables and the *first-order theory* of a structure is the set of first-order sentences that are true of it. We also have the following fragment of *transitive-closure logic*:

Definition 3.3. The logic $\boldsymbol{FO(TC[\Delta_0])}$ is first-order logic extended with binary atomic predicates of the form $\overline{\phi(x,y)}$ where $\phi(x,y)$ is a *quantifier free* formula whose free variables are all belong to $\{x,y\}$. For nodes c and c' of the graph we have $c\,\overline{\phi(x,y)}\,c'$ holding iff there exist nodes $c := c_1, c_2, \ldots, c_{k-1}, c_k =: c'$ such that $\phi(c_i, c_{i+1})$ holds for every $1 \leq i < k$.

3.4. **Monotonic Automata and Derivatives.** In [6, 7] we show that given an n_S-CPDA $\mathcal{A}$ we can construct an n_S-CPDA $\mathcal{A}^{\uparrow}$ such that $\mathcal{G}^{\epsilon}(A^{\uparrow}) \cong \mathcal{G}^{\epsilon}(\mathcal{A})$ but such that if:

$$c := (q, [s_1\ s_2\ \cdots\ s_{m-1}\ s_m]) \xrightarrow{\epsilon^* a} (q, [s_1\ s_2\ \cdots\ s_{m-1}\ s'_m\ s'_{m+1}\ \cdots\ s'_{m'}]) =: c'$$

where every configuration occurring in the run has the form $(p, [s_1\ s_2\ \cdots\ s_{m-1}\ t_1\ \cdots\ t_l])$ for $l \geq 1$, then $\mathcal{A}^{\uparrow}$ has a run from c to c' without performing pop_n or *collapse* on an n-link. In particular every configuration in $\boldsymbol{R}(\mathcal{A}^{\uparrow})$ can be reached by $\mathcal{A}^{\uparrow}$ in this way.

The derivative [6, 7] of an n-CPDA $\mathcal{A}$ is defined to be the $(n-1)$-CPDA:

$$\partial(\mathcal{A}) = \left\langle\ \Sigma\ \cup\ \{a_i^p\ :\ a_i \in \Sigma\}, \Pi, Q, q_0, \Gamma, R'_{a_1}, R'_{a_1^p}, R'_{a_2}, R'_{a_2^p}, \ldots, R'_{a_r}, R'_{a_r^p}, P_{b_1}, P_{b_2}, \ldots, P_{b_{r'}}\ \right\rangle$$

where $\mathcal{A}^{\uparrow} = \left\langle\ \Sigma, \Pi, Q, q_0, \Gamma, R_{a_1}, R_{a_2}, \ldots, R_{a_r}, P_{b_1}, P_{b_2}, \ldots, P_{b_{r'}}\ \right\rangle$ and for $1 \leq i \leq r$:

$$R'_{a_i} := \{\ (q,b,\theta,q') \in R_{a_i}\ :\ \theta \notin \{push_n, push_1^{c,1}, pop_n\}\ \}\ \cup\ \{\ (q,b,push_1^c,q')\ :\ (q,b,push_1^{c,2},q') \in R_{a_i}\}$$

$$R'_{a_i^p} := \{\ (q,b,nop,q')\ :\ (q,b,push_n,q') \in R_{a_i}\ \}$$

The derivative simulates the topmost $(n-1)$-stack of the original CPDA.

Remark 3.4. Given an n-CPDA $\mathcal{A}$, the reachable configurations of $\partial(\mathcal{A}^{\uparrow})$ are precisely those of the form $(q, top_n(s))$ where $(q,s) \in \boldsymbol{R}(\mathcal{A}^{\uparrow})$. This is because every configuration of $\mathcal{A}^{\uparrow}$ is reachable without performing pop_n or *collapse* on an n-link.

3.5. **Link Stripping and Stack Colouring for 2-CPDA.** We will want to construct a representation of a graph for a 2-CPDA $\mathcal{A}$ from a representation of the graph of $\partial(\mathcal{A})$. One issue is that in general $\partial(\mathcal{A})$ loses information about 2-links. Writing $\mathbf{stripln}(s)$ for the result of deleting 2-links from a 2-stack s, we will therefore 'colour' stack elements so that the original links of a stack s of $\mathcal{A}$ are uniquely recoverable from $\mathbf{stripln}(s)$. 'Colouring' guarantees that *collapse* discards precisely those 1-stacks of which the top_2 stack is a prefix.

We use two colours, **red** and **blue**, which annotate atomic elements, alternating every time we have a link to a different stack. We also have an annotation 1 that indicates an element with no link. At the top of each 1-stack we indicate the colour that any new links should use. We write $\mathbf{Trail}(\mathcal{A})$ to denote this modified automaton.

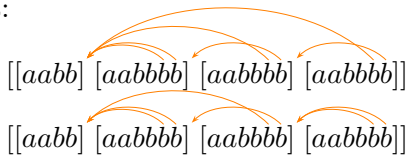
Definition 3.5. Let $\mathcal{A}$ be a 2-CPDA with stack-alphabet Γ. The 2-CPDA $\mathbf{Trail}(\mathcal{A})$ has stack-alphabet $\Gamma \times \{1, \mathbf{red}, \mathbf{blue}\} \times \mathbb{B}^{\perp} \cup \{ \mathbf{red}, \mathbf{blue} \} \times \mathbb{B}$. $\mathbf{Trail}(\mathcal{A})$ begins in '**red**-mode'. A 2-link in a 1-stack is called *fresh* if it points to the 1-stack immediately below.

Where $\mathcal{A}$ would have performed $push_1^a$, $\mathbf{Trail}(\mathcal{A})$ performs $push_1^{(a,1,\perp)}$. Where the original would have performed $push_1^{a,2}$, it performs $push_1^{(a,X,b),2}$ when in X-mode, where b is $\mathbf{t}$ if this will be the first (lowest) fresh link in its top_2 1-stack and $\mathbf{f}$ otherwise. Prior to performing $push_2$ it first pushes (X, b) onto the stack (which is immediately popped off the copy), where X is its mode and $b = \mathbf{t}$ iff the 1-stack being copied contains a fresh link. Note that with these annotations, together with the marking of the lowest fresh link in a 1-stack, the presence of fresh links is easy for the automaton to track.

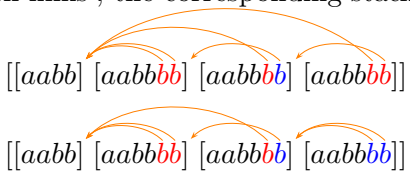
The mode is changed from X-mode (either **red**-mode or **blue**-mode) to the other colour Y just in case one of the following occurs:

- The automaton performs a pop_1 operation discarding a 2-link that is *not* fresh in the current 1-stack and that is the same colour as X.
- The automaton performs a pop_2 or a *collapse* operation in which case it adopts the mode specified by the element on top of the stack.

Example 3.6. Suppose that:



are stacks of $\mathcal{A}$. Then ignoring the additional Boolean values on the stack required for 'tracking the existence of fresh links', the corresponding stacks of $\mathbf{Trail}(\mathcal{A})$ are:



The significance of colouring is given by the following Lemma:

Lemma 3.7. *(1)* $\mathcal{G}^{\epsilon}(\mathcal{A}) \cong \mathcal{G}^{\epsilon}(\mathbf{Trail}(\mathcal{A}))$

(2) If s and s' are two stacks belonging to reachable configurations of $\mathbf{Trail}(\mathcal{A})$ *such that* $\mathbf{stripln}(s) = \mathbf{stripln}(s')$*, then* $s = s'$*.*

(3) If $\mathbf{stripln}(s) = [s_1\ s_2\ \cdots\ s_r\ s_{r+1}\ \cdots\ s_m]$ *is a stack in a reachable configuration of* $\mathbf{Trail}(\mathcal{A})$*, then if* $top_1(s)$ *has a 2-link,* $\mathbf{stripln}(collapse(s)) = [s_1\ s_2\ \cdots\ s_r]$ *where* $s_m \sqsubseteq_1 s_i$ *for every* $r+1 \leq i \leq m$ *but* $s_m \not\sqsubseteq_1 s_r$*.*

Proof. (1) The isomorphism comes from the fact that $\mathbf{Trail}(\mathcal{A})$ mimics the same operations as $\mathcal{A}$ and colours for new elements, mode and freshness tracking are uniquely determined from the stack content in any given configuration.

(2) Suppose for contradiction that the statement is false. We can then pick the smallest h such that there exist s and s' with $\mathbf{stripln}(s) = \mathbf{stripln}(s')$ and $|s| = |s'| = h$ but $s \neq s'$. It must thus be the case that there are instances of an atomic element γ in the same position in s and s' such that $\boldsymbol{l_o}(a) = \boldsymbol{l_o}(a') = 2$ but $\boldsymbol{l_a}(a) \neq \boldsymbol{l_a}(a')$. Indeed these must be the respective top elements of s and s' or else we would contradict minimality of h. But this in turn means that one of them must have been created afresh in the top 1-stack and the other must have been copied from the 1-stack below. W.l.o.g. suppose that a was copied from the 1-stack below and that a' was created afresh. In order to create a in s it would have been necessary to pop_1 the copy of a' resulting from $push_2$ on the stack below; no other links would be pop_1'ed or else we would violate the minimality of h. But by assumption a' would not have been fresh in the top_2-stack and so the mode would have changed to be the opposite colour of a' at the point when a is created, contradicting $\mathbf{stripln}(s) = \mathbf{stripln}(s')$.

(3) Ignoring the $\mathbf{stripln}(\_)$ the statement holds since the target of a 2-link in a 2-CPDA is completely determined by the 1-stack in which it was first created and moreover the atomic elements below a link cannot have been changed (since then the link would have been discarded). The statement must thus hold by item 2.

□

3.6. **Characterisation of the 2-(C)PDA Graphs.** A large part of this paper is devoted to showing that the following relationship holds:

Theorem 3.8. $\boldsymbol{\mathcal{RW}_{sr} = \mathbf{Pd}^{\mathcal{C}}_{\epsilon 2}}$ *and* $\boldsymbol{\mathcal{RW}_{rr} = \mathbf{Pd}_{\epsilon 2}}$.

We thus see that the power of the 'collapse' operation corresponds precisely to the ability of a rewrite system to look at summaries. Note that this characterisation also makes clear the inherent *asymmetry* of the collapse operation (see also remark 7.3); whilst 2-PDA are characterised using 'symmetric rules' ('rat' on both sides) this is not so for 2-CPDA.

4. From Suffix Rewriting to CPDA

This section will be devoted to converting rewrite systems to 2-(C)PDA, *i.e.* proving:

Lemma 4.1. $\boldsymbol{\mathcal{RW}_{sr} \subseteq \mathbf{Pd}^{\mathcal{C}}_{\epsilon 2}}$ *and* $\boldsymbol{\mathcal{RW}_{rr} \subseteq \mathbf{Pd}_{\epsilon 2}}$.

4.1. **Proof Idea.** First we will give the basic intuition of our construction. We define maps encoding nested-words as order-2 stacks:

$$\mathfrak{S} : \mathbf{NWord}(\Sigma) \longrightarrow stack_2(\mathfrak{V}(\Sigma)) \quad \text{and} \quad \mathfrak{S}^{\mathcal{C}} : \mathbf{NWord}(\Sigma) \longrightarrow stack^{\mathcal{C}}_{2_2}(\mathfrak{V}(\Sigma))$$

Consider $w^{\curvearrowleft E} \in \mathbf{NWord}(\Sigma)$. We define $\mathfrak{S}^{\mathcal{C}}(w^{\curvearrowleft E}) := \mathfrak{S}^{\mathcal{C}-}(\mathfrak{V}(w^{\curvearrowleft E}))$ where:

$$\mathfrak{S}^{\mathcal{C}-}(\epsilon) := \bot_2 \quad \mathfrak{S}^{\mathcal{C}-}(v\ \dot{a}) := (push_2; push_1^{\dot{a},2})(\mathfrak{S}^{\mathcal{C}}(v)) \quad \mathfrak{S}^{\mathcal{C}-}(v\ a) := push_1^{a}(\mathfrak{S}^{\mathcal{C}}(v))$$

$$\mathfrak{S}^{\mathcal{C}-}(v\ \dot{a}\ \cdots\ c\ \mathring{b}) := push_1^{\mathring{b}}(\mathfrak{S}^{\mathcal{C}}(v\ \dot{a}\ \cdots\ c)\ ::\ top_2(\mathfrak{S}^{\mathcal{C}}(v\ \dot{a})))$$

where we define $[\ s_1 \cdots\ s_m\ ] :: s_{m+1} := [\ s_1 \cdots\ s_m\ s_{m+1}\ ]$. We define $\mathfrak{S}(w^{\curvearrowleft E})$ in the same way as $\mathfrak{S}^{\mathcal{C}}(w^{\curvearrowleft E})$, ignoring stack pointers.

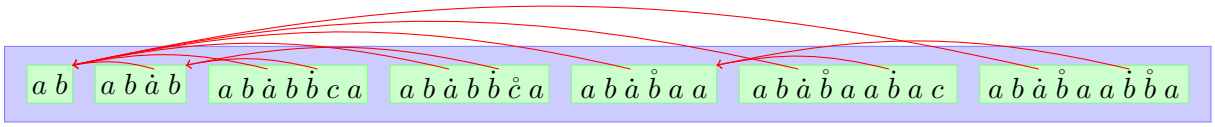


Figure 3: The 2-CPDA stack corresponding to a nested-word

Example 4.2. Consider the nested-word: $w^{\curvearrowleft E} := a\ b\ a\ b\ b\ c\ a\ c\ a\ b\ a\ a\ b\ a\ c\ b\ a$ We illustrate $\mathfrak{S}^{\mathcal{C}}(w^{\curvearrowleft E})$ in Figure 3.

We construct a 2-PDA (resp. 2-CPDA) that performs a series of transitions with labels of the form $\epsilon^* e$ from stack $\mathfrak{S}(w^{\curvearrowleft E})$ to $\mathfrak{S}(w'^{\curvearrowleft E'})$ (*resp.* $\mathfrak{S}^{\mathcal{C}}(w^{\curvearrowleft E})$ to $\mathfrak{S}^{\mathcal{C}}(w'^{\curvearrowleft E'})$) iff $w^{\curvearrowleft E}$ suffix rewrites to $w'^{\curvearrowleft E'}$ via an e-labelled rule.

In order to skip over a pointer sourced at $\mathring{b}$ one may simply pop_1 and then *collapse* from its target $\dot{a}$. This is useful when 'reading a summary backwards' for the left-hand side of a sum-rat rule. If one wishes to ignore pointers (as needed for the left-hand-side of a rat-rat rule) one may use pop_2 at $\dot{a}$ instead of $pop_1; collapse$. For this a 2-PDA suffices. The next lemma formalises how a CPDA can 'compute' a representation of a nested-word on the fly.

Lemma 4.3. *For any $w^{\curvearrowleft E} \in \mathbf{NWord}(\Sigma)$ and prefix $u^{\curvearrowleft F}$ of $\mathfrak{V}(w^{\curvearrowleft E})$ with $F = E\restriction_{\mathbf{dom}(u)}$:*

$$\ulcorner u^{\curvearrowleft F} \urcorner = top_2(\mathfrak{S}^{\mathcal{C}-}(u^{\curvearrowleft F}))$$

$$\mathfrak{S}^{\mathcal{C}-}(u^{\curvearrowleft F}) = \begin{cases} pop_2(\mathfrak{S}^{\mathcal{C}-}((u\ a)^{\curvearrowleft F})) & \text{if } a \text{ of form } \dot{b} \text{ or } \mathring{b} \\ pop_1(\mathfrak{S}^{\mathcal{C}-}((u\ a)^{\curvearrowleft F})) & \text{otherwise} \end{cases}$$

$$\mathfrak{S}^{\mathcal{C}-}(u^{\curvearrowleft F}) = (pop_1; collapse)(\mathfrak{S}^{\mathcal{C}-}(u^{\curvearrowleft F} \dot{a} \cdots \mathring{b}))$$

Proof. The second item of the Lemma is immediate from the definition of $\mathfrak{S}^{\mathcal{C}-}$. The third item comes from the fact that in $top_2(\mathfrak{S}^{\mathcal{C}-}(v\ \dot{a}))$ we have $\boldsymbol{l_a(\dot{a})} = |\mathfrak{S}^{\mathcal{C}-}(v)|$ and that we always have $\mathfrak{S}^{\mathcal{C}-}(x) \sqsubseteq \mathfrak{S}^{\mathcal{C}-}(x')$ when $x \sqsubseteq x'$. The first item is an easy induction on $|u|$. □

We will extend the stack alphabet so that an atom a in the stack, corresponding to a letter a in a nested-word $w^{\curvearrowleft E}$, is annotated with state information concerning a run on $w^{\curvearrowleft E}$ by the deterministic nested-word automaton $\mathcal{A}$ recognising $\mathcal{L}$. The definition of $\mathfrak{S}(w^{\curvearrowleft E})$ allows the (C)PDA to 'lookup' the state of $\mathcal{A}$ at the target $\dot{a}$ of a pointer when reading its source $\mathring{b}$, since $\mathring{b}$ will always be recorded immediately above $\dot{a}$ in some 1-stack.

4.2. **State Decorations.** Let $\mathcal{A}$ be a *deterministic* nested-word automaton (with states $Q_{\mathcal{A}}$ and $Q_{\mathcal{A}}^{\perp} := Q_{\mathcal{A}} \cup \{\perp\}$). Define a map:

$$\mathfrak{D}_{\mathcal{A}} \ : \ \mathbf{NWord}(\Sigma) \longrightarrow \mathbf{NWord}(\Sigma \times Q_{\mathcal{A}} \times Q_{\mathcal{A}}^{\perp} \times \mathbb{B})$$

such that $\mathfrak{D}_{\mathcal{A}}(w^{\curvearrowleft E}) = w'^{\curvearrowleft E}$ where $\pi_1(w') = w$; $\pi_2(w')$ gives the unique run of $\mathcal{A}$ on w; $\pi_3(w'(i)) = \perp$ at each $i \in \mathbf{dom}(w')$ unless $w' \in \mathbf{dom}(E)$, in which case $\pi_3(w'(i))$ is the state of the run at $E(i)$; $\pi_4(w'(i)) = \mathbf{t}$ iff for all $j < i$, $j \in \mathbf{img}(E)$ implies there exists $j' < i$ such that $E(j') = j$. That is $\pi_4(w')$ is $\mathbf{t}$ only at those points in $w'^{\curvearrowleft E}$ at which there is no unmatched target strictly preceding it. We write $\boldsymbol{symb}$, $\boldsymbol{st}_{\mathcal{A}}$, $\boldsymbol{st}^{\boldsymbol{pr}}_{\boldsymbol{\mathcal{A}}}$ and $\boldsymbol{cld}$ for the

projections π_1, π_2, π_3 and π_4 respectively with the subscripts used to disambiguate when we decorate with multiple deterministic nested-word automata at once.

It is easy to see that if $\mathcal{L}$ is nested-word regular, then its image $\mathfrak{D}_{\mathcal{A}}(\mathcal{L})$ must also be nested-word regular. Observe that $\mathfrak{D}_{\mathcal{A}}(\mathcal{L})$ and $\mathcal{L}$ enjoy a natural one-one correspondence.

For technical reasons we also find it helpful to define $\mathfrak{D}^{\star}_{\mathcal{A}}$ to be a map on *languages* that maps a Σ-labelled nested-word language $\mathcal{L}$ to a nested-word language $\mathcal{L}'$ over the alphabet $\Sigma \times Q_{\mathcal{A}} \times Q_{\mathcal{A}}^{\perp} \times \mathbb{B}$ such that $\pi_1(\mathcal{L}') = \mathcal{L}$ and for every $w^{\frown E} \in \mathcal{L}$ of length k and sequence $r_1, \ldots, r_k$ in $(Q_{\mathcal{A}} \times Q_{\mathcal{Q}}^{\perp} \times \mathbb{B})^*$, there exists $w'^{\frown E} \in \mathcal{L}'$ with $\pi_1(w'^{\frown E}) = w^{\frown E}$ and $\pi_{2,3,4}(w'^{\frown E}) = r_1, \ldots, r_k$. In particular $\mathfrak{D}^{\star}_{\mathcal{A}}(\Sigma)$ produces the alphabet for the image of Σ-labelled languages under $\mathfrak{D}_{\mathcal{A}}$. We can think of $\mathfrak{D}^{\star}_{\mathcal{A}}$ as ignoring the actual transition function of $\mathcal{A}$ but just providing a way to decorate words with all possible arbitrary combinations of state information.

4.3. **The Transformation.**

Lemma 4.4. *Every rat-rat suffix-rewrite graph is the ϵ-closure of a 2-PDA graph.*

Proof. Consider a rat-rat suffix-rewrite system:

$$\mathcal{RW} := \left\langle \mathcal{L}, \mathcal{L}_1 \xrightarrow{e_1} \mathcal{L}'_1, \ \ldots, \ \mathcal{L}_k \xrightarrow{e_k} \mathcal{L}'_k \right\rangle$$

Let $\mathcal{A}$ be a *deterministic* nested-word automaton recognising $\mathcal{L}$. Consider the following suffix rewrite system:

$$\mathcal{RW}' := \left\langle \mathfrak{D}_{\mathcal{A}}(\mathcal{L}), \mathfrak{D}^{\star}_{\mathcal{A}}(\mathcal{L}_1) \xrightarrow{e_1} \mathfrak{D}^{\star}_{\mathcal{A}}(\mathcal{L}'_1), \ \ldots, \ \mathfrak{D}^{\star}_{\mathcal{A}}(\mathcal{L}_k) \xrightarrow{e_k} \mathfrak{D}^{\star}_{\mathcal{A}}(\mathcal{L}'_k) \right\rangle$$

Since $\mathcal{A}$ is deterministic, $\mathcal{L}$ and $\mathfrak{D}_{\mathcal{A}}(\mathcal{L})$ are in bijective correspondence and so $\mathcal{G}(\mathcal{RW}) = \mathcal{G}(\mathcal{RW}')$. It thus suffices to construct a 2-PDA $\mathcal{B}$ such that $\mathcal{G}(\mathcal{RW})' = \mathcal{G}^{\epsilon}(\mathcal{B})$. We describe the permissible behaviours of the automaton when generating an e_i edge; an unbounded number of steps are involved, which should make up a series of transitions of the form $\epsilon^* e_i$.

Let $\mathcal{A}_i$ and $\mathcal{A}'_i$ be a *deterministic* standard non-nested-word automaton recognising the regular languages ${\mathcal{L}_i}^{\leftarrow}$ and ${\mathcal{L}'}_i$ respectively, with respective state spaces $Q_{\mathcal{A}_i}$ and $Q_{\mathcal{A}'_i}$ and transition functions $\delta_{\mathcal{A}_i}$ and $\delta_{\mathcal{A}'_i}$. An e_i-edge is generated by $\mathcal{B}$ as follows. It maintains a register $\boldsymbol{cld}'$ keeping track of whether the currently represented word has unmatched targets (rather than just the word strictly prior to the final symbol), a register $\mathbf{q}$ with value in $Q_{\mathcal{A}_i}$ and a register $\mathbf{q}'$ with value in $Q_{\mathcal{A}'_i}$. In our description we write top_1 to refer to the top element of the stack prior to performing the transition.

- Set $\mathbf{q} := q_0$ where q_0 is the initial state of $\mathcal{A}_i$ and $\boldsymbol{cld}' := \mathbf{t}$.
- *Phase 1:* If the value of $\mathbf{q}$ is an accepting state of $\mathcal{A}_i$, then a non-deterministic choice is made whether to jump to *phase 2a* or to continue with *phase 1*. If it is not accepting, then *phase 1* must continue.
- – If $\boldsymbol{symb}(top_1) = a \in \Sigma$, then a pop_1 is performed and we set $\mathbf{q} := \delta_{\mathcal{A}_i}(a, \mathbf{q})$.
  – If $\boldsymbol{symb}(top_1) = \mathring{a} \in \mathring{\Sigma}$, then we perform pop_2 and set $\mathbf{q} := \delta_{\mathcal{A}_i}(a, \mathbf{q})$
  – If $\boldsymbol{symb}(top_1) = \dot{a} \in \dot{\Sigma}$, then we perform pop_2 and set $\mathbf{q} := \delta_{\mathcal{A}_i}(a, \mathbf{q})$
  In all cases we set $\boldsymbol{cld}' := \boldsymbol{cld}(top_1)$.
- Return to the start of *phase 1*.
- *Phase 2a:* Set $\mathbf{q}' := q'_0$ where q'_0 is the initial state of $\mathcal{A}'_i$.

- *Phase 2b:* If the value of $\mathbf{q}'$ is an accepting state of $\mathcal{A}'_i$ *and* $\boldsymbol{st}_{\mathcal{A}}(top_1)$ is an accepting state of $\mathcal{A}$, top_1 is *not* of the form $\dot{a}$ *and* $\boldsymbol{cld}' = \mathbf{t}$, then we non-deterministically decide whether to perform an e_i-transition and terminate, or continue.
- Non-deterministically pick $a \in \Sigma$. Do one of the following:
  - Perform $push_1^{(a,q,\perp,c)}$ where $q := \delta_{\mathcal{A}}(a, \boldsymbol{st}_{\mathcal{A}}(top_1))$ and
$$c = \begin{cases} \boldsymbol{cld}(top_1) & \text{if } \boldsymbol{symb}(top_1) \notin \dot{\Sigma} \\ \mathbf{f} & \text{if } \boldsymbol{symb}(top_1) \in \dot{\Sigma} \end{cases}$$
    Set $\mathbf{q}' := \delta_{\mathcal{A}'_i}(a, \mathbf{q}')$ and $\boldsymbol{cld}' := \boldsymbol{cld}'$ (no change).
  - Perform $(push_2; push_1^{(\dot{a},q,\perp,c)})$ where $q := \delta_{\mathcal{A}}(a, \boldsymbol{st}_{\mathcal{A}}(top_1))$ and c is as above.
    Set $\mathbf{q}' := \delta_{\mathcal{A}'_i}(a, \mathbf{q}')$ and $\boldsymbol{cld}' := \mathbf{f}$.
  - If $\boldsymbol{cld}(top_1) = \mathbf{f}$ with $p = \boldsymbol{st}_{\mathcal{A}}(top_1)$ then we may do the following. Perform $push_2$ followed by iterated pop_1 precisely until one has as the top element u such that $\boldsymbol{symb}(u) \in \dot{\Sigma}$ that is *not* followed by an element with $\boldsymbol{symb}$ in $\mathring{\Sigma}$. We should then perform $push_1^{(\mathring{a},q,q',c)}$ where $c = \boldsymbol{cld}(u)$, $q' = \boldsymbol{st}_{\mathcal{A}}(u)$ and $q = \delta_{\mathcal{A}}(a, q', p)$.
    Set $\mathbf{q}' := \delta_{\mathcal{A}'_i}(a, \mathbf{q}')$ and $\boldsymbol{cld}' := c$.

For the following induction argument set $u^{\curvearrowleft E} = \mathfrak{V}(w^{\curvearrowleft E})$.

Suppose that the initial stack configuration represents $\mathfrak{S}(w^{\curvearrowleft E}) = \mathfrak{S}^-(u^{\curvearrowleft E})$, where $\mathfrak{S}^-$ is $\mathfrak{S}^{\mathcal{C}-}$ ignoring links, for some $w^{\curvearrowleft E} \in \mathfrak{D}_{\mathcal{A}}(\mathbf{NWord}(\Sigma))$. For phase one we can check by induction on k that after k iterations (of phase one) we have a stack equal to $\mathfrak{S}^-(u'^{\curvearrowleft E\restriction_{u'}})$ where $u = u'\ a_1 \cdots a_k$ and $\mathbf{q}$ is set to the state in which $\mathcal{A}_i$ would be after reading $\boldsymbol{symb}(a_k) \cdots \boldsymbol{symb}(a_1)$. The induction step for the former comes via the second equality of Lemma 4.3 and for the latter is immediate from the way in which $\mathbf{q}$ is updated.

Given the condition for terminating phase one (and moving to phase two) this tells us that if phase one begins with stack configuration $\mathfrak{S}^-(u^{\curvearrowleft E})$, then if it terminates it must terminate with stack configuration of the form $\mathfrak{S}^-(u'^{\curvearrowleft E\restriction_{u'}})$ where $u = u'\ a_1 \cdots a_k$ and $a_k \cdots a_1 \in \mathfrak{D}^{\star}_{\mathcal{A}}(\mathcal{L}_i^{\leftarrow})$ and so $a_1 \cdots a_k \in \mathfrak{D}^{\star}_{\mathcal{A}}(\mathcal{L}_i)$. Conversely since $\mathcal{A}_i$ is deterministic and we can easily check by induction that $\mathcal{B}$ can proceed with phase one indefinitely until the stack becomes empty, we can see that if $a_1 \cdots a_k \in \mathfrak{D}^{\star}_{\mathcal{A}}(\mathcal{L}_i)$ then $\mathcal{B}$ can begin phase one with $\mathfrak{S}^-(u^{\curvearrowleft E})$ and end with $\mathfrak{S}(u'^{\curvearrowleft E\restriction_{u'}})$ with u and u' as above.

Now suppose that at the end of phase one the stack configuration is $\mathfrak{S}^-(u'^{\curvearrowleft E'})$. We may again argue by induction on k that after k steps of phase two we have a stack $\mathfrak{S}^-(v^{\curvearrowleft F})$ such that $v^{\curvearrowleft F} \in \mathfrak{D}_{\mathcal{A}}(\mathbf{NWord}(\mathfrak{V}(\Sigma)))$ and $v^{\curvearrowleft F} = u'^{\curvearrowleft E'}\ a_1 \cdots a_k$ for some $a_1, \ldots, a_k$ and moreover $\mathbf{q}'$ is the state in which $\mathcal{A}'_i$ would be after reading $\boldsymbol{symb}(a_1) \cdots \boldsymbol{symb}(a_k)$. The only substantive part of the induction step is when pushing an element with $\boldsymbol{symb}$ of the form $\mathring{a}$ onto the stack. For this consider the first part of Lemma 4.3. Since the top_2 stack prior to this operation must be the summary of the word hitherto created, the highest $\dot{b}$ that is not followed immediately by a $\mathring{b}'$ must indeed be the most recent unmatched target.

Given the condition for terminating phase two, this tells us that if phase two begins with the stack $\mathfrak{S}^-(u'^{\curvearrowleft E'})$ and terminates with the stack $\mathfrak{S}^-(v^{\curvearrowleft F})$ where $v = u'^{\curvearrowleft E'}\ a_1 \cdots a_k$, then $a_1 \cdots a_k \in \mathcal{A}'_i$, and so $a_1 \cdots a_k \in \mathcal{L}'_i$. Moreover we must have $v^{\curvearrowleft F} \in \mathcal{L}$ since it is ensured that the $\boldsymbol{st}_{\mathcal{A}}$ of the top element is an accepting state of $\mathcal{A}$. Since $\mathcal{B}$ is allowed to

simulate the addition of an arbitrary symbol emanating a pointer (except for an element in $\dot{\Sigma}$ when there is no corresponding $\mathring{\Sigma}$) the converse must also hold.

By combining these properties of phases one and two we may thus conclude that $\mathcal{B}$ emits an e_i edge precisely when required and that starting with a stack corresponding to an element of $\mathcal{L}$ ensures that the stack at the end of phase 2 also corresponds to an element of $\mathcal{L}$. Therefore all configurations of $\mathcal{B}$ reachable from a configuration encoding an element of the domain of the rewrite system do themselves encode elements in the domain of the rewrite system. In order to ensure that the domain of the ϵ-closure contains the correct elements we can add a fresh automaton edge label ρ and map it to the rewrite rule $\{\epsilon\} \longrightarrow \mathcal{L}$. By the definition of ϵ-closure for CPDA, the ρ edges will not appear in the graph provided that we ensure the initial configuration is never subsequently reached. This requirement can be fulfilled by choosing a fresh initial control-state that is never reused.

□

Lemma 4.5. *Every sum-rat suffix-rewrite graph is the ϵ-closure of some 2-CPDA graph.*

Proof. Consider a sum-rat suffix-rewrite system:

$$\mathcal{RW} := \left\langle \mathcal{L}, \mathcal{L}_1 \xrightarrow{e_1} \mathcal{L}'_1, \ldots, \mathcal{L}_k \xrightarrow{e_k} \mathcal{L}'_k \right\rangle$$

Let $\mathcal{A}$ be a *deterministic* nested-word automaton recognising $\mathcal{L}$. Consider the following suffix rewrite system:

$$\mathcal{RW}' := \left\langle \mathfrak{D}_{\mathcal{A}}(\mathcal{L}), \mathfrak{D}^{\star}_{\mathcal{A}}(\mathcal{L}_1) \xrightarrow{e_1} \mathfrak{D}^{\star}_{\mathcal{A}}(\mathcal{L}'_1), \ldots, \mathfrak{D}^{\star}_{\mathcal{A}}(\mathcal{L}_k) \xrightarrow{e_k} \mathfrak{D}^{\star}_{\mathcal{A}}(\mathcal{L}'_k) \right\rangle$$

Let $\mathcal{A}_i$ be a deterministic non-nested-word automaton recognising the regular language ${\mathcal{L}_i}^{\leftarrow}$. Let $\mathcal{A}'_i$ be a finite automaton recognising the regular language $\mathcal{L}'_i$. We can construct a 2-CPDA $\mathcal{B}$ in a manner very similar to the 2-PDA in the proof of Lemma 4.4. Indeed phase 2 is identical with the exception that all $push_1$ operations should now attach 2-links. This works since $\mathcal{L}'_i$ is a regular language as with the case for rat-rat rewrite systems. We thus give only the analogue for phase one here (in preparation to emit an e_i labelled edge):

- Set $\mathbf{q} := q_0$ where q_0 is the initial state of $\mathcal{A}_i$ and $\boldsymbol{cld}' := \mathbf{t}$:
- *Phase 1:* If the value of $\mathbf{q}$ is an accepting state of $\mathcal{A}_i$, then a non-deterministic choice is made whether to jump to the second phase (as in proof of Lemma 4.4) or to continue with *phase 1*. If it is not accepting, then *phase 1* must continue.
- – If $\boldsymbol{symb}(top_1) = a \in \Sigma$, then a pop_1 is performed and we set $\mathbf{q} := \delta_{\mathcal{A}_i}(a, \mathbf{q})$ and $\boldsymbol{cld}' := \boldsymbol{cld}(top_1)$
  - If $\boldsymbol{symb}(top_1) = \mathring{a} \in \mathring{\Sigma}$, set $\mathbf{q} := \delta_{\mathcal{A}_i}(a, \mathbf{q})$ and then make a non-deterministic choice to either:
    - Perform a pop_2 and set $\boldsymbol{cld}' := \mathbf{f}$.
    - Perform a pop_1 to result in an element $\dot{b} \in \dot{\Sigma}$ (by point 1 of Lemma 4.3) and set $\mathbf{q} := \delta_{\mathcal{A}_i}(a, \delta_{\mathcal{A}_i}(b, \mathbf{q}))$. Then perform *collapse*. Set $\boldsymbol{cld}' := \boldsymbol{cld}(\dot{b})$.
  - If $\boldsymbol{symb}(top_1) \in \dot{\Sigma}$, then we must terminate with failure (i.e. no edge is emitted).

We argue by induction on k that after k iterations of phase one starting with a stack $\mathfrak{S}^{\mathcal{C}-}(u^{\curvearrowleft E})$ the stack will be of the form $\mathfrak{S}^{\mathcal{C}-}({u'}^{\curvearrowleft E\restriction u'})$ such that $u = u'v$
with $v = a_1u_1b_1a_2u_2b_2 \cdots a_ku_kb_k$ where $u_i = b_i = \epsilon$ iff $a_i \notin \mathbf{img}(E \restriction_v)$ otherwise $E(b_i) = a_i$. Moreover $\mathbf{q}$ is the state in which $\mathcal{A}_i$ would be after reading $b_ka_kb_{k-1}a_{k-1} \cdots b_1a_1$. The induction step is provided by Lemma 4.3 points two and three.

Conversely it must be possible for the automaton to arrive after k steps to any such $\mathfrak{S}^{\mathcal{C}-}(u'^{\curvearrowleft E\restriction u'})$. This is because the only reason the automaton will stall is if it begins an iteration with a symbol of the form $\dot{a}$ on top. But this will only happen if the corresponding $\mathring{a}'$ was discarded previously without discarding $\dot{a}$ (by means of a pop_2). Thus one could reach the representation of the word resulting from discarding $\dot{a}$ by instead going with the pop_1; *collapse* operation at this previous point when pop_2 was instead used. Opting not do discard $\dot{a}$ corresponds to the case when one does not want to include $\dot{a}$ in the suffix but one does want to include the corresponding $\mathring{a}$—*i.e.* the case when the pointer from a' to a is not included in the suffix despite its source being included.

This ensures that a phase one may start with $\mathfrak{S}^{\mathcal{C}-}(u^{\curvearrowleft E})$ and end with $\mathfrak{S}^{\mathcal{C}-}(u'^{\curvearrowleft E\restriction u'})$ just in case $w = tv$ and $v^{\curvearrowleft E\restriction v} \in \mathcal{L}_i$. As with Lemma 4.4 we can set up 'dummy ρ transitions' for the rewrite rule $\{\epsilon\} \longrightarrow \mathcal{L}$ to correctly set the domain. □

5. Nested-Trees and Automata

In order to establish the converse direction, from 2-CPDA to rewrite systems, we use an intermediate representation of the graphs; this is a kind of 'automaticity' for nested-words. There are two main variants which we call *isophilic* and *dendrisophilic*, together with an auxiliary notion called *symmetric-dendrisophilic*. We will also introduce a notion of automaticity with respect to *'nested-trees'*, which will be able to describe 3_2-CPDA graphs and lead to our new decidability result for first-order logic. Since even the nested-word versions of automaticity will make use of nested-trees as 'convolutions', we begin by considering nested-trees and various types of nested-tree automaton and their properties

5.1. **Nested-Trees.** A Σ-*labelled tree* with *degree bounded by* d (for $d \in \mathbb{N}$) is a map $T : \mathbf{dom}(T) \longrightarrow \Sigma$ where $\mathbf{dom}(T) \subseteq [1..d]^*$ is prefix-closed. A maximal element of $\mathbf{dom}(T)$ is known as a *leaf*, ϵ as *the root* and a *branch* is a path from the root to a leaf.

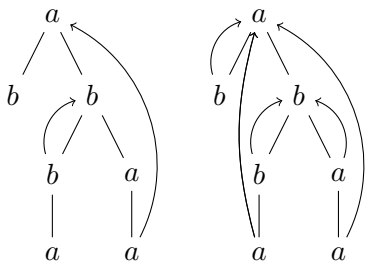


Figure 4: Path-wise nested-tree and nested-tree

Definition 5.1. A *path-wise nested-tree* $T^{\curvearrowleft E}$ consists of a tree T together with a partial map $E : \mathbf{dom}(T) \rightharpoonup \mathbf{dom}(T)$ such that every branch is a nested-word. A *nested-tree* is a path-wise nested-tree $T^{\curvearrowleft E}$ such that for each leaf v and each node $v' \in \mathbf{img}(E)$ such that $v' \sqsubset v$ there exists a $u \in \mathbf{dom}(E)$ with $v' \sqsubset u \sqsubseteq v$ such that $E(u) = v'$.

The additional constraint for nested-trees ensures that if a node of the tree is the target of a pointer along some path, then it is the target of a pointer along every path passing through it. This definition is very similar to that of Alur *et al.* [1] with the difference that our trees are finite and as with nested-words, there is no notion of a node being the target of a pointer with no source. We illustrate the idea in Figure 4.

Definition 5.2. Let $\mathbf{NTree}(\Sigma)$ be the set of Σ-labelled nested-trees (with any degree-bound). We write $\mathbf{NTree}_k(\Sigma)$ to denote the set of nested-trees that have at most k leaves (branches).

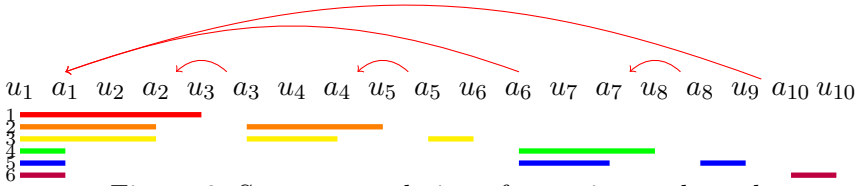


Figure 6: Summary ordering of a semi-nested word

5.2. **Branch and Summary Ordering.** We place a linear ordering on the branches of a tree following the manner in which a left-to-right depth first search would occur.

Definition 5.3. Let $l := a_1 \cdots a_r$ and $l' := a'_1 \cdots a'_{r'}$ be *leaves* of a (nested-)tree $T^{\frown E}$ and let b be the branch tipped by l and b' by l'. We say $b \prec b'$ iff $a_i < a'_i$ where i is the least j such that $a_j \neq a'_j$.

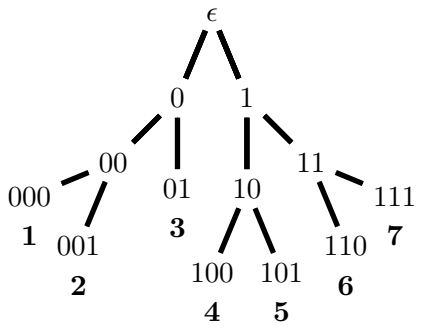


Figure 5: Branch ordering

Figure 5 exhibits this ordering. We treat branches of a nested-tree as nested-words. Note though that distinct branches might be associated with the 'same' nested-word (same pointer structure and node labels). For a tree $T^{\frown E}$ we write $T^{\frown E}_{\prec_i}$ to denote the ith nested-word in its branch ordering.

Similarly we place a linear ordering on the *summaries* of certain prefixes of *semi*-nested-words. This amounts to the branch ordering on the standard tree representation thereof.

Definition 5.4. Let $w^{\frown E}$ be a semi-nested-word and let $succ(u_1) < succ(u_2) < \cdots < succ(u_r)$ be an exhaustive ordered list of the nodes in $\mathbf{dom}(E)$ (since the first node cannot be in $\mathbf{dom}(E)$ all must be of the form $succ(u)$). Writing $v_i{}^{\frown E_i}$ for the prefix of $w^{\frown E}$ ending with node u_i we define the summary ordering $\blacktriangleleft$ of $w^{\frown E}$ to be:

$$\ulcorner v_1{}^{\frown E_1} \urcorner \blacktriangleleft \ulcorner v_2{}^{\frown E_2} \urcorner \blacktriangleleft \cdots \ulcorner v_r{}^{\frown E_r} \urcorner \blacktriangleleft \ulcorner w^{\frown E} \urcorner$$

.

We treat summaries as *non-nested* words and write $w^{\frown E}_{\blacktriangleleft_i}$ to denote the ith word in the summary ordering of $w^{\frown E}$.

5.3. **Traditional Nested-Tree Automata.** We begin with Alur *et al.*'s nested-tree automata (NTA) [1]. These read a nested-tree from top to bottom and have access to the state at the target of a pointer when reading its source. Formally an NTA is a tuple:

$$\mathcal{A} \;=\; \langle \Sigma, Q, I, \delta_{\oplus}, \delta, \delta_{\ominus}, F \rangle$$

such that Σ is a finite alphabet, Q is a finite set of control-states, $I \subseteq Q$ is a set of initial states, $F \subseteq Q$ is a set of accepting states and $\delta_{\oplus}, \delta, \delta_{\ominus}$ are transition functions with the following types for some $k \in \mathbb{N}$:

$$\delta_{\ominus} \;:\; \Sigma \times Q \longrightarrow 2^{\bigcup_{i=1}^{k} Q^i} \qquad \delta \;:\; \Sigma \times Q \longrightarrow 2^{\bigcup_{i=1}^{k} Q^i} \qquad \delta_{\oplus} \;:\; \Sigma \times Q \times Q \longrightarrow 2^{\bigcup_{i=1}^{k} Q^i}$$

A *run-tree* of $\mathcal{A}$ on $T^{\frown E} \in \mathbf{NTree}(\Sigma)$ is a Q-labelled tree $\mathcal{R}$ such that $\mathbf{dom}(\mathcal{R}) = \mathbf{dom}(T) \cup \{\ u0\ :\ u \in \mathbf{dom}(T) \text{ is maximal }\ \}$ and:

- $\mathcal{R}(\epsilon) \in I$
- If $u \in \mathbf{img}(E)$ and u has i children in $\mathcal{R}$ (for some $1 \leq i \leq k$), then $(\mathcal{R}(u_1), \ldots, \mathcal{R}(u_i)) \in \delta_{\ominus}(T(u), \mathcal{R}(u))$.
- If $u \in \mathbf{dom}(T) - (\mathbf{dom}(E) \cup \mathbf{img}(E))$ and u has i children ($1 \leq i \leq k$), $(\mathcal{R}(u_1), \ldots, \mathcal{R}(u_i)) \in \delta(T(u), \mathcal{R}(u))$.
- If $u \in \mathbf{dom}(T)$ and u has i children (for some $1 \leq i \leq k$), then $(\mathcal{R}(u_1), \ldots, \mathcal{R}(u_i)) \in \delta_{\oplus}(T(u), \mathcal{R}(E(u)), \mathcal{R}(u))$.

Note that this is well-defined as by construction the nodes in $\mathcal{R}$ with children are precisely the nodes in T. A run-tree is deemed *accepting* just in case all of its leaves are labelled with a state in F. The language defined by $\mathcal{A}$ is:

$$\mathcal{L}(\mathcal{A}) \ :=\ \{T^{\frown E} \in \mathbf{NTree}(\Sigma)\ :\ \mathcal{A} \textit{ accepts } T^{\frown E}\}$$

We say that $\mathcal{A}$ is *deterministic* if I is a singleton and for each d (with $1 \leq d \leq k$) there is at most one d-tuple in any given set belonging to the image of a tranition function of $\mathcal{A}$. Intuitively this means that the behaviour of $\mathcal{A}$ on any given input tree is uniquely determined (it has at most one run tree on the input).

5.4. **Some Automaton Variants.** Nested-tree automata are not closed under Boolean operations. The three devices introduced here may be viewed as special-cases of nested-tree automata, and could be considered the *cascade product* of two automata. The idea is that they have better properties for defining notions of 'automaticity' and moreover are the right devices for the precise characterisations that we are seeking. They take the form $\mathcal{B}^{\mathcal{C}}$ where $\mathcal{B}$ is a nested-tree automaton and $\mathcal{C}$ is a conventional finite tree automaton that has access to the state of $\mathcal{B}$. Formally, if $\mathcal{B} = \langle \Sigma, Q_{\mathcal{B}}, I_{\mathcal{B}}, \delta_{\oplus\mathcal{B}}, \delta_{\mathcal{B}}, \delta_{\ominus\mathcal{B}} \rangle$ we require $\mathcal{C}$ to have the form $\mathcal{C} = \langle \Sigma, Q_{\mathcal{B}}, Q_{\mathcal{C}}, I_{\mathcal{C}}, \delta_{\mathcal{C}}, F_{\mathcal{C}} \rangle$ where $I \ \subseteq\ Q_{\mathcal{C}}$ is a set of initial states, $F_{\mathcal{C}} \ \subseteq\ Q_{\mathcal{C}}$ is a set of final states and:

$$\delta_{\mathcal{C}}\ :\ \Sigma\ \times\ Q_{\mathcal{B}}\ \times\ Q_{\mathcal{C}}\ \longrightarrow\ 2^{\bigcup_{i=1}^{k} Q_{\mathcal{C}}^{i}}$$

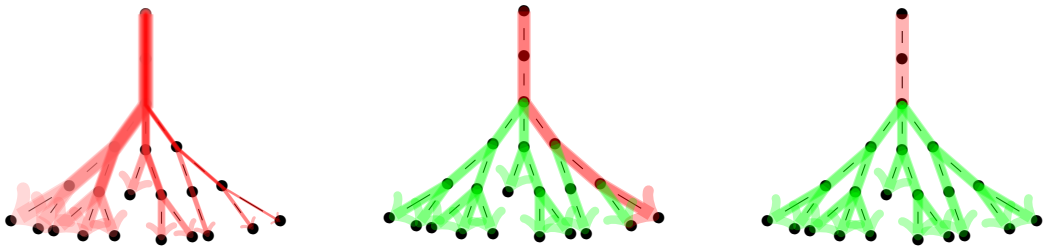
. We do not require $\mathcal{B}$ to have any final states as they are irrelevant to the definition of accepting run.

A *run-tree* $\mathcal{R}$ for $\mathcal{B}^{\mathcal{C}}$ on $T^{\frown E} \in \mathbf{NTree}(\Sigma)$ is a $(Q_{\mathcal{B}}\ \times\ Q_{\mathcal{C}})$-labelled tree such that:

- $\pi_1(\mathcal{R})$ is a run-tree for $\mathcal{B}$.
- $\pi_2(\mathcal{R}(\epsilon)) = q$ for some $q \in I_{\mathcal{C}}$.
- If u has i children and $\mathcal{R}(u) = (p, q)$, then for each $1 \leq j \leq i$ we must have $\pi_2(\mathcal{R}(uj)) = q_j$ where $(q_1, \ldots, q_i) \in \delta_{\mathcal{C}}(T(u), p, q)$.

A run-tree is deemed *accepting* just in case all of its leaves have a label of the form $(\ \_\ , q)$ for some $q \in F_{\mathcal{C}}$. The language recognised by $\mathcal{B}^{\mathcal{C}}$ consists of precisely the elements of $\mathbf{NTree}(\Sigma)$ for which $\mathcal{B}^{\mathcal{C}}$ has an accepting run-tree.

In the special case when $\mathcal{B}$ is a *deterministic* nested-*word* automaton acting on each branch of the tree individually we call $\mathcal{B}^{\mathcal{C}}$ a *path-nested automaton*. The determinism and branching-agnosticism of $\mathcal{B}$ ensures that the flow of information from $\mathcal{B}$ to $\mathcal{C}$ can only be one way—we can describe a path-nested automaton as having access to both the nesting structure and branching structure of the tree but 'not in combination'.



Path-Nested Spine Nested Trunk Nested

Figure 7: The manner in which path-nested, spine nested and trunk nested automata differ—the red (dark) lines indicate the branches along which $\mathcal{B}$ must be deterministic and ignore branching whilst the green (light) lines indicate the branches along which it is allowed to behave as an unrestricted tree automaton.

Definition 5.5. A path-nested automaton is an automaton $\mathcal{B}^{\mathcal{C}}$ such that $\mathcal{B}$ is deterministic and for any $(q_1, q_2, \ldots, q_k), (p_1, p_2, \ldots, p_l) \in S \in \mathbf{img}(\delta_{\oplus\mathcal{B}}) \cup \mathbf{img}(\delta_{\mathcal{B}}) \cup \mathbf{img}(\delta_{\ominus\mathcal{B}})$, $p_1 = p_2 = \cdots = p_l = q_1 = q_2 = \cdots = q_k$, where $\delta_{\oplus\mathcal{B}}, \delta_{\mathcal{B}}, \delta_{\ominus\mathcal{B}}$ are the transition functions of $\mathcal{B}$.

Despite the fact that $\mathcal{B}$ is a nested-tree automaton we may naturally view it as a nested-word automaton as it acts on each branch independently in a deterministic manner.

A different restriction of $\mathcal{B}$ gives a *spine nested automaton*, which behaves like a path-nested automaton along *'the spine'* of an input tree—the $\prec$-greatest path, which consists of right-most children. Elsewhere it can behave as an unrestricted nested-tree automaton. (A segment of a branch in a tree that is not part of the spine is often referred to as a *rib'*)

Definition 5.6. We call $\mathcal{B}^{\mathcal{C}}$ a *spine nested automaton* if it has a subset of control-states $P \subseteq Q_{\mathcal{B}}$ such that:

- $I_{\mathcal{B}}$ is a singleton set (just one initial state) with element belonging to P.
- For every $p \in P$ there exists a unique $p' \in P$ such that $\delta(\_, p)$, $\delta_{\ominus}(\_, p)$ and $\delta_{\oplus}(\_, \_, p)$ map to sets S such that for every $\vec{v} \in S$, $\pi_{|\vec{v}|}(\vec{v}) = p'$.

Finally we have a *trunk nested automaton*, which is required to behave like a path-nested automaton only along the 'trunk' of the tree (the bit prior to any branching).

Definition 5.7. We call $\mathcal{B}^{\mathcal{C}}$ a *trunk nested automaton* if it has a subset of control-states $P \subseteq Q_{\mathcal{B}}$ such that:

- $I_{\mathcal{B}}$ is a singleton set (just one initial state) with element belonging to P.
- For every $p \in P$ there exists a unique $p' \in P$ such that $\delta(\_, p)$, $\delta_{\ominus}(\_, p)$ and $\delta_{\oplus}(\_, \_, p)$ map to sets S such that for every $\vec{v} \in S$, $|\vec{v}| = 1$ implies $\vec{v} = p'$.

These three variants are illustrated in Figure 7.

5.5. **Bottom-Up Automata.** It will be useful to have a notion of bottom-up nested-tree automaton (BUNTA). Whilst the NTA introduced above begin at the root of a tree and move down towards its leaves, a bottom-up automaton begins at the leaves and moves towards the root. We are only interested in these devices when it is possible to determinise them. We thus restrict our attention to the case when the trees fed to the automaton have

a bounded number of branches, or equivalently a bounded number of leaves. Formally a k-BUNTA $\mathcal{A}$ acts on Σ-labelled nested-trees with at most k branches where:

$$\mathcal{A} = \langle \Sigma, Q, I_1, I_2, \ldots, I_k, \delta_\oplus, \delta, \delta_\ominus, F \rangle$$

where Q is a finite set of control-states; $I_j \subseteq Q$ is a set of initial states for the jth leaf for each $1 \leq j \leq k$; $F \subseteq Q$ is a set of final states and there are transition functions:

$$\delta_\oplus \ : \ \Sigma \times \bigcup_{1 \leq j \leq k} Q^j \longrightarrow 2^Q \qquad \delta \ : \ \Sigma \times \bigcup_{1 \leq j \leq k} Q^j \longrightarrow 2^Q$$

$$\delta_\ominus \ : \ \Sigma \times \bigcup_{1 \leq j \leq k} Q^j \times \bigcup_{1 \leq j \leq k} Q^j \longrightarrow 2^Q$$

We still intend $\delta_\oplus$ to act on sources of pointers and $\delta_\ominus$ on targets, but because we are reading the tree in a bottom-up manner, sources will be read prior to targets and so the types of $\delta_\oplus$ and $\delta_\ominus$ are 'the other way around' with respect to the original top-down version.

A *run-tree* of $\mathcal{A}$ on $T^{\frown E} \in \mathbf{NTree}_k(\Sigma)$ is a Q-labelled tree $\mathcal{R}$ such that $\mathbf{dom}(\mathcal{R}) = \mathbf{dom}(T) \ \cup \ \{ \ u0 \ : \ u \in \mathbf{dom}(T) \text{ is maximal } \}$ and where $b_1 \prec b_2 \prec \cdots \prec b_m$ for $m \leq k$ is the complete branch ordering of $\mathcal{R}$ and l_i is the leaf tipping b_i for each $1 \leq i \leq m$ we have:

- $\mathcal{R}(l_j) \in I_j$ for each $1 \leq j \leq m$
- If $u \in \mathbf{dom}(E)$ and u has i children $u_1, \ldots, u_i$, then $\mathcal{R}(u) \in \delta_\oplus(T(u), (\mathcal{R}(u_1), \ldots, \mathcal{R}(u_i)))$.
- If $u \notin \mathbf{dom}(E) \cup \mathbf{img}(E)$ and u has i children $u_1, \ldots, u_i$, then $\mathcal{R}(u) \in \delta(T(u), (\mathcal{R}(u_1), \ldots, \mathcal{R}(u_i)))$.
- If $u \in \mathbf{img}(E)$ and u has i children $u_1, \ldots, u_i$, then

  $$\mathcal{R}(u) \in \delta_\ominus(T(u), (\mathcal{R}(v_1), \ldots, \mathcal{R}(v_j)), (\mathcal{R}(u_1), \ldots, \mathcal{R}(u_i)))$$

  where $v_1 \prec v_2 \prec \cdots \prec v_j$ and $E^{-1}(u) = \{ \ v_1, \ldots, v_j \ \}$.

As before, note that the nodes with children in $\mathcal{R}$ are precisely the nodes in T and the leaves of $\mathcal{R}$ are precisely the nodes that $\mathcal{R}$ posesses but which T lacks.

We say that $\mathcal{R}$ is *accepting* just in case $\mathcal{R}(\epsilon) \in F$. The language defined by $\mathcal{A}$ is:

$$\mathcal{L}(\mathcal{A}) \ := \ \{T^{\frown E} \in \mathbf{NTree}_k(\Sigma) \ : \ \mathcal{A} \textit{ accepts } T^{\frown E}\}$$

The top-down and bottom-up automata are equi-expressive.

Lemma 5.8. *Let $\mathcal{A}$ be an NTA such that $\mathcal{L}(\mathcal{A}) \subseteq \mathbf{NTree}_k(\Sigma)$. It is then the case that there exists a k-BUNTA $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$. Conversely let $\mathcal{A}'$ be a k-BUNTA. Then there exists an NTA $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.*

Proof. Let $\mathcal{A} = \langle \Sigma, Q, I, \delta_\oplus, \delta, \delta_\ominus, F \rangle$ be an NTA such that $\mathcal{L} \subseteq \mathbf{NTree}_k(\Sigma)$. We define an equivalent k-BUNTA $\mathcal{A}' = \langle \Sigma, Q', I'_1, I'_2, \ldots, I'_k, \delta'_\oplus, \delta', \delta'_\ominus, F' \rangle$ as follows:

- Take $Q' := Q \cup Q \times Q$. For the purposes of this proof, let us us write $(\_, q)$ to mean 'either q or (p, q) for any p'.
- Take $F' := I$ and $I'_1 := I'_2 := \cdots := I'_k := F$
- Take $\delta'_\oplus$ to be the function such that $(p, q) \in \delta'_\oplus(a, (\_, q_1), (\_, q_2), \ldots, (\_, q_j))$ iff $(q_1, q_2, \ldots, q_j) \in \delta_\oplus(a, p, q)$ for every $q_1, q_2, \ldots, q_j, p, q \in Q$.
- Take δ' to be the function such that $q \in \delta'(a, (\_, q_1), (\_, q_2), \ldots, (\_, q_j))$ iff $(q_1, q_2, \ldots, q_j) \in \delta(a, q)$ for every $q_1, q_2, \ldots, q_j, q \in Q$.

- Take $\delta'_\ominus$ to be the function such that $p \in \delta'_\ominus(a, (p, r_1), \ldots, (p, r_l), (\_, q_1), \ldots, (\_, q_j))$ iff $(q_1, q_2, \ldots, q_j) \in \delta_\ominus(a, p)$. Any input to $\delta'_\ominus$ not matching this pattern is mapped to the empty set $\emptyset$.

By construction, there is an accepting run tree $\mathcal{R}$ of $\mathcal{A}$ on an input $T^{\curvearrowleft E}$ iff there exists a run tree $\mathcal{R}'$ of $\mathcal{A}'$ on $T^{\curvearrowleft E}$ such that for every $u \in \mathbf{dom}(\mathcal{R})$ we have $\mathcal{R}'(u) = \mathcal{R}(u)$ when $u \notin \mathbf{dom}(E)$, and $\mathcal{R}'(u) = (p, q)$ when $\mathcal{R}(u) = q$ and $p = \mathcal{R}'(E(u)) = \mathcal{R}(E(u))$ when $u \in \mathbf{dom}(E)$. Thus in particular $\mathcal{A}$ accepts $T^{\curvearrowleft E}$ iff $\mathcal{A}'$ accepts $T^{\curvearrowleft E}$.

Now let us consider the converse. Let $\mathcal{A}' = \langle \Sigma, Q', I'_1, I'_2, \ldots, I'_k, \delta'_\oplus, \delta', \delta'_\ominus, F' \rangle$ be a k-BUNTA. We construct an equivalent NTA $\mathcal{A} = \langle \Sigma, Q, I, \delta_\oplus, \delta, \delta_\ominus, F \rangle$. This time it is the top-down automaton doing the guessing at the target of a pointer using its $\delta_\oplus$ function. The guess specifies several control-states that it postulates will occur at the corresponding sources of the pointers by means of a *partial* function from sets of branch indices. The construction implies that $\mathcal{A}'$ has an accepting run-tree $\mathcal{R}'$ on an input $T^{\curvearrowleft E}$ iff $\mathcal{A}$ has an accepting run-tree $\mathcal{R}$ on $T^{\curvearrowleft E}$ such that for all $u \notin \mathbf{img}(\mathcal{R})$, $\mathcal{R}(u) = (q, I)$ iff u is on the ith branch in the branch ordering for precisely those $i \in I$ and $\mathcal{R}'(u) = q$, and for all $u \in \mathbf{img}(\mathcal{R})$, $\mathcal{R}(u) = (f, q, I)$ iff q and I are as before, and $\mathbf{dom}(f) = \{I_1, \ldots, I_l\}$ where $E^{-1}(u) = \{u_1, \ldots, u_l\}$, with u_j lying on the ith branch iff $i \in I_j$, and $f(I_j) = \mathcal{R}'(u_j)$.

- Take $Q := \mathcal{P} \times Q' \times 2^{[1..k]} \cup Q' \times 2^{[1..k]}$, where $\mathcal{P}$ is the set of partial functions $f : 2^{[1..k]} \rightharpoonup Q$ where all $S \in \mathbf{dom}(f)$ are disjoint and are intervals (*i.e.* such that if $i, j \in S$ and $i < r < j$ then $r \in S$). We have an ordering $L < L'$ on the elements of $\mathbf{dom}(f)$ where $L < L'$ iff $l < l'$ for any representatives $l \in L$ and $l' \in L'$. Write $(\_, q, L)$ to mean either (q, L) or (f, q, L) for some $f \in \mathcal{P}$. We always implicitly assume that for (f, q, L), $\bigcup \mathbf{dom}(f) = L$.
- Take $F := \{ (q, \{j\}) : q \in I'_j, j \in [1..k] \}$ and $I := (1 + \mathcal{P}) \times F' \times \{S \subseteq [1..k] : S \text{ downward closed}\}$.
- Take $\delta_\ominus$ to be the function such that
$$((\_, q_1, L_1), (\_, q_2, L_2), \ldots, (\_, q_j, L_j)) \in \delta_\ominus(a, (f, q, L))$$
iff $L_1 \uplus \cdots \uplus L_j = L$ with $L_1 < L_2 < \cdots < L_j$ and $q \in \delta'_\ominus(f(S_1), \ldots, f(S_i), q_1, \ldots, q_j)$ where $\mathbf{dom}(f) = \{S_1 < S_2 < \cdots < S_i\}$.
- Take δ to be the function such that
$$((\_, q_1, L_1), (\_, q_2, L_2), \ldots, (\_, q_j, L_j)) \in \delta(a, (q, L))$$
iff $L_1 \uplus \cdots \uplus L_j = L$ with $L_1 < L_2 < \cdots < L_j$ and $q \in \delta'(a, q_1, \ldots, q_j)$.
- Take $\delta_\oplus$ to be the function such that
$$((\_, q_1, L_1), (\_, q_2, L_2), \ldots, (\_, q_j, L_j)) \in \delta_\oplus(a, (f, \_, \_), (q, L))$$
iff $L_1 \uplus \cdots \uplus L_j = L$ with $L_1 < L_2 < \cdots < L_j$ and $q \in \delta'_\oplus(q_1, \ldots, q_j)$ and $q = f(L)$.

□

5.6. **Boolean Closure.** In order to develop a notion of automaticity, we need Boolean closure for the languages recognised by these automata. We are only interested in top-down automata for this result, although bottom-up automata will come into the proof. It is easy to get closure under intersection since all of these automata can be run in parallel:

Lemma 5.9. *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be languages of nested-trees recognised by nested-tree automata. Then we can construct a nested-tree automaton recognising $\mathcal{L} := \mathcal{L}_1 \cap \mathcal{L}_2$. Moreover if $\mathcal{L}_1$ and $\mathcal{L}_2$ are both recognised by a path-nested, spine nested or trunk nested automata, then $\mathcal{L}$ is also recognised by such an automaton.*

It is known that nested-word automata can be determinised and hence complemented [3][2]. Unfortunately nested-tree automata can be neither determinised nor complemented in general [1]. We can, however, complement a path-nested automaton $\mathcal{B}^{\mathcal{C}}$.

Lemma 5.10. *Let $\mathcal{A}$ be a path-nested automaton operating over the alphabet Σ. There exists a path-nested automaton $\overline{\mathcal{A}}$ recognising $\overline{\mathcal{L}(\mathcal{A})} := \mathbf{NTree}(\Sigma) - \mathcal{L}(\mathcal{A})$.*

Proof. Let $\mathcal{B}^{\mathcal{C}}$ be a path-nested automaton. The automaton $\mathcal{B}$ is deterministic and so we only need to complement the finite tree automaton $\mathcal{C}$, which is standard. □

By contrast, complementing spine nested and trunk nested automata is impossible in general since both of these variants act as general nested-tree automata on certain components of the tree. Adapting Alur *et al.*'s proof for the determinisation of nested-words [3][2] we can, however, complement a set of nested-trees with a bounded number of leaves. We obtain this result by first showing that bottom-up automata can be determinised. This allows them to be complemented and so we can then just appeal to Lemma 5.8.

Lemma 5.11. *Let $\mathcal{A}$ be a k-BUNTA. Then there exists a* deterministic *k-BUNTA $\mathcal{A}'$—that is one such that the image of the transition functions consists of singleton sets and the leaf-state sets are also singleton—such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.*

Proof. We follow the same proof idea as used by Alur *et al.* for nested-words [3][2]. Recall that bottom-up automata on standard trees can be determinised using a power set contruction. If the bottom up automaton simply skips over pointers, then such a power set construction would still work. This is in effect what happens, except that before skipping over a pointer, a new automaton is spawned that reads the sub-tree between the sources and targets. This spawned automaton is responsible for considering all possible combinations of control-states at the sources; in order to keep track of the required information we rely on the fact that the tree has at most k branches. The result of this automaton is additionally made available at the target of the pointer.

With this intuition in mind, we define $\mathcal{A}' = \langle \Sigma, Q', I'_1, I'_2, \ldots, I'_k, \delta'_\oplus, \delta', \delta'_\ominus, F' \rangle$ from $\mathcal{A} = \langle \Sigma, Q, I_1, I_2, \ldots, I_k, \delta_\oplus, \delta, \delta_\ominus, F \rangle$ to be as follows:

- $Q' := 2^{Q \times \bigcup_{0 \leq i \leq k} Q^i} \cup 2^{Q \times \bigcup_{0 \leq i \leq k} Q^i} \times 2^{Q \times \bigcup_{0 \leq i \leq k} Q^i}$, writing $(\_, T)$ to denote either T or (S, T) for some S.
- $I'_j := \{ I_j \times \{\epsilon\} \}$ for each $1 \leq i \leq k$.
- $\delta'_\oplus(a, (\_, T_1), \ldots, (\_, T_j)) := \{(\{\ (q', w_1 w_2 \cdots w_j)\ :\ q' \in \delta_\oplus(a, q_1, \ldots, q_j) \text{ and } (q_i, w_i) \in T_i \text{ for each } 1 \leq i \leq j\ \}, \{\ (q, q)\ :\ q \in Q\ \})\}$
- $\delta'(a, (\_, T_1), \ldots, (\_, T_j)) := \{\{\ (q', w_1 w_2 \cdots w_j)\ :\ q' \in \delta(a, q_1, \ldots, q_j) \text{ and } (q_i, w_i) \in T_i \text{ for each } 1 \leq i \leq j\ \}\}$
- $\delta'_\ominus(a, ((S'_1, T'_1), \ldots, (S'_l, T'_l)), ((\_, T_1), \ldots, (\_, T_j))) :=$
  $\{\{\ (q', w'_1 w'_2 \cdots w'_l)\ :\ q' \in \delta_\ominus(a, p_1, \ldots, p_l, q_1, \ldots, q_j) \text{ for some } (q_i, w_i) \in T_i \text{ for each } 1 \leq i \leq j \text{ such that } w_1 w_2 \cdots w_j = p_1 p_2 \cdots p_l, \text{ and } (p_i, w'_i) \in S'_i \text{ for each } 1 \leq i \leq l\ \}\}$
- $F' := \{ F \times \{\epsilon\} \}$

We claim that the unique run tree $\mathcal{R}$ of $\mathcal{A}'$ on a nested-tree $T^{\frown_E} \in \mathbf{NTree}_k(\Sigma)$ has the following properties:

- At a node $u \notin \mathbf{dom}(E)$ we have $\mathcal{R}(u) = T \in 2^{Q \times \bigcup_{0 \leq i \leq k} Q^i}$. At a node $u \in \mathbf{dom}(E)$ we have $\mathcal{R}(u) = (S, T) \in 2^{Q \times \bigcup_{0 \leq i \leq k} Q^i} \times 2^{Q \times \bigcup_{0 \leq i \leq k} Q^i}$. We keep this notation u, S and T in the items below.
- Let $N_{\leq u} := \{\ v \in \mathbf{dom}(E)\ :\ v \sqsubseteq u$ but $E(v) \sqsupset u$ and for all v' s.t. $v \sqsubset v' \sqsubset u, v' \in \mathbf{dom}(E)$ implies $E(v') \sqsubseteq u\ \}$. That is $N_{\leq}$ is the set of sources of pointers closest to u in the subtree rooted at u whose pointers have not yet been discharged by reaching their targets. Note that if $u \in \mathbf{dom}(E)$ then $N_{\leq u} = \{u\}$. We can order $N_{\leq u}$ using the branch ordering: $v_1 \prec v_2 \prec \cdots \prec v_m$. Then the set T consists of precisely those elements $(q, p_1 p_2 \cdots p_m)$ such that $\mathcal{A}$ could have a run-tree starting at the subtree with leaves $v_1, v_2, \ldots, v_m$ in respective states $p_1, p_2, \ldots, p_m$ ending at the node u in control-state q. If $N_{\leq u}$ is empty, then T consists of precisely those elements (q, ϵ) such that $\mathcal{A}$ would have a run-tree starting at the leaves of $T^{\curvearrowleft E}$ in initial states and ending at u in state q.
- Now consider $u \in \mathbf{dom}(E)$ so that (S, T) is the associated state. Let $N_{<u} := \{\ v \in \mathbf{dom}(E)\ :\ v \sqsubset u$ but $E(v) \sqsupset u$ and for all v' s.t. $v \sqsubset v' \sqsubset u, v' \in \mathbf{dom}(E)$ implies $E(v') \sqsubseteq u\ \}$—that is the same definition as $N_{\leq u}$ except that we consider the nodes $v \in \mathbf{dom}(E)$ that are *strictly below* u. Again order the members of $N_{<u}$ with the branch ordering: $v_1 \prec v_2 \prec \cdots \prec v_m$. Then the set S consists of precisely those elements $(q, p_1 p_2 \cdots p_m)$ such that $\mathcal{A}$ could have a run-tree starting at the subtree with leaves $v_1, v_2, \ldots, v_m$ in respective states $p_1, p_2, \ldots, p_m$ ending at the node u in control-state q. If $N_{<u}$ is empty, then T consists of precisely those elements (q, ϵ) such that $\mathcal{A}$ would have a run-tree starting at the leaves of $T^{\curvearrowleft E}$ in initial states and ending at u in state q.

It is mechanical to verify that the transition rules maintain these invariants on the assumption that they have been maintained whilst reading the nodes strictly below u in the subtree rooted at u. The third item of the invariant is necessary to verify that $\delta_{\ominus}$ preserves the second item of the invariant. The second item of the invariant implies (due to the choice of F') that the unique run-tree of $\mathcal{A}'$ is accepting iff $\mathcal{A}$ has a run-tree. □

Now we can get:

Lemma 5.12. *Fix $k \in \mathbb{N}$. Let $\mathcal{A}$ be a nested-tree automaton operating over the alphabet Σ. There exists a nested-tree automaton $\overline{\mathcal{A}}$ recognising $\mathbf{NTree}_k(\Sigma) - \mathcal{L}(\mathcal{A})$.*

Proof. Apply Lemma 5.8 to get a k-BUNTA $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A}) \cap \mathbf{NTree}_k(\Sigma)$. We may then determinise $\mathcal{A}'$ using Lemma 5.11. Since the automaton has a unique run-tree on any given $T^{\curvearrowleft E} \in \mathbf{NTree}_k(\Sigma)$, we may complement it by complementing its set of final states. The automaton $\overline{\mathcal{A}}$ may be obtained by applying Lemma 5.8 again to get a top-down NTA, which can then be intersected with a simple automaton recognising $\mathbf{NTree}_k(\Sigma)$. □

Remark 5.13. The reader may wonder why we do not encode elements of $\mathbf{NTree}_k(\Sigma)$ as nested-words and then appeal to the boolean closure of nested-word automata. Whilst such an encoding is possible it is perhaps not as neat as it would be for non-nested trees—encodings of branches would have to be embedded within one another rather than just be concatenated. This is due to the need to preserve well-nesting of pointers. It is felt that it is as easy to reformulate the proof in terms of nested-trees as it is to deal with encodings and decodings. The reader might also ask why we do not just forget about $\mathbf{NTree}_k(\Sigma)$ altogether and deal exclusively with nested-words. The reason for this lies in our application of them, which relies on having both the pointer *and genuine* branching structure.

And as a corollary to Lemma 5.9, Lemma 5.10 and Lemma 5.12 we have:

Lemma 5.14. *Over the universe* $\mathbf{NTree}(\Sigma)$ *path-nested automata are closed under Boolean operations. Over the universe* $\mathbf{NTree}_k(\Sigma)$ *for fixed* $k \in \mathbb{N}$ *nested-tree automata (which subsume spine and trunk nested automata) are closed under Boolean operations.*

Note that the lemma above guarantees only that the complement of a trunk nested automaton is a nested-tree automaton; it does not say that its complement can be represented by another trunk nested automaton. However, we can get:

Lemma 5.15. *Over the universe* $\mathbf{NTree}_2(\Sigma)$ *trunk nested automata are closed under complement and since they are closed under intersection are thus Boolean closed.*

Proof. Consider a trunk-nested automaton $\mathcal{B}^{\mathcal{C}}$. Decompose $\mathcal{B}$ into a nested-*word* automaton $\mathcal{B}_T$ that behaves as $\mathcal{B}$ when reading the trunk of the tree and nested-*word* automata $\mathcal{B}_{(q,p)}$ behaving as $\mathcal{B}^{\mathcal{C}}$ would starting in state (q,p) *after* branching from the trunk. Noting that the target-states of pointers targeting the trunk but sourced after branching can be treated as part of the input alphabet for the purposes of determinisation, we can determinise each nested-word automaton $\mathcal{B}_{q,p}$. We then define $\mathcal{B}'$ to be the nested-tree automaton that behaves as $\mathcal{B}_T$ along the trunk and then runs all of the $\mathcal{B}_{q,p}$ in parallel along every branch after branching. $\mathcal{B}'$ must be deterministic in the sense that given any input tree it has at most one run. It is also deterministic along the trunk and so meets the criteria for being spine-nested. We then define $\mathcal{C}'$ to behave as $\mathcal{C}$ does along the trunk (looking at the state of $\mathcal{B}_T$, which will be the same as that of $\mathcal{B}$ on the trunk) but at the branching point picks a $\mathcal{B}_{q,p}$ along the left-branch and a $\mathcal{B}_{q',p'}$ along the right branch such that $\mathcal{B}^{\mathcal{C}}$ could have spawned (p,q) down the left-branch and (p',q') down the right. The sole function of $\mathcal{C}'$ after branching is to check that the chosen $\mathcal{B}_{p,q}$ and $\mathcal{B}_{p',q'}$ accept the left and right branches respectively. We thus must have that $\mathcal{B}'^{\mathcal{C}'}$ accurately simulates $\mathcal{B}^{\mathcal{C}}$.

Since $\mathcal{B}'$ has at most one run on any input tree, we can complement $\mathcal{B}'^{\mathcal{C}'}$ by simply complementing $\mathcal{C}'$ in the usual way for regular tree automata.

□

This construction cannot in general be extended beyond $\mathbf{NTree}_2(\Sigma)$. If there were another branching point, the $\mathcal{B}_{p,q}$ would also need to branch, thwarting determinisation.

5.7. **Skeletons and Projection.** We turn our attention to closure under a notion of 'projection of branches'. For this we use a *skeleton selector* which is a tree automaton $\mathcal{S}$ whose state space can be partitioned into two sets P and S, where S 'selects' a subtree ('skeleton') of the input of $\mathcal{S}$.

Definition 5.16. A *skeleton selector* is a finite tree automaton $\mathcal{S} = \langle \Sigma, Q, q_0, \delta, F, P, S \rangle$ where Q is a finite set of control-states; $q_0 \in Q$ is the initial state; $\delta : \Sigma \times Q \longrightarrow 2^{\bigcup_{i=1}^{k} Q^i}$ is a transition function, $F \subseteq Q$ is a set of final states, and $P, S \subseteq Q$ are such that $Q = P \cup S$ and $P \cap S = \emptyset$. We also require that if $q \in S$ and $\vec{r} \in \delta(a,q)$, then *at least one* element in $\vec{r}$ must be in S. Moreover if $q \in P$ and $\vec{r} \in \delta(a,q)$, then *every* element in $\vec{r}$ must be in P.

An $\mathcal{S}$-skeleton of $T^{\curvearrowleft E} \in \mathbf{NTree}(\Sigma)$ is a subtree consisting of precisely the nodes assigned a state in S by an accepting run of $\mathcal{S}$. We denote the set of $\mathcal{S}$-skeletons by $\mathcal{S}(T^{\curvearrowleft E})$.

The $\mathcal{S}$-*skeleton language* and $\mathcal{S}^\infty$-*skeleton language* recognised by a nested-tree automaton $\mathcal{A}$ are respectively denoted and defined by:

$$\mathcal{L}^\pi_{\mathcal{S}}(\mathcal{A}) := \{\ S^{\curvearrowleft D}\ :\ S^{\curvearrowleft D} \in \mathcal{S}(T^{\curvearrowleft E}) \text{ for some } T^{\curvearrowleft E} \in \mathcal{L}(\mathcal{A})\ \}$$
$$\mathcal{L}^\infty_{\mathcal{S}}(\mathcal{A}) := \{\ S^{\curvearrowleft D}\ :\ S^{\curvearrowleft D} \in \mathcal{S}(T^{\curvearrowleft E}) \text{ for infinitely many } T^{\curvearrowleft E} \in \mathcal{L}(\mathcal{A})\ \}$$

So a skeleton belongs to $\mathcal{L}^\pi_{\mathcal{S}}(\mathcal{A})$ if there is some tree recognised by $\mathcal{A}$ with that skeleton. A skeleton belongs to $\mathcal{L}^\infty_{\mathcal{S}}(\mathcal{A})$ if there are *infinitely many* trees recognised by $\mathcal{A}$ that all have that same skeleton.

Very often we will be interested in skeleton selectors $\mathcal{S}$ that choose precisely one subtree—that is such that $\mathcal{S}(T^{\curvearrowleft E})$ is a singleton for every $T^{\curvearrowleft E}$. An important example is the *exoskeleton* selector that chooses precisely the $\prec$-greatest and $\prec$-least branches of a tree. For trees with branching degree bounded by d this can be defined by a deterministic tree automaton with $S = \{q_0, l, r\}$, $P = \{p\}$ where q_0 is the initial state and the transition function is given by:

$$\delta(\_, p) := \bigcup_{i=1}^{d} \{p^i\} \qquad \delta(\_, q_0) := \{q_0, (l,r), (l,p,r), (l,p,p,r), \ldots, (l, \underbrace{p,p,\ldots,p}_{d-2 \text{ times}}, r)\}$$
$$\delta(\_, l) := \{l, (l,p), (l,p,p), \ldots, (l, \underbrace{p,p,\ldots,p}_{d-1 \text{ times}})\} \qquad \delta(\_, r) := \{r, (p,r), (p,p,r), \ldots, (\underbrace{p,p,\ldots,p}_{d-1 \text{ times}}, r)\}$$

where all states are accepting states.

These skeleton languages provide a form of projection under which our three automaton variants are closed.

Lemma 5.17. *Let $\mathcal{S}$ be a skeleton-selector and let $\mathcal{A}$ be a nested-tree (*resp. *path-nested) automaton. There exists nested-tree (*resp. *path-nested) automata $\mathcal{S}(\mathcal{A})$ and $\mathcal{L}^\infty_{\mathcal{S}}(\mathcal{A})$ with:*

$$\mathcal{L}(\mathcal{S}(\mathcal{A})) = \mathcal{L}^\pi_{\mathcal{S}}(\mathcal{A}) \quad \textit{and} \quad \mathcal{L}(\mathcal{S}^\infty(\mathcal{A})) = \mathcal{L}^\infty_{\mathcal{S}}(\mathcal{A})$$

If $\mathcal{S}$ selects the spine of every tree (amongst possibly other branches) then the above also holds for spine nested automata. If $\mathcal{S}$ selects both the $\prec$-greatest and $\prec$-least branches (amongst possibly other branches), then it also holds for trunk nested automata.

Proof. We first exhibit $\mathcal{S}(\mathcal{A})$ recognising the required language. Let us assume that $\mathcal{A}$ is of the form $\mathcal{B}^{\mathcal{C}^-}$ (if $\mathcal{A}$ is a generic nested-tree automaton then it can be represented in this form by ignoring the $\mathcal{C}$ component). We begin by replacing $\mathcal{C}^-$ with the product of $\mathcal{S}$ and $\mathcal{C}^-$, which we call $\mathcal{C}$. For $(p,q) \in Q_{\mathcal{B}} \times Q_{\mathcal{C}}$ we define the set:

$$\mathit{Reach}(p,q) := \{\ S \subseteq Q_{\mathcal{B}} \times Q_{\mathcal{C}}\ :\ \exists T^{\curvearrowleft E}.\ \text{there is a run-tree of } \mathcal{B}^{\mathcal{C}} \text{ on } T^{\curvearrowleft E} \text{ with root labelled } (p,q) \text{ and leaves labelled by pairs of states in } S\}$$

This set is computable. Consider $(p,q) \in Q_{\mathcal{B}} \times Q_{\mathcal{C}}$ and $S \subseteq Q_{\mathcal{B}} \times Q_{\mathcal{C}}$. We can decide whether $S \in \mathit{Reach}(p,q)$ by simply checking for emptiness the nested-tree automaton formed from $\mathcal{B}^{\mathcal{C}}$ by changing the initial state to (p,q) and setting the final states to S.

We also define a set of control-states $Q^+_{\mathcal{B}} := Q_{\mathcal{B}} \times 2^{Q_{\mathcal{B}} \times Q_{\mathcal{C}}}$. The automaton $\mathcal{B}^+$ endowed with this state set behaves in the same manner as $\mathcal{B}$, keeping track of additional information in the second component of its state. If it is in state (q,S) at a node u of the input tree $T^{\curvearrowleft E}$, this means that it is possible to replace the subtree of $T^{\curvearrowleft E}$ rooted at u with another subtree (with root at u) such that $\mathcal{B}^{\mathcal{C}}$ would be able to complete the run-tree from u to become an accepting run-tree starting in a state $(q',p) \in S$.

$\mathcal{B}^+$ is able to compute this set S on the fly in a manner that also respects any deterministic behaviour of the original $\mathcal{B}$. In its initial state $(q_{0\mathcal{B}}, S_0)$ we can just take $S_0 := \{(p,q) : \exists F \in \mathit{Reach}(p,q)$ containing only accepting states$\}$. So long as neither the target nor source of a link is traversed the set S remains the same and $\mathcal{B}^+$ need not update it. This is because the set of ancestors of the current node that are targets with unmatched sources would not have changed. When transitioning from a node that is a source of a link u to its children, the set of ancestor nodes that are targets with unmatched sources returns to exactly what it was at the target $E(u)$ of u. Thus the transition from the source of a link to its children can simply restore S to what it was at $E(u)$.

So we only need to consider the case when u is the target of a pointer—*i.e.* $u \in \mathbf{img}(E)$. In this case the children of u will have an additional unmatched target amongst its ancestors in comparison to u. We define the following operator on S assuming the simulated $\mathcal{B}$ is in control-state p at u:

$$\begin{aligned}\mathit{AddTarget}(p,S) :=& \{\ (p',q) \in Q_{\mathcal{B}} \times Q_{\mathcal{C}}\ :\ \exists F \in \mathit{Reach}(p',q) \text{ s.t. } \forall (r,s) \in F.\\ & \exists a \in \Sigma.\ \exists k \in \mathbb{N}.\ \exists((r_1,s_1),\ldots,(r_k,s_k)) \in \delta_{\oplus\mathcal{B}^{\mathcal{C}}}(a,p,(r,s))\\ & \text{s.t. } (r_i,s_i) \in S \text{ for each } 1 \leq i \leq k\ \}\end{aligned}$$

So if the simulated control-state of $\mathcal{B}$ is p, the S sent to the children of u is $\mathit{AddTarget}(p,S)$. In summary the transition functions of $\mathcal{B}^+$ can be described thus:

$$\begin{aligned}\delta_{\ominus\mathcal{B}^+}(a,(q,S)) &:= \{\ ((q_1,\mathit{AddTarget}(q,S)),(q_2,\mathit{AddTarget}(q,S)),\\ &\quad \ldots,(q_k,\mathit{AddTarget}(q,S)))\ :\ (q_1,q_2,\ldots,q_k) \in \delta_{\ominus\mathcal{B}}(a,q)\ \}\\ \delta_{\mathcal{B}^+}(a,(q,S)) &:= \{\ ((q_1,S),(q_2,S),\ldots,(q_k,S))\ :\ (q_1,q_2,\ldots,q_k) \in \delta_{\mathcal{B}}(a,q)\ \}\\ \delta_{\oplus\mathcal{B}^+}(a,(p,R),(q,S)) &:= \{\ ((q_1,R),(q_2,R),\ldots,(q_k,R))\ :\ (q_1,q_2,\ldots,q_k) \in \delta_{\oplus\mathcal{B}}(a,p,q)\ \}\end{aligned}$$

and $\mathcal{B}^+$ will maintain the invariant that when in control-state (q,S) at node u there exists a subtree that could replace the subtree rooted at u such that $\mathcal{B}^{\mathcal{C}}$ could continue with a run from u in this replacement tree from a control-state in S and finish at every leaf of the replacement subtree in accepting states.

We can now modify the automaton $\mathcal{B}^+$ to form $\mathcal{B}^{++}$ that actually implements deletion. To preserve path nestedness/spine nestedness/trunk nestedness we do not give $\mathcal{B}^+$ the power to directly consider the skeleton selector as this may introduce undesirable non-determinism.

The automaton $\mathcal{B}^{++}$ makes a transition to k children $u_1,u_2,\ldots,u_k$ in control states $(q_1,q_2,\ldots,q_k)$ where $\mathcal{B}^+$ could make a transition to $k' \geq k$ children arranged as $\vec{v_1}u_1\vec{v_2}u_2\ldots\vec{v_k}u_k\vec{u_k}$ in control states $\vec{p_1},q_1,\vec{p_2},q_2,\ldots,\vec{p_k}q_k\vec{p_{k+1}}$. No constraints are placed on the choice of $u_1,\ldots,u_k$ for generic nested-tree automata or for path-nested automata. Observe that if $\mathcal{B}$ is path-nested, $\mathcal{B}^+$ will also be path-nested and because path-nestedness is immune to influence from behaviour at siblings. If $\mathcal{B}$ is spine-nested or trunk-nested, however, then we impose some additional constraints:

- If $\mathcal{B}$ is spine nested and $\mathcal{S}$ selects the spine of *every* tree, then we insist that $|\vec{p_{k+1}}| = 0$—*i.e.* $\mathcal{B}^{++}$ always assumes that the spine of the tree being read is also the spine of the tree on which a run of $\mathcal{B}^+$ is being simulated. Note that this means that $\mathcal{B}^{++}$ will also be spine nested. It also will never violate the constraints imposed by the skeleton selector.
- If $\mathcal{B}$ is trunk nested and $\mathcal{S}$ selects both the leftmost and rightmost branches of *every* tree, then we insist that $|\vec{p_1}| = |\vec{p_{k+1}}| = 0$. The assumption about $\mathcal{S}$ means that this will never be too strong a constraint to prohibit selecting the branches that $\mathcal{S}$ would

select. Moreover, this constraint also means that the trunk of both the original tree and the skeleton will be shared and so $\mathcal{B}^{++}$ will also be a trunk nested automaton.

For technical reasons we additionally extend the state space of $\mathcal{B}^{++}$ to include a record of the $\mathcal{B}$ control-state at $E(u)$ on the children of a node $u \in \mathbf{dom}(E)$. (We make the record on the children of u rather than u itself so as to avoid introducing any non-determinism as a result of this record).

We now define a finite tree automaton $\mathcal{C}^+$ that has access to the control-states of $\mathcal{B}^{++}$. $\mathcal{C}^+$ bases its behaviour on a simulation of $\mathcal{C}$ (recall that $\mathcal{C}$ in turn includes a simulation of $\mathcal{S}$). When it makes a transition to k children $u_1, u_2, \ldots, u_k$, it guesses pairs of control states in $Q_\mathcal{B} \times Q_\mathcal{C}$ as a vector $\vec{r_1}(p_1, q_1)\vec{r_2}(p_2, q_2)\vec{r_3} \cdots \vec{r_k}(p_k, q_k)\vec{r_{k+1}}$ such that $\mathcal{B}^\mathcal{C}$ could make a transition to children $\vec{v_1}u_1\vec{v_2}u_2 \ldots \vec{v_k}u_k\vec{u_k}$ in these control-states. It verifies the legitimacy of this guess when reading the state of $\mathcal{B}^{++}$ at the children, which contains a record of the control-state at the target of a pointer should the ability of $\mathcal{B}^\mathcal{C}$ to perform the guessed transition depend on this. It also checks (via its simulation of $\mathcal{S}$) that precisely the nodes $u_1, u_2, \ldots, u_k$ would be selected out of the postulated vector $\vec{v_1}u_1\vec{v_2}u_2 \ldots \vec{v_k}u_k\vec{u_k}$. A final consistency check ensures that the guess has the property that the control-state of $\mathcal{B}$ simulated by $\mathcal{B}^{++}$ at the child u_i is p_i.

$\mathcal{C}^+$ should also ensure that the pairs in the $\vec{r_i}$ can all yield accepting run trees. This occurs just in case any one of the children u_j could be replaced with a subtree from which $\mathcal{B}^\mathcal{C}$ would have an accepting run tree were it to start at the root of the new subtree in a state specified by such a pair. This is precisely the information contained in the set S from $\mathcal{B}^+$ in the control state of $\mathcal{B}^{++}$ at children—$\mathcal{C}^+$ just needs to check that the pairs in $\vec{r_i}$ all belong to the set S associated with the children (which will be the same for each child).

Note that we do not need to adjust the accepting states of $\mathcal{C}^+$—we can take them to be those that simulate accepting states of $\mathcal{C}$. The transition function of $\mathcal{C}^+$, as described above, will ensure that only $\mathcal{S}$ states that select a branch are propagated down the actual tree asserted to be a skeleton.

We can thus take $\mathcal{S}(\mathcal{A}) := \mathcal{B}^{++^{\mathcal{C}^+}}$.

Now let us turn our attention to $\mathcal{S}^\infty(\mathcal{A})$. In addition to the set $\mathit{Reach}(p, q)$ we need a set representing the fact that an infinite number of trees can be found giving a run with leaves ending in the given set of states:

$$\begin{aligned}\mathit{Inf}(p, q) \ := \{\ S \subseteq Q_\mathcal{B} \times Q_\mathcal{C} \ :\ \exists^\infty T^{\curvearrowleft E}.\ &\text{there is a run-tree of } \mathcal{B}^\mathcal{C} \text{ on } T^{\curvearrowleft E} \text{ with} \\ &\text{root labelled } (p, q) \text{ and leaves labelled by pairs of states in } S\}\end{aligned}$$

Again this set is computable. Consider $(p, q) \in Q_\mathcal{B} \times Q_\mathcal{C}$ and $S \subseteq Q_\mathcal{B} \times Q_\mathcal{C}$. Again we form an automaton from $\mathcal{B}^\mathcal{C}$ by changing the initial state to (p, q) and the final states to S. We have $S \in \mathit{Inf}(p, q)$ just in case the language accepted by this modified automaton is infinite. If it is infinite it must contain a large member on which the Pumping Lemma may be applied. Conversely if it contains a large member we may apply the Pumping Lemma to produce an infinite number of members. Thus we just need to test whether the language contains a large member—we can intersect the automaton with an automaton recognising large trees and test this for emptiness.

To form $\mathcal{S}^\infty(\mathcal{A})$ we adapt $\mathcal{B}^+$ to form $\mathcal{B}^\infty$. This automaton has state space $Q_{\mathcal{B}^\infty} := Q_\mathcal{B} \times 2^{Q_\mathcal{B} \times Q_\mathcal{C}} \times 2^{Q_\mathcal{B} \times Q_\mathcal{C}}$. A state (q, S, T) consists of q and S, which have exactly the same meaning as with $\mathcal{B}^+$ and are updated in exactly the same manner, together with T which is a set of $\mathcal{B}^\mathcal{C}$ from which the current pointer context would allow an accepting run on infinitely many extensions of the current tree.

The initial T_0 is given by $T_0 := \{(p,q) : \exists F \in \mathit{Inf}(p,q)$ containing only accepting states$\}$. At the target of a pointer T is updated to T' in a manner similar to the move from S to S' except that in this case we are also interested in the possibility of there being infinitely many possible extensions prior to the pop occurring. The analogue of the operator $\mathit{AddTarget}(p,S)$ for T is $\mathit{AddTarget}^\infty(p,S,T)$ defined as follows:

$$\begin{aligned}\mathit{AddTarget}^\infty(p,S,T) :=& \{\ (p',q) \in Q_{\mathcal{B}} \times Q_{\mathcal{C}}\ :\ \exists F \in \mathit{Inf}(p',q) \text{ s.t. } \forall (r,s) \in F.\\ &\exists a \in \Sigma.\ \exists k \in \mathbb{N}.\ \exists((r_1,s_1),\ldots,(r_k,s_k)) \in \delta_{\oplus \mathcal{B}^{\mathcal{C}}}(a,p,(r,s))\\ &\text{s.t. } (r_i,s_i) \in S \text{ for each } 1 \leq i \leq k\ \} \cup\\ &\{\ (p',q) \in Q_{\mathcal{B}} \times Q_{\mathcal{C}}\ :\ \exists F \in \mathit{Reach}(p',q) \text{ s.t. } \forall (r,s) \in F.\\ &\exists a \in \Sigma.\ \exists k \in \mathbb{N}.\ \exists((r_1,s_1),\ldots,(r_k,s_k)) \in \delta_{\oplus \mathcal{B}^{\mathcal{C}}}(a,p,(r,s))\\ &\text{s.t. } (r_i,s_i) \in S \text{ for each } 1 \leq i \leq k \text{ and } (r_i,s_i) \in T \text{ for some } 1 \leq i \leq k\ \}\end{aligned}$$

The first component of the union captures the cases when there are an infinite number of subtrees preceding the corresponding source of the target being added, and the second component of the union captures the cases when there are an infinite number of extensions of some subtree beyond the corresponding source. If there are an infinite number of extensions, then the extension must fall into one of these two categories. Note that we depend on S as well as T since not all branches need to admit an infinite number of possibilities for there to be an infinite number of possible trees—only one such branch need admit this.

Thus we implement $\mathcal{B}^\infty$ to update T in the same manner as S at source nodes and nodes that are neither a source nor a target and at a target node u we update T in a state (p,S,T) to $\mathit{AddTarget}^\infty(p,S,T)$ at the children of u. The automaton $\mathcal{B}^{\infty+}$ is then created from $\mathcal{B}^\infty$ in the same way as $\mathcal{B}^{++}$ was created from $\mathcal{B}^+$. The finite tree automaton $\mathcal{C}^\infty$ is defined in the same manner as $\mathcal{C}^+$ (but with respect to $\mathcal{B}^{\infty+}$ instead of $\mathcal{B}^{++}$) except that $\mathcal{C}^\infty$ *additionally* ensures that T can be used in place of S in *at least one* instance of deletion.

We then take $\mathcal{S}^\infty(\mathcal{B}^{\mathcal{C}}) := \mathcal{B}^{\infty+\mathcal{C}^\infty}$. This recognises precisely the trees that belong to the set of skeletons for an infinite number of trees. First suppose that $T^{\curvearrowleft E} \in \mathcal{L}(\mathcal{S}^\infty(\mathcal{B}^{\mathcal{C}}))$. Then it must be possible to decorate $T^{\curvearrowleft E}$ with states of $\mathcal{B}^{\mathcal{C}}$ such that this constitutes a subtree of some accepting run tree $\mathcal{R}$ of $\mathcal{B}^{\mathcal{C}}$ such $T^{\curvearrowleft E}$ is precisely a subtree selected by $\mathcal{S}$. Additionally there must exist at least one node of $\mathcal{R}$ shared with $T^{\curvearrowleft E}$ with a child in $\mathcal{R}$ not shared by $T^{\curvearrowleft E}$ that can be replaced by infinitely many possible alternative subtrees ending in accepting states. Thus $T^{\curvearrowleft E}$ is as required.

Conversely assume that $T^{\curvearrowleft E}$ belongs to the set of skeletons of an infinite number of trees. Since $T^{\curvearrowleft E}$ has finite size, this must mean that there exists at least one assignment σ of states of $\mathcal{B}^{\mathcal{C}}$ to the nodes of $T^{\curvearrowleft E}$ that forms a skeleton-selected subtree of infinitely many run-trees of $\mathcal{B}^{\mathcal{C}}$. Moreover there must exist at least one node u in $T^{\curvearrowleft E}$ from which there are infinitely many extensions to accepting run-trees consisting of nodes not belonging to $T^{\curvearrowleft E}$. Observe that the existence of an extension of a node u in $T^{\curvearrowleft E}$ with a run-tree depends exclusively on the assignment σ (to ancestors of u) and the states spawned down to the children of u—both the children in $T^{\curvearrowleft E}$ and those in the extended run tree. However, since all trees accepted by $\mathcal{B}^{\mathcal{C}}$ have nodes with bounded degree and since there are only finitely many control-states of $\mathcal{B}^{\mathcal{C}}$ it follows that there are only finitely many combinations of possible states on the children of u. Thus there must be at least one child of u that can be added as part of an extension of $T^{\curvearrowleft E}$ decorated with σ to a run tree of $\mathcal{B}^{\mathcal{C}}$ that can be assigned a control-state from which there are infinitely many extensions to accepting run trees. It follows that $\mathcal{S}^\infty(\mathcal{B}^{\mathcal{C}})$ will accept $T^{\curvearrowleft E}$, as required. □

6. Automaticity for Nested-Words and Trees

Traditional word and tree automatic structures (the classes of which we denote ***WAut*** and ***TAut***) present their relations via regular languages of 'convolutions' of words or trees. A convolution of a k-tuple $\vec{T}$ of words or trees is a single word or tree, taking as domain the union of the domains of $\vec{T}$, and labelling each node with k-tuples of alphabet symbols, indicating whether a particular element of $\vec{T}$ has that node in its domain, and if so its label.

We require an alternative notion of convolution, however, for tuples of nested-trees (and nested-words) due to the possibility of disagreement on pointers.

6.1. **Automaticity for Nested-Trees and Words.** The *pointer pattern* $pat_{T^{\frown E}}(u)$ of a node $u \in \mathbf{dom}(T)$ in a nested-tree $T^{\frown E}$ is a string over $\{t, n, s\}$ (target, neutral, source) such that if $u_1, \ldots, u_m$ is the path from the root to u, then $pat_{T^{\frown E}}(u) = p_1 \cdots p_m$ with:

$$p_i = \begin{cases} t & \text{if } u_i \in \mathbf{img}(E) \\ n & \text{if } u_i \notin \mathbf{img}(E) \cup \mathbf{dom}(E) \\ s & \text{if } u_i \in \mathbf{dom}(E) \end{cases}$$

Also consider the *trace* $trace_{T^{\frown E}}(u)$ of a node $u \in \mathbf{dom}(T)$ where T is Σ-labelled, which is an element of Σ^* indicating the sequence of Σ-labels of the nodes in the path from the root to u inclusive. Let us also abuse notation so that the domain of a tree may be a subset of $\{ (u, p, v) \in S_1^* \times S_2^* \times S_3^* : |u| = |p| = |v| \}$ for finite sets S_1, S_2 and S_3. Due to the length constraint we may also view an element (u, p, v) as being in $(S_1 \times S_2 \times S_3)^*$ and any 'prefix closed' set of such elements can naturally be viewed as the domain of a tree. Assuming an ordering on S_1, S_2 and S_3 we may also view such a tree as being ordered by taking the lexicographic ordering on $S_1 \times S_2 \times S_3$.

Let $T_1{}^{\frown E_1}, T_2{}^{\frown E_2}, \ldots, T_n{}^{\frown E_n} \in \mathbf{NTree}(\Sigma)$. We define their *convolution*:

$$\bigotimes \langle T_1{}^{\frown E_1}, T_2{}^{\frown E_2}, \ldots, T_n{}^{\frown E_n} \rangle$$

to be the $\Sigma \times 2^{[1..n]}$-labelled nested-tree $T^{\frown E}$ where:

- $\mathbf{dom}(T) = \{(u, p, v) \in ([1..k]^* \times \{t, n, s\}^* \times \Sigma^*) : \exists i. 1 \leq i \leq n. u \in \mathbf{dom}(T_i)$ and $pat_{T_i{}^{\frown E_i}}(u) = p$ and $trace_{T_i{}^{\frown E_i}}(u) = v\}$
- $\mathbf{dom}(E)$ consists of nodes in $\mathbf{dom}(T)$ of the form (u, ps, v) and we set $E(u, ps, v)$ to be the node $(u', p't, v')$ bearing the matching t to the s at u.
- We set $T(u, p, v'a) = (a, S)$ where:

$$i \in S \text{ iff } u \in \mathbf{dom}(T_i), pat_{T_i{}^{\frown E_i}}(u) = p \text{ and } trace_{T_i{}^{\frown E_i}}(u) = v'.a$$

We say a convolution *'splits'* when it branches due to a difference in pointers or node-labels between two of the $T_i{}^{\frown E_i}$. A node labelled (a, S) is *'associated with the ith element of the convolution'* if $i \in S$. The ordering on $\{t, n, s\}$ and Σ is unimportant; on $[1..k]$ we take the standard ordering, which is the dominant component of the lexicographic ordering.

We define the set $Convo_T^n(\Sigma)$ to be the set of n-ary convolutions over Σ (for trees) and $Convo_W^n(\Sigma)$ to be the set of n-ary convolutions over *nested-words* (rather than trees). $Convo_W^n(\Sigma)$ thus contains nested-trees with at most n leaves. It should be clear that convolutions are recognisable by a standard 'non-nested' tree automaton that can see when a node is a target or source of a pointer but does not need to exploit the pointer structure.

Let us fix a *'signature'* $\sigma = (R_1^{m_1}, \ldots, R_k^{m_k})$ (each $m_i \in \mathbb{N}$) for relational structures; this specifies the various m_i-ary predicates that the structure will define. In the case when

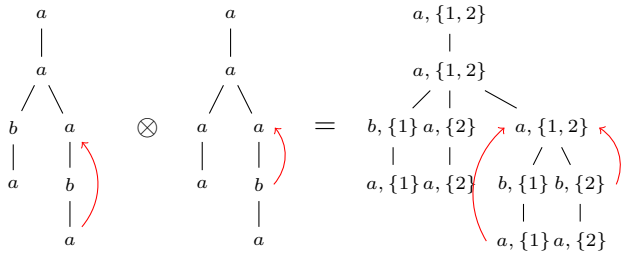


Figure 8: Convolution of Two Trees

$1 \leq m_i \leq 2$ for each i, we say that the structure is a *graph*; the binary relations are called *edges* and the unary predicates are called *colours*.

We describe six different ways of presenting σ-structures $\mathfrak{A} = \langle \boldsymbol{A}, \boldsymbol{R_1}, \ldots, \boldsymbol{R_k} \rangle$ using nested-tree automata. In each case $\boldsymbol{A} \subseteq \mathbf{NTree}(\Sigma)$ (sometimes $\mathbf{NWord}(\Sigma)$) for some finite alphabet Σ and $\boldsymbol{R_i}$ is encoded as a subset of $Convo^n_T(\Sigma)$ (sometimes $Convo^n_W(\Sigma)$) such that $\bigotimes \langle\, T_1{}^{\curvearrowleft E_1}, \ldots, T_k{}^{\curvearrowleft E_k} \,\rangle$ belongs to the encoding iff $(\, T_1{}^{\curvearrowleft E_1}, \ldots, T_k{}^{\curvearrowleft E_k} \,) \in \boldsymbol{R_i}$. If there is an automaton $\mathcal{A}$ recognising $\boldsymbol{A}$ and automata $\mathcal{R}_i = \mathcal{B}_i{}^{\mathcal{C}_i}$ recognising each of the encodings of $\boldsymbol{R_i}$ such that $\mathcal{B}_i$ only depends on the Σ-label of nodes in the convolution (*i.e.* the transition function of $\mathcal{B}_i$ only considers the Σ of labels in $\Sigma \times 2^{[1..n]}$), then we say that $\mathfrak{A}$ is:

- *isophilic* if $\mathcal{A}$ is a nested-*word* automaton (so the domain is a set of nested-*words*) and each of the $\mathcal{R}_i$ is a pathwise-nested automaton. This class is denoted by $\boldsymbol{Iso_2}$.
- *dendrisophilic* if the structure is a *graph*, $\mathcal{A}$ is a nested-word automaton and each of the $\mathcal{R}_i$ is a spine nested automaton. This class is denoted by $\boldsymbol{dIso_2}$.
- *symmetric-dendrisophilic* if the structure is a *graph*, $\mathcal{A}$ is a nested-word automaton and each of the $\mathcal{R}_i$ is a trunk nested automaton. This class is denoted by $\boldsymbol{sIso_2}$.
- *nondisophilic* if $\mathcal{A}$ is a nested-word automaton and each $\mathcal{R}_i$ are nested-tree automata. This class is called $\boldsymbol{nIso_2}$.
- *tree-isophilic* if all of the automata are pathwise-nested automata (the domain may consist of nested-trees, not just words). This class is denoted by $\boldsymbol{Iso_3}$.
- *tree-nondisophilic* if all of the automata are nested-tree automata. This class is denoted by $\boldsymbol{nIso_3}$.

The graphs belonging to the first two enjoy elegant characterisations in the form of prefix rewrite systems and correspond precisely to the transition graphs of second-order (collapsible) automata. We suggest that tree-isophilic structures are a natural third member of the progression from word to tree automatic structures. The graphs therein also subsume the third level of the Caucal hierarchy (although we lack a precise characterisation as with the second level). Tree-nondisophilic structures seem to be less interesting, but we introduce them as they provide the tool by which we get our decidability result for $\boldsymbol{FO}$ on a subclass of order-3 collapsible pushdown graphs.

It is also worth having the name *flat-isophilic* to refer to the special case of isophilic structures where the words are non-nested. This class of structures is denoted by $\boldsymbol{Iso_1}$. These correspond to standard regular prefix rewrite systems but do *not* capture all word automatic structures; remember that our definition of 'convolution' is different to normal.

Let us say that a class C of structures is *closed under* $\boldsymbol{FO^\infty}$*-definability* if for every $\mathfrak{A} \in C$ and $\boldsymbol{FO^\infty}$-formula $\phi(x_1, \ldots, x_m)$, adding the relation defined by ϕ to $\mathfrak{A}$ also results in a structure belonging to C.

Theorem 6.1. ***Iso*$_\mathbf{2}$**, ***nIso*$_\mathbf{2}$** *and* ***Iso*$_\mathbf{3}$** *are closed under* $\boldsymbol{FO^\infty}$ *definability. When we restrict consideration to structures with domains contained in* $\mathbf{NTree}_k(\Sigma)$ *for some fixed* $k \in \mathbb{N}$*, tree-nondisophilic structures are also closed under* $\boldsymbol{FO^\infty}$ *definability.*

Proof. Consider any such structure $\mathfrak{A}$ and an $\boldsymbol{FO^\infty}$ formula $\phi(x_1, \ldots, x_m)$. Argue by induction on the structure of ϕ, constructing an appropriate automaton recognising the encoding of a relation defined by each subformula. Boolean operations are covered by Lemma 5.14.

The induction cases of $\exists$ and $\exists^\infty$ are covered by Lemma 5.17. We use a skeleton selector selecting every node in the convolution except for those associated exclusively with component of the relation being quantified. We then project the labels of the convolution tree—in the isophilic case this will retain the determinism of the nested-word automata since originally their transitions depend only on the Σ-component of labels in the convolution and the construction in the proof of Lemma 5.17 does not change this. □

Since (symmetric-)dendrisophilic structures are just a special case of nondisophilic structures, emptiness testing together with Theorem 6.1 gives us:

Corollary 6.2. ***FO*** *and* $\boldsymbol{FO^\infty}$ *on isophilic, (symmetric-)dendrisophilic, nondisophilic and tree-isophilic structures are decidable.*

6.2. **Handling Tree-nondisophilicity and First-Order Logic.** We run into difficulty when it comes to tree-nondisophilic structures as we have no Boolean closure in general. Instead we need to restrict ourselves to a subclass of these structures where relations can be determined by examining only the right-most k branches of a convolution for some fixed k. This enables us to appeal to the closure properties of nested-tree automata that act only on trees with a bounded number of branches. The inherent locality of first-order logic encompassed by Gaifman's Locality Theorem [15] then allows first-order decidability.

6.2.1. *Gaifman's Locality Theorem.* Let us fix some standard terminology relating to Gaifman Locality from [15].

Definition 6.3. Let $\mathfrak{A} = \langle \boldsymbol{A}, \boldsymbol{R_1}, \ldots, \boldsymbol{R_m} \rangle$ be a relational structure. The *Gaifman graph* $\mathbf{Gaif}(\mathfrak{A})$ of $\mathfrak{A}$ is the graph with domain $\boldsymbol{A}$ and an (undirected) edge between $a, b \in \boldsymbol{A}$ whenever there exists $(c_1, c_2, \ldots, c_l) \in R_i$ for some $1 \leq i \leq m$ such that $a, b \in \{c_1, c_2, \ldots, c_l\}$.

Given $a \in \boldsymbol{A}$ the *local ball of radius* $r \in \mathbb{N}$ *about* a is the subset of $\boldsymbol{A}$:

$$B_r(a) := \{\ b \in \boldsymbol{A}\ :\ \text{there exists a path of at most length } r \text{ in } \mathbf{Gaif}(\mathfrak{A}) \text{ from } a \text{ to } b\ \}$$

Noting that edges in the Gaifman graph are definable by the same formula in all relational structures over a given signature and further noting that we can easily construct a formula $d_r(x, y)$ for each $r \in \mathbb{N}$ asserting that the shortest path between x and y in the Gaifman graph of the structure is of length at least r, the following definition makes sense:

Definition 6.4. An *r-local formula* $\phi(x, \vec{y})$ is one with at least one free variable x whose quantifiers are restricted to $B_r(x)$. A *basic local sentence* is a sentence of the form:

$$\exists x_1.\exists x_2.\cdots\exists x_k.\left(\bigwedge_{i=1}^{k}\phi_i(x_i)\ \wedge\ \bigwedge_{1\leq i<j\leq k} d_r(x_i,x_j)\right)$$

for some $r \in \mathbb{N}$ where the $\phi_i(x_i)$ are all r-local sentences.

Gaifman's Locality Theorem states the following:

Theorem 6.5 (Gaifman's Locality Theorem [15])**.** *Every first-order sentence is (effectively) equivalent to a Boolean combination of basic local sentences.*

6.2.2. *Locality In Tree-Nondisophilic Structures.*

Definition 6.6. Let us say that a set of n-ary convolutions R is *k-compact* for $k \in \mathbb{N}$ iff there exists a nested-tree automaton $\mathcal{A}$ (acting on trees with at most k branches) where:

$$C^{\frown E} \in R \quad \text{iff} \quad \mathcal{S}_k(C^{\frown E}) \in \mathcal{L}(\mathcal{A}) \text{ and each node of } C^{\frown E} \text{ not selected by } \mathcal{S}_k$$
$$\text{is associated with all of the } n \text{ trees yielding } C^{\frown E}$$

where $\mathcal{S}_k$ is the skeleton selector selecting the k right-most branches of a tree (thereby selecting a unique tree so that $\mathcal{S}_k(C^{\frown E})$ is a singleton set and so we may treat it as its element). We just say that a relation is *compact* if it is *k-compact* for some k.

In order to use this definition in the context of relational structures, it will be helpful to separate the two concerns of ensuring membership of the domain and determining whether elements are related. Thus for the purposes of this section we will view relations on a domain as being n-tuples of arbitrary nested-trees, not necessarily restricted to those nested-trees belonging to the domain. The following definition captures the place where we *do* want to be able to consider membership of the domain as well (the second condition):

Definition 6.7. We say that a tree nondisophilic structure $\mathfrak{A} = \langle \boldsymbol{A}, \boldsymbol{R_1}, \ldots, \boldsymbol{R_m}\rangle$ is *compact* if both of the following conditions are met:

- Every relation $\boldsymbol{R_i}$ is compact for $1 \leq i \leq m$.
- For every $r, k \in \mathbb{N}$ there is a compact relation $D^k_r \subseteq \mathbf{NTree}(\Sigma)^{k+1}$ such that:

$$\{\ (a, b_1, b_2, \ldots, b_k)\ :\ b_1, b_2, \ldots, b_k \in B_r(a)\ \cap\ \boldsymbol{A}, a \in \boldsymbol{A}\ \} \subseteq D^k_r$$

  and for every $a \in \boldsymbol{A}$, $(a, b_1, b_2, \ldots, b_k) \in D^k_r$ implies $b_1, b_2, \ldots, b_k \in B_r(a)\ \cap\ \boldsymbol{A}$.

We can use Gaifman's Locality Theorem together with the closure properties of nested-tree automata on trees with a bounded number of branches (Lemma 5.14) to get:

Theorem 6.8. ***FO*** *is decidable on compact tree-nondisophilic structures.*

Proof. Let $\mathfrak{A} = \langle \boldsymbol{A}, \boldsymbol{R_1}, \ldots, \boldsymbol{R_m}\rangle$ be a compact tree-nondisophilic structure with domain recognised by $\mathcal{A}$. We begin by showing that for every r-local formula $\phi(x, y_1, \ldots, y_k)$, there

exists $l \in \mathbb{N}$ and a nested-tree automaton $\mathcal{A}^B_\phi$ acting on trees with at most l branches with:

$$\begin{aligned}\left\{ C^{\frown E} \in \mathcal{L}(\mathcal{A}_l) \ : \ \mathcal{S}_l(C^{\frown E}) \in \mathcal{L}(\mathcal{A}^B_\phi) \text{ and } \hat{\pi}_1(C^{\frown E}) \in \mathcal{L}(\mathcal{A}) \right\} \\ = \left\{ \bigotimes \langle S^{\frown D}, {T_1}^{\frown E_1}, \dots, {T_k}^{\frown E_k} \rangle \ : \ \mathfrak{A} \vDash \phi(S^{\frown D}, {T_1}^{\frown E_1}, \dots, {T_k}^{\frown E_k}) \right. \\ \left. \text{and } {T_i}^{\frown E_i} \in B_r(S^{\frown D}) \text{ for each } 1 \leq i \leq k \right\}\end{aligned}$$

where $\mathcal{A}_l$ is an automaton that ensures that all nodes not belonging to the right-most l branches of the convolution (*i.e.* not selected by $\mathcal{S}_l$) are associated with every tree making up the convolution (*i.e.* that the convolution does not split on these branches), and also ensures that the labels on these nodes of the tree are consistent with it being an n-ary convolution. It ignores the right-most l branches. By $\hat{\pi}_1(C^{\frown E}) \in \mathcal{L}(\mathcal{A})$ we mean that the first component of the convolution is in $\mathcal{L}(\mathcal{A}) = \boldsymbol{A}$. We argue by induction on ϕ.

In the case when ϕ is atomic, it will be of the form $Ry_1 \cdots y_k$. Noting that equality $=$ is 0-compact (nothing needs to be checked to determine equality on the assumption that corresponding branches are the same in each tree being compared) and from the assumption that $\mathfrak{A}$ is compact we get that the relation R is l'-compact for some l'. We also have it that the relation D^k_r is l''-compact for some l''; D^k_r captures the relation "$y_1, \dots y_k \in B_r(x)$ on the assumption that $x \in \boldsymbol{A}$". Note that we may view both of these relations as being $\max(l', l'')$-compact. If w.l.o.g. $l' < l''$, then we simply extend the automaton acting on the right-most l' branches of the convolution (recognising R) to act on the right-most l'' branches, asserting equality between the nodes not lying in the right-most l' branches. We can thus take $\mathcal{A}^B_\phi$ to be the intersection of these two nested-tree automata acting on the right-most $l := \max(l', l'')$ branches of the convolution.

For a conjunction $(\phi \wedge \psi)$ we can take $\mathcal{A}^B_{(\phi \wedge \psi)}$ the intersection of $\mathcal{A}^B_\phi$ and $\mathcal{A}^B_\psi$. Note that ϕ and ψ might contain variables not contained in the other. However the convolutions being recognised by $\mathcal{A}^B_\phi$ and $\mathcal{A}^B_\psi$ will both contain one variable in common, namely x, the centre of the local ball. So if $\mathcal{A}^B_\phi$ acts on the right-most l' branches of a convolution $C^{\frown E}$ and $\mathcal{A}^B_\psi$ on the right-most l'' branches of a convolution ${C'}^{\frown E'}$, then everything outside the right-most l' branches of $C^{\frown E}$ must be shared by all elements, in particular x, and everything outside the right-most l'' branches of ${C'}^{\frown E'}$ must be shared by all elements, in particular x. Since x is shared in both convolutions, merging them must result in a convolution for which everything outside the right-most $l' + l''$ branches is shared by all elements in the convolution (including x). Thus the intersections and unions above can be recognised by automata acting on at most the right-most $l := l' + l''$ branches of a convolution.

For negation we must obtain an automaton $\mathcal{A}^B_{\neg\phi}$ from $\mathcal{A}^B_\phi$. We can get this by complementing $\mathcal{A}^B_\phi$ (possible due to Lemma 5.12) and intersecting with the automaton for D^k_r in the same manner as for the atomic case. Because we are restricting the elements of the convolution to those belonging to $B_r(x)$ this is sound as membership of $B_r(x)$ requires being identical to x at nodes not read by the automaton for D^k_r.

Now we come to existential quantification: $\exists y.\phi(x, y, y_1, \dots, y_k)$. Since $y \not\equiv x$ (because the variable defining the centre of the ball is always kept free) we can thus obtain the requisite automata by projecting away nodes belonging only to y from $\mathcal{A}_\phi$ using Lemma 5.17, deleting references to y from node-labels, and by replacing D^{k+1}_r with D^k_r.

It follows that we can construct a nested-tree automaton $\mathcal{A}^B_\phi$ such that $\mathcal{L}(\mathcal{A}^B_\phi) \cap \boldsymbol{A}$ is the set of nodes defined by an r-local formula $\phi(x)$ with a single free variable representing

the centre of its ball. Additionally note that the relation $d_r(x, y)$ holds iff $y \notin B_{r-1}(x)$. But by assumption membership of local balls 'modulo $x \in \boldsymbol{A}$' is compact as witnessed by the automaton $\mathcal{A}_{B_{r-1}}$ for D^1_{r-1} reading only the right-most l branches for some l. We thus have $y \notin B_{r-1}(x)$ if either the right-most l branches are rejected by $\mathcal{A}_{B_{r-1}}$ or there is a node differing between y and x outside the right-most l branches. The latter can easily be detected by a nested-tree automaton and the former can be detected since $\mathcal{A}_{B_{r-1}}$ can be complemented due to the fact it acts on a bounded number of branches. We can thus construct a nested-tree automaton recognising the relation $d_r(x, y)$.

Since basic local sentences can be formed from such formulae without the need for negation (*i.e.* without the need of complementing automata) and intersection and can be performed on all nested-tree automata, and since moreover we have a nested-tree automaton recognising $\boldsymbol{A}$, which allows us to restrict the range of the centres of our local balls to the domain, it then follows that for any given basic local sentence ϕ we can construct a nested-tree automaton that recognises a non-empty language iff $\mathfrak{A} \models \phi$. Thus by Gaifman's Locality Theorem (re-stated here as Theorem 6.5) we get decidability for $\boldsymbol{FO}$. □

6.3. **Isophilic and Dendrisophilic Chains.** For this section we restrict our attention to isophilic and (symmetric-)dendrisophilic *graphs*; so only (nested-)words make up the domains and all relations are either binary or unary. The fact that convolutions must branch as soon as components begin to differ means that an arbitrary number of convolutions can be 'glued together' whilst retaining a finite alphabet. This allows us to represent *paths* in isophilic and dendrisophilic graphs as trees denoting *'chains'* of words.

A *path* in a graph $\mathfrak{A}$ is a sequence of nodes $u_1, \ldots, u_m$ such that for $1 \leq i < m$ there is an edge from u_i to u_{i+1}. We say that $u\boldsymbol{r}u'$ if u' is *reachable* from u—that is there is a path from u to u'. We say that $u\boldsymbol{r}^{\infty}u'$ if there exist *infinitely many* distinct paths from u to u'—note that if there is a cycle in the graph on some path between u and u' this automatically implies $u\boldsymbol{r}^{\infty}u'$ (although not conversely in an infinite graph).

Definition 6.9. For each graph $\mathfrak{A}$ let $\mathcal{R}_{\mathfrak{A}}$ be the set of its binary relations. A $\mathcal{Q}$-*decorated structure* $\mathfrak{A}_{\mathcal{Q}}$ adds a map $\mathcal{Q} : \mathcal{R}_{\mathfrak{A}} \longrightarrow Q \times Q$ for some finite set Q. This allows for a finely controlled notion of path. A path $u_1, \ldots, u_m$ in $\mathfrak{A}$ is deemed $\mathcal{Q}$-*compatible* if there is a function $f : [1..m] \longrightarrow Q$ and relations $R_1, \ldots, R_{m-1}$ such that for every $1 \leq i < m$ we have $u_i R_i u_{i+1}$ and $\mathcal{Q}(R_i) = (f(i), f(i+1))$.

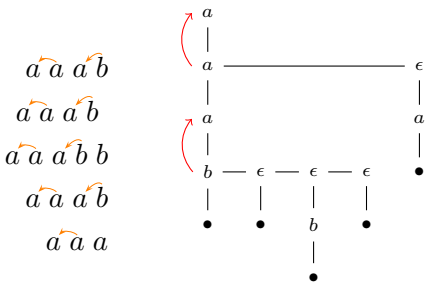


Figure 9: Isophilic Chain

We say that $u\boldsymbol{r}_{\boldsymbol{q},\boldsymbol{q}'}u'$ if there is a $\mathcal{Q}$-compatible path p beginning with u and ending in u' for which $f(1) = q$ and $f(|p|) = q'$ and $u\boldsymbol{r}^{\boldsymbol{\infty}}_{\boldsymbol{q},\boldsymbol{q}'}u'$ if there exist infinitely many such paths. Note $\boldsymbol{r}$ and $\boldsymbol{r}^{\boldsymbol{\infty}}$ arise when Q is a singleton.

Example 6.10. The motivating example is when the nested(-words) $s_1, s_2, \ldots, s_k$ represent the contents of a stack of a pushdown automaton and we take Q to be its set of control states. The relations would then represent stack operations and a suitable $\mathcal{Q}$ would associate control-states to stack operations. If $s_1, s_2, \ldots, s_k$ forms a $\mathcal{Q}$-compatible path, then this means that there exist control-states $q_1, q_2, \ldots, q_k \in Q$ such that $(q_1, s_1), (q_2, s_2), \ldots, (q_k, s_k)$ is a run of the automaton. Likewise $s\boldsymbol{r}_{\boldsymbol{q},\boldsymbol{q}'}s'$ would mean

that there is a run from (q,s) to (q',s') and $s\boldsymbol{r}^{\infty}_{\boldsymbol{q},\boldsymbol{q'}}s'$ would mean that there are infinitely many runs from (q,s) to (q',s').

6.4. **Branch Chains.** Let $\mathfrak{A}_{\mathcal{Q}}$ be a $\mathcal{Q}$-decorated (symmetric-)dendrisophilic *resp.* isophilic structure. Assume the nested-words in $\mathfrak{A}$'s domain use alphabet Σ. A *(symmetric-)dendrisophilic (*resp. *isophilic) chain* over $\mathfrak{A}_{\mathcal{Q}}$ is a $\Sigma \cup \{\epsilon, \bullet\}$-labelled nested-tree $C^{\curvearrowleft E}$ such that:

- It is a binary tree.
- Every node $u \in \mathbf{dom}(C)$ with $C(u) = \bullet$ satisfies $u \notin \mathbf{dom}(E)$ and $u \notin \mathbf{img}(E)$ and may have no children.
- If a node is not labelled $\bullet$, then it must have a child.
- Every node $u \in \mathbf{dom}(C)$ with $C(u) = \epsilon$ satisfies $u \notin \mathbf{dom}(E)$ and $u \notin \mathbf{img}(E)$ and cannot be the root.
- The *left* child *or only* child of a node may *not* be ϵ-labelled. If a node has two children, then the *right* child *must* be ϵ-labelled.
- Let $wth(C^{\curvearrowleft E})$ be the number of leaves of $C^{\curvearrowleft E}$. For $1 \leq i < wth(C^{\curvearrowleft E})$ let ${T_i}^{\curvearrowleft E_i}$ be the subtree consisting of just the i and $(i+1)$th branches of $C^{\curvearrowleft E}$. It should be the case that $\pi_\Sigma({T_i}^{\curvearrowleft E_i}) = \pi_1\left(\bigotimes\left\langle\, \pi_\Sigma(C^{\curvearrowleft E}_{\prec_i}), \pi_\Sigma(C^{\curvearrowleft E}_{\prec_{i+1}})\,\right\rangle\right)$.
- $\pi_\Sigma(C^{\curvearrowleft E}_{\prec_1}), \ldots, \pi_\Sigma(C^{\curvearrowleft E}_{\prec_{wth(C^{\curvearrowleft E})}})$ forms a $\mathcal{Q}$-compatible path in $\mathfrak{A}$.

where we write $\pi_\Sigma(T^{\curvearrowleft E})$ to denote the tree formed by deleting all nodes not labelled in Σ and connecting the parent of a deleted node to its closest Σ-labelled descendants.

Let us denote the space of (symmetric-)dendrisophilic (*resp.* isophilic) chains over $\mathfrak{A}_{\mathcal{Q}}$ by $\boldsymbol{Ch}(\mathfrak{A}_{\mathcal{Q}})$. We should emphasise that neighbouring branches in a chain can be viewed as forming a convolution (modulo ϵ-edges), which is important for the lemma below:

Lemma 6.11. *For any $\mathcal{Q}$-decorated symmetric-dendrisophilic, dendrisophilic or isophilic graph $\mathfrak{A}_{\mathcal{Q}}$ it is the case that:*

(1) There exists respectively a trunk nested, spine nested or path-nested automaton recognising $\boldsymbol{Ch}(\mathfrak{A})$.

(2) Let $u_1, \ldots, u_m$ in $\mathfrak{A}$ be a $\mathcal{Q}$-compatible path in $\mathfrak{A}$. Then there exists a unique *$C^{\curvearrowleft E} \in \boldsymbol{Ch}(\mathfrak{A})$ such that $wth(C^{\curvearrowleft E}) = m$ and $u_i = \pi_\Sigma(C^{\curvearrowleft E}_{\prec_i})$ for each $1 \leq i \leq m$. The converse also holds.*

Proof. Item (2) holds by construction. For (1) we begin with the most general case—when the graph is symmetric-dendrisophilic. Let $\mathcal{A}_{p,q} = {\mathcal{B}_{p,q}}^{\mathcal{C}_{p,q}}$ be a trunk nested automaton recognising the union of all relations decorated by $p, q \in Q$ (where $Q \times Q$ is the image of $\mathcal{Q}$). Recall that this means that the $\mathcal{B}_{p,q}$ are deterministic along the trunks of trees. The convolutions read by $\mathcal{A}_{p,q}$ are binary relations between nested-words, so the trees upon which they act have at most two branches. We call the component leaving the trunk to the left as the *left-branch* and the component leaving the trunk to the right as the *right-branch*.

We define an automaton $\mathcal{A} = \mathcal{B}^{\mathcal{C}}$ recognising $\boldsymbol{Ch}(\mathfrak{A})$. This will act on a trees with an arbitrary number of branches where adjacent branches should be related in a manner that would be accepted by one of the $\mathcal{A}_{p,q}$. It has control-states $Q_{\mathcal{B}}$ where:

$$Q_{\mathcal{B}} = \underbrace{\prod_{\in Q\times Q}^{(p,q)} Q_{\mathcal{B}_{p,q}}}_{\text{Trunk-Sim}} \times \underbrace{\prod_{\in Q\times Q}^{(p,q)} Q^{\perp}_{\mathcal{B}_{p,q}} \times \mathbb{B}}_{\text{Left-Branch Sim}} \times \underbrace{\prod_{\in Q\times Q}^{(p,q)} Q^{\perp}_{\mathcal{B}_{p,q}} \times \mathbb{B}}_{\text{Right-Branch Sim}} \times \Sigma^{\perp}$$

The set $Q_{\mathcal{B}_{p,q}}$ is the set of control-states of the automaton $\mathcal{B}_{p,q}$. For a set S we write $S^{\perp}$ to denote $S \uplus \{\perp\}$. We use the value $\perp$ when the value is 'irrelevant' or undefined.

The idea is that the *trunk sim* component will always simulate the behaviour that each $\mathcal{B}_{p,q}$ would exhibit when reading the trunk of a binary convolution. Since $\mathcal{A}_{p,q}$ is a trunk nested automaton, by definition this behaviour must be deterministic. This means that $\mathcal{B}_{p,q}$ is able to carry out this simulation despite a node in a chain potentially belonging to the trunk of an unbounded number of binary convolutions. The *left-branch sim* and the *right-branch sim* components will respectively simulate the behaviour of each $\mathcal{B}_{p,q}$ at the left or right branch of a binary convolution. Behaviour here may be non-deterministic, but this is not a problem as any given node in a chain will belong to the left-branch of at most one convolution and the right-branch of at most one convolution.

The Boolean flag $\mathbb{B}$ associated with the left and right branch simulations is used to keep track of whether any pointer encountered during the left/right simulation should be treated as a pointer to the trunk of the convolution or a pointer to the left/right branch of the convolution. A value $\mathbf{f}$ means that subsequent pointer sources should be simulated as pointing to a target in the trunk and a value $\mathbf{t}$ means that subsequent pointer sources should be treated as pointing to a target in the left/right branch.

The Σ component is just used to remember the most recent non-ϵ symbol visited. This is implicitly used when branching having skipped over ϵ nodes.

The automaton $\mathcal{B}$ begins in state $(q_{0\vec{p,q}}, \vec{\perp}, \mathbf{f}, \vec{\perp}, \mathbf{f}, \perp)$ where $q_{0\vec{p,q}}$ is the vector of initial states for each $\mathcal{B}_{p,q}$. It then transitions as follows:

- The *trunk sim* component is updated according to the transition functions of the $\mathcal{B}_{p,q}$ as they would act when a node has at most one child. If transitioning from a node that is the source of the pointer, the trunk component of the target is used as the target state. This results in it simulating the behaviour of the automata under the assumption that they never come across any branching. This update must be deterministic. Any nodes labelled ϵ in the chain are simply ignored by this component. The Σ component is set to the node label just read whenever this node-label belongs to Σ (*i.e.* is not ϵ).
- The automaton only allows nodes of the input tree to have at most two children. When such a node is reached, it simulates reaching the single point of branching in a binary convolution. Let $a \in \Sigma$ be the current node symbol if this belongs to Σ or the value in the Σ component of the control state if the current node symbol is ϵ. Let $s_{p,q}$ be the state simulated in the *trunk* component for $\mathcal{B}_{p,q}$. For each pair $(p,q) \in Q \times Q$ the automaton should non-deterministically pick a pair of states $(s^l_{p,q}, s^r_{p,q}) \in Q_{\mathcal{B}_{p,q}} \times Q_{\mathcal{B}_{p,q}}$ belonging to the set given by the transition function of $\mathcal{B}_{p,q}$ acting on an a-labelled node in state $s_{p,q}$ and appropriate for the current node's pointer status. If the current node is the source of a pointer, the state in the trunk component of the target is used as the target state. The *left sim* component of the *left child* should be set to $(\vec{s^l_{p,q}}, \mathbf{f})$ and the *right sim* component of the *right child* should be set to $(\vec{s^r_{p,q}}, \mathbf{f})$. The value $\mathbf{f}$ is correct for the Boolean flags since we have not yet passed any node that is the target of a pointer since beginning this fresh left/right branch simulation.
- The *left sim* and *right sim* components are updated non-deterministically according to the transition functions of the $\mathcal{B}_{p,q}$ when transitioning from a node with at most one child. Nodes labelled ϵ are ignored. The Boolean flag in each component is set

to $\mathbf{t}$ when transitioning from a node that is the target of a pointer. This is correct behaviour as a source node occurring subsequently must point to a node visited after the start of this left/right branch simulation. When transitioning from the source of a node, the Boolean flag is set to the value of the Boolean flag in the corresponding component at the target of the pointer.

- When the current node has two children, the *left sim* simulation is propagated down the *right* child and the *right sim* simulation is propagated down the *left* child. This is correct behaviour since the left component of one of the binary convolutions in the chain will be the right-most branch beginning at the left child of the branching point of the convolution and the right component will similarly be the left-most branch beginning at the right child of the branching point of the convolution. Note further that this does not conflict with the spwaning of fresh left and right-sims in the second item, which are spawned in the opposite directions.

Note that $\mathcal{C}_{p,q}$ has no obligation to be deterministic anywhere on its input tree. However, in contrast to $\mathcal{B}$ whose transitions can depend on states at non-local nodes in the tree, the locality of the dependencies for transitions of $\mathcal{C}_{p,q}$ allows us to make it deterministic along the trunk of a convolution (before it splits) using the power-set construction. This enables us to simulate each $\mathcal{C}_{p,q}$ in a similar manner as the simulation for each $\mathcal{B}_{p,q}$. However, we simulate only one of the $\mathcal{C}_{p,q}$ in the left and right branch simulations rather than all of them simultaneously as we did with $\mathcal{B}$. This is reflected in the fact that we have taken unions rather than products. This is important since it forces $\mathcal{C}$ to decide which $\mathcal{C}_{p,q}$ it is going to use at each convolution in the chain, which amounts to guessing the $\mathcal{Q}$-decoration that will be placed on the elements of the chain. Define $\mathcal{C}$ to have state space:

$$Q_{\mathcal{C}} := \underbrace{\prod_{\substack{(p,q)\\ \in Q\times Q}} 2^{Q_{\mathcal{C}_{p,q}}}}_{\text{trunk-sim}} \times \underbrace{\left(\bigcup_{\substack{(p,q)\\ \in Q\times Q}} Q_{\mathcal{C}_{(p,q)}}\right)^{\perp}}_{\text{left-branch-sim}} \times \underbrace{\left(\bigcup_{\substack{(p,q)\\ \in Q\times Q}} Q_{\mathcal{C}_{(p,q)}}\right)^{\perp}}_{\text{right-branch-sim}} \times \Sigma$$

The initial state of $\mathcal{Q}_{\mathcal{C}}$ is $(\vec{I_{p,q}}, \perp, \perp)$ (where $\vec{I_{p,q}}$ are the sets of initial states of each $Q_{\mathcal{C}_{p,q}}$). The trunk-sim component behaves the same on all branches of the chain, keeping track of all possible states that each $\mathcal{Q}_{p,q}$ could be in prior to it splitting.

The *left*-branch-sim always propagates along the *right* child or *only* child and the *right*-branch-sim always propagates along the *left* child or *only* child, simulating behaviour *after* the convolution has split. When the chain branches, some $(p, q) \in Q \times Q$ is chosen and a fresh left-branch-sim is propagated down the *left* child and a fresh right-branch-sim down the *right* child based on a permissible transition of $\mathcal{C}_{p,q}$ at the splitting point of a convolution from one of the states in the trunk-sim component.

The automaton $\mathcal{C}$ has access to the states of $\mathcal{B}$ to enable its simulation to depend on the simulated $\mathcal{B}_{p,q}$. It accepts if all of its left-branch-sim and right-branch-sim components simulate accepting states, with the additional condition that if the left-branch-sim at a leaf is simulating $\mathcal{C}_{p,q}$, then the right-branch-sim of that leaf must simulate $\mathcal{C}_{q,r}$ for some $r \in Q$.

So this gives the requisite trunk automaton $\mathcal{B}^{\mathcal{C}}$. If we are considering dendrisophilic graphs so that the $\mathcal{B}^{\mathcal{C}_{p,q}}_{p,q}$ are spine-nested automata, then we can make $\mathcal{B}^{\mathcal{C}}$ a spine-nested automaton as well by simply removing the right-sim component from the states of $\mathcal{B}$. We could rename the 'trunk sim' component the 'spine sim' component. The simulation in the

right-sim component of $\mathcal{C}$ will look at the state in the trunk-sim of $\mathcal{B}$ all of the time, which is correct by the definition of spine nested automata.

For isophilic graphs we can make $\mathcal{B}^{\mathcal{C}}$ path-nested by removing both the right-sim *and* left-sim components of $\mathcal{B}$. We could rename the 'trunk sim' component the 'path sim' component. Both the left and right-sim components of $\mathcal{C}$ refer to the path sim simulation of $\mathcal{B}$ all of the time, which is adequate for path-nested automata. □

We are now in a position to show that reachability is definable in dendrisophilic and isophilic graphs. It is unclear whether this proof can be extended to symmetric-dendrisophilic structures. (In [9] there is a bug in our proof claiming this to be so[1].) The proof relies on applying projection to a chain, but this does not necessarily result in a convolution as the projected chain might branch 'too early' to be a convolution. The automaton recognising convolutions in the reachability relation must therefore be able to simulate an early split. Spine-nested automata are able to do this 'retrospectively' by simulating the split having entered the branch along which it is allowed to behave non-deterministically. The same trick would not work for trunk-nested automata as such a 'retrospective simulation' would have to be synchronised in both the left and right branches.

Theorem 6.12. *Let $\mathfrak{A}$ be a dendrisophilic or isophilic graph. The graph obtained by adding the relations $\boldsymbol{r}$ and $\boldsymbol{r}^{\boldsymbol{\infty}}$ is also respectively dendrisophilic or isophilic. Where $\mathfrak{A}_{\mathcal{Q}}$ is $\mathcal{Q}$-decorated the same applies to $\boldsymbol{r}_{\boldsymbol{q},\boldsymbol{q'}}$ and $\boldsymbol{r}^{\boldsymbol{\infty}}_{\boldsymbol{q},\boldsymbol{q'}}$ for each $q, q' \in Q$.*

Proof. In order to get an automaton recognising $\boldsymbol{r}_{\boldsymbol{q},\boldsymbol{q'}}$, we first obtain an automaton recognising $\boldsymbol{Ch}(\mathfrak{A})$ using Lemma 6.11. We modify this to ensure that the simulation on the left-most binary convolution is of $\mathcal{C}_{q,r}$ for some $r \in Q$ and that the simulation on the right-most binary convolution is of $\mathcal{C}_{r',q'}$ for some r'. We then project onto the exoskeleton of the chain using Lemma 5.17. Call the resulting automaton $\mathcal{B}^{\mathcal{C}}$; this is either path-nested or spine-nested depending respectively on whether the structure is isophilic or dendrisophilic.

Note that any node labelled ϵ can be neither the source nor the target of the pointer. The automaton can thus easily be modified to remove ϵ-transitions from the members of its language. A convolution should split iff a difference in pointer or label occurs. Whilst the two-branched trees accepted by $\mathcal{B}^{\mathcal{C}}$ must split when such a difference occurs, the converse does not hold; an accepted tree might split when nodes have identical labels and pointers. We must therefore create another automaton $\mathcal{B}'^{\mathcal{C}'}$ that accepts only the binary convolutions of nested-words making up the two branches of some tree accepted by $\mathcal{B}^{\mathcal{C}}$. In particular $\mathcal{B}'^{\mathcal{C}'}$ must be able to simulate a premature split of $\mathcal{B}^{\mathcal{C}}$.

In the case of isophilic structures this is straightforward, since $\mathcal{B}$ would act deterministically along every branch independently, and $\mathcal{C}$ can be modified to form a $\mathcal{C}'$ prohibiting premature splitting and making a non-deterministic choice to start simulating a premature split of the original $\mathcal{B}^{\mathcal{C}}$. In the dendrisophilic case, we must be more careful since the point where the convolution splits involves $\mathcal{B}$ also making a non-deterministic choice, and this cannot be directly executed prematurely without violating spine-nestedness. Ideas borrowed from the determinisation of nested-word automata can deal with this. We thus create a spine-nested automaton $\mathcal{B}'$ with states belonging to the following set:

$$Q_{\mathcal{B}'} := \underbrace{2^{(Q_{\mathcal{B}}\times Q_{\mathcal{C}}\times\{trunk,rib,spine\})\times(Q_{\mathcal{B}}\times Q_{\mathcal{C}}\times\{trunk,rib,spine\})}}_{\text{trunk states}} \cup \underbrace{\Big((Q_{\mathcal{B}} \times Q_{\mathcal{C}} \times \{trunk, rib, spine\})^{\perp} \times Q_{\mathcal{B}}\Big)}_{\text{rib states}}$$

[1]We are grateful to Alexander Kartzow for pointing this out.

The 'trunk states' are used along the trunk of the convolution being read by $\mathcal{B}'$ (*i.e.* before the convolution splits). A triple $(q_{\mathcal{B}}, q_{\mathcal{C}}, trunk) \in Q_{\mathcal{B}} \times Q_{\mathcal{C}} \times \mathbb{B}$ asserts that $\mathcal{B}^{\mathcal{C}}$ could be in state $(q_{\mathcal{B}}, q_{\mathcal{C}})$ whilst reading the trunk of a tree (prior to it splitting, possibly prematurely). A triple $(q_{\mathcal{B}}, q_{\mathcal{C}}, rib)$ asserts that $\mathcal{B}^{\mathcal{C}}$ could be in state $(q_{\mathcal{B}}, q_{\mathcal{C}})$ whilst reading the *rib* (left branch after branching) of a tree that has already split prematurely. A triple $(q_{\mathcal{B}}, q_{\mathcal{C}}, spine)$ asserts that $\mathcal{B}^{\mathcal{C}}$ could be in state $(q_{\mathcal{B}}, q_{\mathcal{C}})$ whilst reading the *spine* of a tree when a premature split has already occurred (right branch after branching). A pair of such triples (t, t') asserts that if the assertion made by triple t were true at the most recent pointer-target whose source has not yet been seen, then t' would be true at the current node. We can program $\mathcal{B}'$ to maintain the invariant that its state S on the trunk is a maximal set of such pairs of triples making a correct assertion. This invariant can be *deterministically* maintained by the following transition functions on trunk-states (an idea similar to the determinisation of nested-word automata); to keep the description concise, we w.l.o.g. assume splits occur only at nodes neither the source nor target of a pointer:

$$\begin{aligned}
\delta_{\mathcal{B}'}(a, S) :=& \{\{ \ (t, (q'_{\mathcal{B}}, q'_{\mathcal{C}}, b')) \ : \ q_{\mathcal{B}'} \in \delta_{\mathcal{B}}(a, q_{\mathcal{B}}), q'_{\mathcal{C}} \in \delta_{\mathcal{C}}(a, q_{\mathcal{B}}, q_{\mathcal{C}}) \text{ for some } (t, (q_{\mathcal{B}}, q_{\mathcal{C}}, b')) \in S \ \} \\
& \cup \{ \ (t, (q'_{\mathcal{B}}, q'_{\mathcal{C}}, rib)), (t, (q''_{\mathcal{B}}, q''_{\mathcal{C}}, spine)) \ : \ (q_{\mathcal{B}'}, q_{\mathcal{B}''}) \in \delta_{\mathcal{B}}(a, q_{\mathcal{B}}), \\
& (q_{\mathcal{C}'}, q_{\mathcal{C}''}) \in \delta_{\mathcal{C}}(a, q_{\mathcal{B}}, q_{\mathcal{C}}) \text{ for some } (t, (q_{\mathcal{B}}, q_{\mathcal{C}}, trunk)) \in S \}\} \\
\delta_{\ominus\mathcal{B}'}(a, S) :=& \{\{ \ ((q_{\mathcal{B}}, q_{\mathcal{C}}, b'), (q'_{\mathcal{B}}, q'_{\mathcal{C}}, b')) \ : \ q_{\mathcal{B}'} \in \delta_{\ominus\mathcal{B}}(a, q_{\mathcal{B}}), q'_{\mathcal{C}} \in \delta_{\mathcal{C}}(a, q_{\mathcal{B}}, q_{\mathcal{C}}) \text{ for some} \\
& (t, q_{\mathcal{B}}, q_{\mathcal{C}}, b') \in S \ \}\} \\
\delta_{\oplus\mathcal{B}'}(a, S^-, S) :=& \{\{ \ (t, (q'_{\mathcal{B}}, q'_{\mathcal{C}}, b')) \ : \ q_{\mathcal{B}'} \in \delta_{\oplus\mathcal{B}}(a, q^-_{\mathcal{B}}, q_{\mathcal{B}}), q'_{\mathcal{C}} \in \delta_{\mathcal{C}}(a, q_{\mathcal{B}}, q_{\mathcal{C}}) \text{ for some} \\
& ((q^-_{\mathcal{B}}, q^-_{\mathcal{C}}, b^-), (q_{\mathcal{B}}, q_{\mathcal{C}}, b')) \in S \text{ and some } (t, q^-_{\mathcal{B}}, q^-_{\mathcal{C}}, b^-) \in S^- \ \}\}
\end{aligned}$$

Initial state is $(\_, (q_{0\mathcal{B}}, q_{0\mathcal{C}}, \mathbf{t}))$ (first triple is immaterial). $\mathcal{C}'$ does nothing along the trunk.

Note that the invariant and the fact that $\mathcal{B}$ is spine-nested means that in any reachable trunk state S there is at most one $\mathcal{B}$-state $q_{\mathcal{B}}$ such that an element of the form $(\_, (q_{\mathcal{B}}, q^1_{\mathcal{C}}, trunk))$ or $(\_, (q_{\mathcal{B}}, q^1_{\mathcal{C}}, spine))$ belongs to S. This specifies the state that $\mathcal{B}$ would have been in on the spine of the input tree. This is used by $\mathcal{C}'$ along the spine after branching.

When the actual convolution being read by $\mathcal{B}'^{\mathcal{C}'}$ splits, then $\mathcal{B}'$ alters its behaviour along the rib, whilst ignoring branching and continuing as above along the spine. After computing $\delta_{\mathcal{B}'}$, as defined above, to yield a new trunk state S', along the rib it non-deterministically adopts a *rib state* $(t, q_{\mathcal{B}})$ where there is an element of the form $(t, (q_{\mathcal{B}}, q^0_{\mathcal{C}}, rib)) \in S'$. $\mathcal{B}'$ will now continue the simulation of $\mathcal{B}$ from state $(t, q_{\mathcal{B}})$. The automaton $\mathcal{C}'$ then has enough information to simulate $\mathcal{C}$ along the rib and the spine and just needs to ensure that it adopts a consistent $q^1_{\mathcal{C}}$ along the spine for the $q^0_{\mathcal{C}}$ along the rib. The following correctly updates a rib state of $\mathcal{B}'$:

$$\begin{aligned}
\delta_{\mathcal{B}'}(a, (t, q_{\mathcal{B}})) :=& \{ \ (t, q'_{\mathcal{B}}) \ : \ q_{\mathcal{B}'} \in \delta_{\mathcal{B}}(a, q_{\mathcal{B}})\} \\
\delta_{\ominus\mathcal{B}'}(a, (t, q_{\mathcal{B}})) :=& \{ \ (\bot, q'_{\mathcal{B}}) \ : \ q_{\mathcal{B}'} \in \delta_{\ominus\mathcal{B}}(a, q_{\mathcal{B}})\} \\
\delta_{\oplus\mathcal{B}'}(a, S^-, ((q^-_{\mathcal{B}}, q^-_{\mathcal{C}}, b^-), q_{\mathcal{B}})) :=& \{ \ (t^-, q'_{\mathcal{B}}) \ : \ q_{\mathcal{B}'} \in \delta_{\oplus\mathcal{B}}(a, q^-_{\mathcal{B}}, q_{\mathcal{B}}) \text{ for some } (t^-, (q^-_{\mathcal{B}}, q^-_{\mathcal{C}}, b^-)) \in S^-\} \\
\delta_{\oplus\mathcal{B}'}(a, (t^-, q^-_{\mathcal{B}}), (\bot, q_{\mathcal{B}})) :=& \{ \ (t^-, q_{\mathcal{B}'}) \ : \ q_{\mathcal{B}'} \in \delta_{\oplus\mathcal{B}}(a, q^-_{\mathcal{B}}, q_{\mathcal{B}})\}
\end{aligned}$$

The only remaining issues are to add indices $\{1\}, \{2\}, \{1, 2\}$ to the nodes of the tree recognised in order to give it the form of a binary convolution; to ignore the $\bullet$s at the end of the branches, both of which are trivial.

In order to get an automaton for $\boldsymbol{r}^{\infty}_{q,q'}$ we do exactly the same thing except that when applying Lemma 5.17 using the exoskeleton we opt for the version of projection asserting the existence of an infinite number of trees with the given skeleton.

The $\boldsymbol{r}$ and $\boldsymbol{r}^{\infty}$ relations are just special cases of the above when Q is a singleton set. □

Closing a symmetric-dendrisophilic structure under reachability does, however, still give a nondisophilic structure. This is enough to get decidability of first-order logic with transitive closure over quantifier free formulae ($\boldsymbol{FO(TC[\Delta_0])}$) for symmetric-dendrisophilic graphs (and so in particular dendrisophilic graphs):

Theorem 6.13. *Symmetric-dendrisophilic graphs have decidable* $\boldsymbol{FO(TC[\Delta_0])}$ *theories.*

Proof. First note that due to Lemma 5.15 it is possible to recognise any *quantifier-free* first-order definable relation $\phi(x, y)$ using a trunk nested automaton. The structure thus remains symmetric-dendrisophilic by adding such relations.

For symmetric-dendrisophilic structures we can perform the first part of the proof of Theorem 6.12, applying Lemma 6.11 to get a symmetric-dendrisophilic chain and Lemma 5.17 on the exoskeleton thereof. In order to recognise convolutions belonging to the reachability relation of a symmetric-dendrisophilic graph, we can use a fully non-deterministic nested-tree automaton that has the full freedom to simulate a premature split of the exoskeleton. Thus closing a symmetric-dendrisophilic graph under reachability gives us a nondisophilic structure. In particular adding relations $\overline{\phi(x, y)}$ is nondisophilic.

Then apply Lemma 6.1 plus decidability of emptiness checking. □

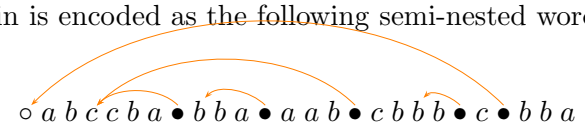
6.5. **Flat Isophilic Chains based on Summaries.** We also define a notion of *flat isophilic chain*, which takes the form of a *semi*-nested-word representing a path in a *flat isophilic* structure. This is essentially the nested-word representation of the isophilic chain defined above (noting that in this special case it would be a non-nested tree). Formally, given a $\mathcal{Q}$-indexed flat isophilic structure $\mathfrak{A}_{\mathcal{Q}}$ (defined in the same way as before) a *flat isophilic chain* is a *semi*-nested-word $c^{\curvearrowleft E}$ over the alphabet $\Sigma \cup \{\circ, \bullet\}$ such that:

- $c(0) = \circ$ and $i \in \mathbf{dom}(E)$ iff $c(i) = \bullet$.
- For each $i \in \mathbf{dom}(E)$ it must be the case that $c(succ(i))$ is distinct from the label of the position coming after $E(i)$ (if it exists) in $\ulcorner {c^{\curvearrowleft E}}_{<i} \urcorner$, where ${c^{\curvearrowleft E}}_{<i}$ is the prefix of $c^{\curvearrowleft E}$ consisting of all positions strictly prior to i.
- Suppose that $c_1 \blacktriangleleft c_2 \blacktriangleleft \cdots \blacktriangleleft c_m$ is the summary ordering of $c^{\curvearrowleft E}$. We require $\pi_\Sigma(c_1), \ldots, \pi_\Sigma(c_m)$ to be a path in $\mathfrak{A}$ and moreover there to exist a function $f : [1..m] \longrightarrow Q$ such that $\pi_\Sigma(c_i) R_i \pi_\Sigma(c_{i+1})$ for some relations of the form $R_i \in \mathcal{Q}(f(i), f(i+1))$.

Example 6.14. Suppose we have the following sequence of non-nested words:

$$abccba, abcbba, abcbaab, abccbbb, abccbbc, bba$$

This flat-isophilic chain is encoded as the following semi-nested word:



The set of flat-isophilic chains of $\mathfrak{A}$ is denoted by $\boldsymbol{Ch}_{\blacktriangleleft}(\mathfrak{A})$. We have an analogue of Lemma 6.11:

Lemma 6.15. *Let $\mathfrak{A}$ be a $\mathcal{Q}$-decorated flat-isophilic structure.*

(1) There exists a semi-nested-word automaton recognising $\boldsymbol{Ch}_{\blacktriangleleft}(\mathfrak{A})$.
(2) For every $\mathcal{Q}$-compatible path in $\mathfrak{A}$ there exists a unique corresponding element in $\boldsymbol{Ch}_{\blacktriangleleft}(\mathfrak{A})$ (and conversely).

Proof. (2) again holds by construction. (1) is a very similar idea to proof of Lemma 6.11. Indeed the construction is essentially the same as the tree automaton given in the proof of Lemma 6.11 mapped to a semi-nested-word automaton recognising precisely the semi-nested-words encoding the trees recognised by the tree automaton.

Take $\mathcal{C}_{p,q}$ to be the union of the finite tree automata recognising the convolutions of relations decorated by (p,q) for each $(p,q) \in Q \times Q$, the co-domain of $\mathcal{Q}$.

The semi-nested-word automaton $\mathcal{C}$ recognising $\boldsymbol{Ch}_{\blacktriangleleft}(\mathfrak{A})$ is given state-space:

$$Q_{\mathcal{C}} := \underbrace{\prod_{\substack{(p,q)\\ \in Q\times Q}} 2^{Q_{\mathcal{C}_{p,q}}}}_{\text{trunk-sim}} \times \underbrace{\left(\prod_{\substack{(p,q)\\ \in Q\times Q}} 2^{Q_{\mathcal{C}_{p,q}}\times Q_{\mathcal{C}_{p,q}}}\right)^{\perp}}_{\text{left-branch-sim}} \times \underbrace{\left(\bigcup_{\substack{(p,q)\\ \in Q\times Q}} Q_{\mathcal{C}_{(p,q)}}\right)^{\perp}}_{\text{right-branch-sim}}$$

The initial state is $(I_{\vec{(p,q)}}, \perp, \perp)$ where $I_{p,q}$ is the set of initial states of $\mathcal{C}_{p,q}$.

Suppose that $\mathcal{Q}_{\mathcal{C}}$ has read the first segment u of a semi-nested-word. The trunk-sim deterministically keeps track of the state that $\mathcal{C}_{p,q}$ would be in if it read $\ulcorner u \urcorner$ from its initial state. In order to do this, it ignores the targets of pointers but at the source of a pointer v it will pick up from where it left off at $E(v)$ rather than from the predecessor of v.

Whenever transitioning to the *successor of a target*, a fresh *left*-branch-sim is started. If u' is the segment of the word that has been read since and including the immediate successor of the target, then the left-branch-sim component tracks the pairs (r, r') for each $\mathcal{C}_{p,q}$, meaning the automaton could have started in state r and read $\ulcorner u' \urcorner$ ending in r'.

Whenever transition is made from the *successor of a source* v a fresh *right*-branch-sim is started. This simulates a specific $\mathcal{C}_{p,q}$ which may be chosen arbitrarily. The state the simulation begins with must meet the following constraints: (p,q) may be arbitrary if the previous right-branch-sim was $\perp$, otherwise only q may be arbitrary and p must be such that the previous right-branch-sim was simulating (r,p) for some $r \in Q$. The actual simulated state c is chosen so that there exists some pair (a,b) in the current left-branch-sim of $Q_{\mathcal{C}_{p,q}}$ such that b is accepting for $Q_{\mathcal{C}_{p,q}}$ and there exists some state d in the trunk-sim component of the state at $E(v)$ such that $\mathcal{C}_{p,q}$ could branch from d to (a,c).

Whenever a fresh right-branch-sim is begun $Q_{\mathcal{C}}$ must also ensure that the previous right-branch-sim terminated with an accepting state and reject otherwise. □

Whilst this could be used to show that $\boldsymbol{r}$ and $\boldsymbol{r}^{\boldsymbol{\infty}}$ can be added to flat-isophilic structures, this can already be seen by applying Theorem 6.12 to this special case. The utility of flat-isophilic chains lies in the fact that they themselves naturally form isophilic and dendrisophilic structures.

6.6. **Graphs from Flat Chains.** Consider a $\mathcal{Q}$-decorated flat-isophilic graph $\mathfrak{A}_{\mathcal{Q}}$. Let $\mathcal{R}_{\mathfrak{A}}$ be the set of its *binary* relations and let $\mathcal{P}_{\mathfrak{A}}$ be the set of its *unary* predicates. Let us also consider two maps $\mathcal{Q}_{pop}, \mathcal{Q}_{push} : \mathcal{R}_{\mathfrak{A}} \longrightarrow Q \times Q$ and a further map:

$\mathcal{Q}_{panic} : \mathcal{P}_{\mathfrak{A}} \times \mathcal{P}_{\mathfrak{A}} \longrightarrow Q \times Q$. The structure $\boldsymbol{Ch}^{\boldsymbol{I}}_{\blacktriangleleft}(\mathfrak{A}_{\mathcal{Q}}, \mathcal{Q}_{pop}, \mathcal{Q}_{push})$ has domain $\boldsymbol{Ch}_{\blacktriangleleft}(\mathfrak{A})$, and decoration $\mathcal{Q}_{\blacktriangleleft}$ where it has relations:

- For each binary relation R of $\mathfrak{A}$ a relation $R_{\blacktriangleleft}$ relating each encoding of the path $w_1, \ldots, w_{m-1}, w_m$ to the encoding of the path $w_1, \ldots, w_{m-1}, w'_m$ where $w_m R w'_m$. We set $\mathcal{Q}_{\blacktriangleleft}(R_{\blacktriangleleft}) := \mathcal{Q}(R)$.
- For each binary relation R of $\mathfrak{A}$ a relation R_{pop} relating the encoding of a path $w_1, \ldots, w_{m-1}, w_m$ to $w_1, \ldots, w_{m-1}$ whenever $w_{m-1} R w_m$. We set $\mathcal{Q}_{\blacktriangleleft}(R_{pop}) := \mathcal{Q}_{pop}(R)$.
- For each $(p, q) \in Q_{push}$ a relation R_{push} relating the encoding of a path $w_1, \ldots, w_{m-1}, w_m$ to $w_1, \ldots, w_{m-1}, w_m, w'_m$ whenever $w_m R w'_m$. We define $\mathcal{Q}_{\blacktriangleleft}(R_{push}) := \mathcal{Q}_{push}(R)$.

We will now consider some extensions of this structure with additional relations:

- For each pair of *unary* predicates P_1 and P_2 we have a relation $(P_1, P_2)_{panic}$ that relates every encoding of a path $w_1, \ldots, w_{m-1}, w_m$ where $w_m \in P_1$ to the encoding of the shortest sub-path $w_1, \ldots, w_r$ such that $r < m$ and w_m is a prefix of w_i for every $r < i \leq m$ with w_m *not* a prefix of w_r, and $w_r \in P_2$. We set $\mathcal{Q}_{\blacktriangleleft}((P_1, P_2)_{panic}) := \mathcal{Q}_{panic}(P_1, P_2)$. The 'panic' terminology is inspired by [20].

Lemma 3.7 hints at how this panic relation will be used to model collapse on order-2 links.

The structure $\boldsymbol{Ch}^{\boldsymbol{D}}_{\blacktriangleleft}(\mathfrak{A}, \mathcal{Q}_{pop}, \mathcal{Q}_{push}, \mathcal{Q}_{panic})$ extends $\boldsymbol{Ch}^{\boldsymbol{I}}_{\blacktriangleleft}(\mathfrak{A}, \mathcal{Q}_{pop}, \mathcal{Q}_{push})$ with the relation $(P_1, P_2)_{panic}$ for every $P_1, P_2 \in \mathcal{P}_{\mathfrak{A}}$.

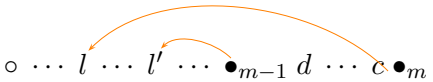
First we give a lemma explaining how to 'compute' the sub-chain resulting from the *panic* relation given a starting chain.

Lemma 6.16. *Let c be a chain encoding $w_1, \ldots, w_{m-1}, w_m$ where $m \geq 2$. Then there exists a unique chain c' encoding $w_1, \ldots, w_r$ such that $r < m$ and w_m is a prefix of w_i for every $r < i \leq m$ and such that r is minimal. If the final position of c is not $\bullet$, then $r = m - 1$. Otherwise it is $\bullet$ and hence the source of a pointer with target l. Then the unique c' relating to c in this way is the subword of c finishing with the right-most $\bullet$ or $\circ$ symbol left of l.*

Proof. We argue by induction on m. The case when $m = 2$ is immediate from the fact that $r = 1$ is the only shorter chain and trivially satisfies the property since $i = 2$ is the only possible value for i. For the induction step suppose first that c does not end in $\bullet$. This means that c has the form:

where no $\bullet$ occurs amongst the final $\cdots$. The definition of chain ensures that $\pi_\Sigma(\ulcorner \circ \cdots l \cdots \bullet \urcorner)$ is the largest common prefix of w_{m-1} and w_m as a must be different to whatever follows l in $\pi_\Sigma(\ulcorner \circ \cdots l \cdots c \urcorner)$. Thus w_m cannot be a prefix of w_{m-1} and so $r = m - 1$ is the necessary value of r (so that i only takes the value m).

Suppose now that c has the form:



where the right-most two $\cdots$ do not contain any $\bullet$ symbols. Thus we can see that w_m is a prefix of w_{m-1}. Now let us adjust the chain by replacing w_{m-1} with w'_{m-1} formed by moving l' to the left to the same position as l and removing the right-most $d \cdots c$ segment. Indeed $w'_{m-1} = w_m$. The r given by the induction hypothesis to this modified chain is thus

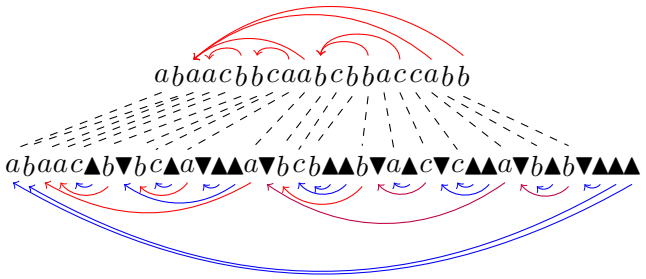


Figure 10: The Υ-mapping of a semi-nested word.

correct for $w'_{m-1} = w_m$. But since w_m is a prefix of the original w_{m-1} it must be correct for the original chain too.

The only other form that c could take is:

$$\circ \cdots \bullet_{m-1} \; a \cdots l \cdots \bullet_m$$

where the right-most $\cdots$ does not contain any $\bullet$ symbol. By the induction hypothesis $\pi_\Sigma(\ulcorner \circ \cdots \bullet_{m-1} \; a \cdots l \urcorner)$ is not a prefix of w_{m-2} and so neither can be w_m. Thus again $r = m - 1$ is the appropriate value for r, as required. □

Now we can place the chain derived structures in our hierarchy.

Theorem 6.17. *Let $\mathfrak{A}_\mathcal{Q}$ be a Q-decorated flat-isophilic graph. Then for every $\mathcal{Q}_{pop}, \mathcal{Q}_{push} : \mathcal{R}_\mathfrak{A} \longrightarrow Q \times Q$ and $\mathcal{Q}_{panic} \subseteq Q \times Q$ the structure $\boldsymbol{Ch}^{\boldsymbol{D}}_{\blacktriangleleft}(\mathfrak{A}, \mathcal{Q}_{pop}, \mathcal{Q}_{push}, \mathcal{Q}_{panic})$ is dendrisophilic and the structure $\boldsymbol{Ch}^{\boldsymbol{I}}_{\blacktriangleleft}(\mathfrak{A}, \mathcal{Q}_{pop}, \mathcal{Q}_{push})$ is isophilic.*

Proof. It is first necessary to encode the semi-nested words making up the domain of these structures as nested-words, since isophilic and dendrisophilic structures require a domain of nested-words. A naïve encoding is suggested by Figure 1, but this results in convolutions splitting too early to decide the $R_{\blacktriangleleft_{p,q}}$ relations. Instead we use an encoding Υ illustrated in Figure 10. Υ ensures the convolution does not split until the *sources* of the differing pointers. This is achieved by treating all positions not sourcing a pointer as the target of a pointer. This results in 'spurious pointers' that are discharged using additional ▲-labelled poistions. Positions labelled ▼ are targets used to simulate pointers sharing the same target.

Suppose that $w^{\frown E}$ is a semi-nested-word over an alphabet Σ, viewing $\mathbf{dom}(w) \subseteq \{\star\}^*$. We view $\mathbf{dom}(\Upsilon(w))$ as being a prefix closed subset of $(\star + ▲ + ▼)^*$. For $u \in (\star + ▲ + ▼)^*$ we write $\pi_\star(u)$ to denote the string that results from deleting the ▲ and ▼ elements from u. Note also that well-nesting ensures that we can uniquely define the pointers in a nested-word purely by specifying which positions belong to $\mathbf{img}(E)$ and which to $\mathbf{dom}(E)$.

$\Upsilon(w^{\frown E}) =: w'^{\frown E'}$ is defined as follows: where u is the longest element of $\mathbf{dom}(w)$, take $\mathbf{dom}(w')$ to be the prefix closure of $\{u'\}$ where u' satisfies the following:

- We have $\pi_\star(u') = u$.
- If the ith position of u is in $\mathbf{dom}(E)$ then the ith $\star$ in u' is in $\mathbf{dom}(E')$ and is followed immediately in u' by a single ▼ which is in $\mathbf{img}(E')$

- If the ith position of u is *not* in $\mathbf{dom}(E)$ (and may or may not be in $\mathbf{img}(E)$) then the ith $\star$ in u' is in $\mathbf{img}(E')$.
- If the ith position of u is in $\mathbf{dom}(E)$ and maps under E to the jth position, then the ith $\star$ in u', which we denote l, is immediately preceded by a sequence of ▲s in $\mathbf{dom}(E')$ ensuring that $E'(l)$ is either the jth $\star$ in u' or else is a ▼ following a kth element of $\pi_\star(u')$ where that kth element maps to the jth element of u under E. Well-nesting of E' ensures that only one of these options is possible and that when the ▼ option is taken there is precisely one suitable ▼.
- We add ▲'s in $\mathbf{dom}(E)$ to the end to discharge any remaining elements in $\mathbf{img}(E)$.

Finally we label each $\star$ position in u' with the corresponding label from u and label each ▼ position with ▼ and each ▲ position with ▲. Observe that Υ must be injective and hence respect equality.

Given a semi-nested word automaton $\mathcal{A}$ we can construct a nested-word automaton $\mathcal{A}'$:

$$\mathcal{L}(\mathcal{A}') = \{\ \Upsilon(w^{\curvearrowleft E}) \in \mathbf{NWord}(\Sigma \uplus \{\ ▲, ▼\ \})\ :\ w^{\curvearrowleft E} \in \mathcal{L}(\mathcal{A})\ \}$$

$\mathcal{A}'$ simulates $\mathcal{A}$ when reading $\star$ positions. It must non-deterministically guess whether a $\star$ position in $w'^{\curvearrowleft E'}$ belonging to $\mathbf{img}(E')$ corresponds to a position of $w^{\curvearrowleft E}$ in $\mathbf{img}(E)$ or in neither $\mathbf{img}(E)$ nor $\mathbf{dom}(E)$. It flags its guess in the control-state at this position l. It may then proceed to verify its guess at the position in $w'^{\curvearrowleft E'}$ in $\mathbf{dom}(E')$ mapping to l under E'. If this position is labelled with ▲ then l is not in $\mathbf{img}(E) \cup \mathbf{dom}(E)$, otherwise it is in $\mathbf{img}(E)$. $\mathcal{A}'$ must also repeat the state at $E(l)$ at l where l is a $\star$ node in $\mathbf{dom}(E')$ at the following ▼ node. This enables the correct simulation of multiple pointers with the same target. Since this is a nested-word automaton, it may be determinised.

We now turn our attention to recognising the relations in the structures considered by the theorem. Let R be a relation of $\mathfrak{A}$. Since $\mathfrak{A}$ is *flat*-isophilic it must be that R is recognised by a standard (non-nested) tree automaton.

Recall that the relation $R_◀$ relates flat-isophilic chains c and c' (semi-nested-words) made up respectively of elements $w_1, w_2, \ldots, w_{m-1}, w_m$ and $w_1, w_2, \ldots, w_{m-1}, w'_m$ where $w_m R w'_m$. Recall that $w_m = \ulcorner c \urcorner$ and $w'_m = \ulcorner c' \urcorner$. Let i_c be the last position in c (labelled •) to be a source of a pointer. Let $i_{c'}$ be the analogue for c'. Recall further that we can obtain the semi-nested-word representation of the chain consisting of the first $m-1$ elements in c by taking the prefix consisting of everything strictly before i_c (so the target of i_c may no longer be the target of a pointer). Similarly we can obtain a representation of the first $m-1$ elements of c'. We thus have $c R_◀ c'$ iff the following two conditions are met:

- $\ulcorner c \urcorner R \ulcorner c' \urcorner$
- The prefix of c consisting of everything strictly before i_c is identical to the prefix of c' consisting of everything strictly before $i_{c'}$ (indeed $i_c = i_{c'}$)

In order to understand the purpose of Υ, observe that i_c and $i_{c'}$ might source pointers with different targets. This would result in a 'semi-nested word convolution' splitting at the target of i_c or $i_{c'}$, preventing the necessary comparisons between the target of i_c and i_c itself from being carried out. This problem is eliminated under the Υ mapping as a pointer-caused difference can only occur at a source—non-sources are all treated as targets. The two conditions above are met under precisely the following conditions:

- $\ulcorner c \urcorner R \ulcorner c' \urcorner$

- The branches from the tip of the trunk of $\bigotimes \langle \Upsilon(c), \Upsilon(c') \rangle$ have the form $\Upsilon(i_c)\blacktriangledown\Sigma^*\blacktriangle^*$ and $\blacktriangle^*\Upsilon(i_{c'})\blacktriangledown\Sigma^*\blacktriangle^*$, or $\blacktriangle^*\Upsilon(i_c)\blacktriangledown\Sigma^*\blacktriangle^*$ and $\Upsilon(i_{c'})\blacktriangledown\Sigma^*\blacktriangle^*$ (where $\Upsilon(i_c)$ and $\Upsilon(i_{c'})$ are the corresponding positions of i_c and $i_{c'}$ under Υ).

Since R can be recognised by a non-nested tree automaton, it must be recognised by a non-nested automaton $\mathcal{A}_R$ that acts *deterministically* on the trunk of the tree. (We can use the powerset construction to determinise the trunk.) We must then have a nested-word automaton $\mathcal{A}_R^N$ whose state after reading a nested-word $\Upsilon(w^{\frown E})$ for semi-nested-word $w^{\frown E}$ is the same as that of $\mathcal{A}_R$ after reading $\ulcorner w^{\frown E} \urcorner$. This can be achieved by programing $\mathcal{A}_R^N$ to recall its state at the target of a pointer when reading its source (at $\bullet$-labelled positions). We also encode in each $\mathcal{A}_R^N$ state following a source the state and label that was at that source's target (or in the case of a $\blacktriangledown$, the state and label obtained by recursively looking at the target of the source preceding the $\blacktriangle$).

We can then construct a non-nested tree automaton $\mathcal{A}_R^S$ that ignores the trunk of the convolution $\bigotimes \langle \Upsilon(c), \Upsilon(c') \rangle$ until it reaches the tip. It will then verify that the left and right branches conform to one of the patterns above. Suppose that the branches from the tip of the trunk of $\bigotimes \langle \Upsilon(c), \Upsilon(c') \rangle$ have the form $\Upsilon(i_c)\blacktriangledown\Sigma^*\blacktriangle^*$ (left) and $\blacktriangle^*\Upsilon(i_{c'})\blacktriangledown\Sigma^*\blacktriangle^*$ (right) (the other option is similar). This means that the target of the pointer from $\Upsilon(i_{c'})$ must represent the branching point of $\bigotimes \langle \ulcorner c \urcorner, \ulcorner c' \urcorner \rangle$ since the target of $\Upsilon(i_{c'})$ must be closer to the root of $\bigotimes \langle \Upsilon(c), \Upsilon(c') \rangle$ than the target of $\Upsilon(i_c)$. Let σ_c be the Σ^* component of the left branch and $\sigma_{c'}$ that of the right. Moreover, let $\sigma_\blacktriangle$ be the sequence in Σ^* corresponding to the targets of the first $\blacktriangle^*$ in the right branch (if the target is a $\blacktriangledown$ then the Σ element corresponding to the $\blacktriangle$ in the original semi-nested word). The left branch of $\bigotimes \langle \Upsilon(c), \Upsilon(c') \rangle$ must then be $\sigma_\blacktriangle{}^{\leftarrow}\sigma_c$ and the right branch $\sigma_{c'}$. This is because the first $\blacktriangle^*$ on the right branch scans backwards through the segment from the target of i_c to the target of $i_{c'}$. The automaton $\mathcal{A}_R^S$ can thus be extended to simulate $\mathcal{A}_R$ on the left and right branches of $\bigotimes \langle \ulcorner c \urcorner, \ulcorner c' \urcorner \rangle$, using the label information provided by $\mathcal{A}_R^N$ to find the labels making up $c_\blacktriangle$ and obtaining the possible control-states of $\mathcal{A}_R$ at the tip of the trunk of $\bigotimes \langle \ulcorner c \urcorner, \ulcorner c' \urcorner \rangle$ by looking at the target of i_c. Thus we get a path-nested automaton $\mathcal{A}_R^{N\mathcal{A}_R^S}$ recognising $R_\blacktriangleleft$.

For R_{push} and R_{pop} it suffices to exhibit a nested-word automaton $\mathcal{A}_R^P$ acting on single flat-isophilic chains c made up from $w_1, w_2, \ldots, w_{m-1}, w_m$ such that c is accepted iff $w_{m-1} R w_m$. Write $i_{c'}$ and i_c for the penultimate and final $\bullet$ in c respectively. R_{push} and R_{pop} can then be recognised by running such an $\mathcal{A}_R^P$ along the longest branch c of a convolution whilst additionally requiring that the other component (left in the case of R_{push} and right for R_{pop}) of the convolution be identical up to $i_{c'}$ and then consist only of $\blacktriangledown\blacktriangle^*$.

We can construct a non-deterministic $\mathcal{A}_R^P$ since it acts only on nested-words and hence can be determinised. Noting that the target of i_c represents the branching point of $\bigotimes \langle w_{m-1}, w_m \rangle$, $\mathcal{A}_R^P$ can simply simulate $\mathcal{A}_R$ along summaries, guessing the element that is the target of i_c. At this point it simulates branching, following the link to simulate the w_m branch and moving to its successor to simulate the w_{m-1} branch. The node guessed to be the target of i_c can be flagged for subsequent verification when the final $\bullet$ (i_c) is reached.

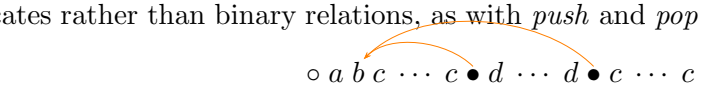
Now we show that $(P_1, P_2)_{panic}$ can be recognised by a spine-nested automaton for every pair of unary predicates P_1 and P_2 recognisable by a non-nested word automaton. Given flat-isophilic chain c it is easy to construct a deterministic nested-word automaton acting on $\Upsilon(c)$ to check whether the final element in the chain c satisfies some unary predicate P. After all, we just need to check whether $\ulcorner c \urcorner \in P$ and P itself will be recognised by some deterministic non-nested word automaton $\mathcal{A}_P$. So we just need to have a nested-word

automaton that reads $\Upsilon(c)$ simulating $\mathcal{A}_P$, picking up from the state at the target of a pointer when reading the source of a pointer at a Σ-labelled position. In particular, given two flat-isophilic chains c and c' and *two* predicates P_1 and P_2 we can have a single nested-word automaton simulating both $\mathcal{A}_{P_1}$ and $\mathcal{A}_{P_2}$ simultaneously. This single automaton can act on each branch of $\bigotimes \langle \Upsilon(c), \Upsilon(c') \rangle$ independently.

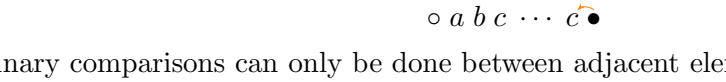
Thus we just need to check that we can construct a spine nested automaton that recognises convolutions $\bigotimes \langle \Upsilon(c), \Upsilon(c') \rangle$ where c encodes $w_1, \ldots, w_{m-1}, w_m$ and c' encodes the longest $w_1, \ldots, w_r$ for some $r < m$ such that w_m is a prefix of w_i for every $r < i \leq m$, but w_m is not a prefix of w_r. By Lemma 6.16 we just need to ensure that $\Upsilon(c')$ is the same as an initial prefix of $\Upsilon(c)$ and that the the first position in $\Upsilon(c)$ corresponding to a target in c after the end of $\Upsilon(c')$ is as dictated by Lemma 6.16. This can be checked by a nested-tree automaton that marks this target. This will require a non-deterministic guess in the case of $\Upsilon(c)$—we have seen how the fact a Σ symbol is a target in the underlying semi-nested words can only be verified at the source. However, since this occurs in the left-branch from the trunk of the convolution, it can be carried out by a spine nested automaton.

□

To understand why the preconditions for *panic* relations have to be based on unary predicates rather than binary relations, as with *push* and *pop*, consider the following chain:



There is no way for an automaton reading left-to-right to be able to determine whether the two instances of $c \ \ldots \ c$ have the same contents. If they were next to each other without the intervening $d \ \cdots \ d$ such a comparison would be possible; equality would give:



So binary comparisons can only be done between adjacent elements in the chain.

6.7. **Graphs from Chains of Nested-Words.** We build the 'next level' from $\mathcal{Q}$-decorated graphs $\mathfrak{A}_\mathcal{Q}$ that are either isophilic or dendrisophilic. The structure $\boldsymbol{Ch}^{\boldsymbol{T}}_{\prec}(\mathfrak{A}, \mathcal{Q}_{pop}, \mathcal{Q}_{push})$ is a structure whose domain consists of either isophilic or dendrisophilic chains (depending on $\mathfrak{A}$), which are nested-*trees* and is defined in a way completely analogous to $\boldsymbol{Ch}^{\boldsymbol{I}}_{\blacktriangleleft}$—the definition can almost be read verbatim except that we name the decorating function $\mathcal{Q}_\prec$ instead of $\mathcal{Q}_\blacktriangleleft$, write $R_\prec$ instead of $R_\blacktriangleleft$ and 'representation of a chain' refers to an isophilic chain (nested-tree) rather than a flat-isophilic chain (nested-word).

Theorem 6.18. *Let $\mathfrak{A}_\mathcal{Q}$ be a Q-decorated isophilic (*resp. *dendrisophilic) structure. For every $\mathcal{Q}_{pop}, \mathcal{Q}_{push} : \mathcal{R}_\mathfrak{A} \longrightarrow Q \times Q$ $\boldsymbol{Ch}^{\boldsymbol{T}}_{\prec}(\mathfrak{A}, \mathcal{Q}_{pop}, \mathcal{Q}_{push})$ is tree-isophilic (*resp. *tree-nondisophilic).*

Proof. To recognise a relation $R_\prec$ we just need to check that the two trees in the binary convolution differ only in their right-most branches and then run the automaton recognising R on this pair of right-most branches. For R_{push} and R_{pop} we just need to check that one tree is precisely the subtree consisting of all nodes not belonging exclusively to the right-most branch of the other. We then just need to check that the right-most branch and branch immediately to the left of that branch are related by R, which can be done by treating this pair of branches like a convolution and applying the automaton for R. □

6.8. **Relationship with Tree Automatic Structures.** It is known that the word and tree-automatic ordinals are precisely those below ω^ω and ω^{ω^ω} respectively [14]. We illustrate the utility of the flat, isophilic, tree-isophilic hierarchy by re-exhibiting the definability of ω^ω, ω^{ω^ω} and additionally $\omega^{\omega^{\omega^\omega}}$ as a natural progression exploiting isophilic chains.

All ordinals α have a unique *Cantor Normal Form* $\alpha = \sum_{i=1}^{k} \omega^{\beta_i}$ where $\beta_1 \geq \beta_2 \geq \cdots \geq \beta_k \geq 0$ are ordinals. When $\alpha < \epsilon_0$ we also have $\beta_1 < \alpha$. If $\alpha' = \sum_{i=1}^{k'} \omega^{\beta'_i}$ then:

$$\alpha \leq \alpha' \text{ iff } \begin{cases} k' \geq k \text{ and } \beta_i = \beta'_i \text{ for every } 1 \leq i \leq k \\ \text{if } j \text{ is the least } i \text{ s.t.} \beta_i \neq \beta'_i \text{ then } \beta_j < \beta'_j \end{cases}$$

We can thus encode every ordinal below ω^ω (for which the β_i will be natural numbers) as a finite-word over the alphabet $\{a, b\}$. This will take the form $b^{\beta_1}\ a\ b^{\beta_2}\ a\ \cdots\ a\ b^{\beta_k}$. This is how it is shown that the set of ordinals below ω^ω is word-automatic; in fact they are flat-isophilic. As soon as a difference in two encodings is spotted (which is when a convolution would split) the ordering can be inferred. So let $\boldsymbol{On(\omega^\omega)}$ be this flat-isophilic structure representing the ordinals below ω^ω under the *strict* ordering $<$.

An ordinal below ω^{ω^ω} has a Cantor Normal Form for which the $\beta_i < \omega^\omega$—that is it can be represented as a chain of ordinals less than ω^ω and $\alpha < \alpha'$ if there is a common prefix of the chains for α and α' that is followed by β in α and β' in α' such that $\beta < \beta'$. We can thus take domain $\boldsymbol{Ch_{\blacktriangleleft}(On(\omega^\omega))}$. The relation $R_<$ induced on flat isophilic chains from the ordering on $\boldsymbol{On(\omega^\omega)}$ then gives the correct order relation between ordinals for which $k = k'$ and for all $i < k$ we have $\beta_i = \beta'_i$. But the automaton recognising $R_<$ can then be easily extended to recognise the full order relation by ignoring subsequent elements of the chain after it has split other than to checking that $k' > k$ when the chain encoding α is a prefix of the chain encoding α'.

In a similar vein we can finish the hierarchy by constructing $\boldsymbol{On(\omega^{\omega^\omega})}$ from isophilic chains, making all ordinals below $\omega^{\omega^{\omega^\omega}}$ are tree-isophilic. Delhommé's result [14] thus implies that there exist tree-isophilic structures that are not tree-automatic.

It is also straightforward to see that every tree-automatic structure is tree-isophilic. To prevent convolutions from splitting due to a difference in labels we shift the labels to an additional child of each node. An automaton reading the convolution guesses the label at each node and verifies the guess by checking the child containing the label. This transformation is illustrated in Figure 11.

Word ($\boldsymbol{WAut}$) and tree ($\boldsymbol{TAut}$) automatic structures are *not* closed under reachability (unlike the flat-isophilic, isophilic and dendrisophilic structures) and considering the relationship between nested-words and trees:

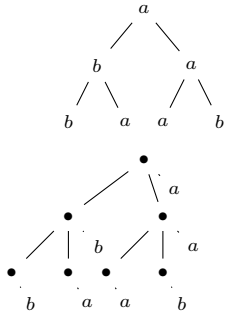


Figure 11: 'Leafication' of a tree

Theorem 6.19. $\boldsymbol{Iso_1 \subsetneq WAut \subsetneq TAut \subsetneq Iso_3}$ *and* $\boldsymbol{Iso_1 \subsetneq Iso_2 \subsetneq TAut}$.

It is also worth noting that every word-automatic structure is nondisophilic:

Theorem 6.20. $\boldsymbol{WAut \subsetneq nIso_2}$

Proof. Let $\mathfrak{A}$ be a word-automatic structure. We encode each element w of the domain A of $\mathfrak{A}$ as a nested-word of the form

$\bullet^{|w|}w^{\leftarrow}$, where $w^{\leftarrow}$ is the word w written backwards. Each element of $w^{\leftarrow}$ should source a pointer, pointing to a canonical target in $\bullet^{|w|}$ determined by well-nesting. Suppose that A is over the alphabet Σ. Word-automatic n-ary convolutions can then be simulated along the trunk of a nested-word convolution since the nested-tree automaton can nondeterministically guess an n-tuple in Σ^n representing the Σ-label of each of the components in the nested-word convolution. These guesses can then be verified when reading the $w^{\leftarrow}$ component of the encoding of each word. This may occur after the nondisophilic convolution has split, but this does not matter since the simulation of the synchronisation of the word-automatic convolution would have been based on the guess whilst reading the trunk.

Strictness of the inclusion comes from the fact that $\boldsymbol{nIso_2}$ contains $\boldsymbol{Iso_2}$, which contains the ordinal ω^{ω^ω}, which is not word-automatic [14]. □

7. From CPDA to Rewriting

In this section we will complete the proof of Theorem 3.8 by establishing the following:

Lemma 7.1. $\mathbf{Pd}_{\epsilon 2} \subseteq \boldsymbol{\mathcal{RW}_{rr}}$ *and* $\mathbf{Pd}_{\epsilon 2}^{\mathcal{C}} \subseteq \boldsymbol{\mathcal{RW}_{sr}}$

In fact this will be broken down into showing that $\boldsymbol{Iso_2} \subseteq \boldsymbol{\mathcal{RW}_{rr}}$ and $\boldsymbol{dIso_2} \subseteq \boldsymbol{\mathcal{RW}_{sr}}$ (the subject of the next subsection) as well as $\mathbf{Pd}_{\epsilon 2} \subseteq \boldsymbol{Iso_2}$ and $\mathbf{Pd}_{\epsilon 2}^{\mathcal{C}} \subseteq \boldsymbol{dIso_2}$. Note in particular that Lemma 4.1 then also implies that $\mathbf{Pd}_{\epsilon 2} = \boldsymbol{Iso_2}$ and $\mathbf{Pd}_{\epsilon 2}^{\mathcal{C}} = \boldsymbol{dIso_2}$, making the isophilic and dendrisophilic structures another precise characterisation of the ϵ-closures of the 2-PDA and 2-CPDA graphs.

7.1. Equivalence with Automatic Structures.

Lemma 7.2. $\boldsymbol{Iso_2} \subseteq \boldsymbol{\mathcal{RW}_{rr}}$ *and* $\boldsymbol{dIso_2} \subseteq \boldsymbol{\mathcal{RW}_{sr}}$.

Proof. Consider an isophilic structure $\mathfrak{A}$ with *deterministic* nested-word automaton $\mathcal{A}$ recognising its domain (we can determinise if necessary) and path-nested automata ${\mathcal{B}_i}^{\mathcal{C}_i}$ recognising convolutions for its e_i-labelled edges for $1 \leq i \leq k$. We define the deterministic word automaton $\wp(\mathcal{C}_i)$ to have state space $2^{Q_{\mathcal{C}_i}}$ and keep track of the states that $\mathcal{C}_i$ could be in if it had not yet branched (we implicitly assume that it has access to the current state of $\mathcal{B}_i$).

We define an equivalent rat-rat *suffix*-rewrite system. Recall the notion of state decoration $\mathfrak{D}_{\mathcal{A}}$ for some nested-tree (or in particular non-nested-tree) automaton $\mathcal{A}$ from subsection 4.2. We extend this to $\mathfrak{D}_{\mathcal{A}_1,\ldots,\mathcal{A}_m}$ for several automata $\mathcal{A}_1,\ldots,\mathcal{A}_m$, which adds decorations simultaneously for all of them (we can view this as multiple applications of $\mathfrak{D}_{\mathcal{A}_i}$ for each $1 \leq i \leq m$). We take as the domain of the rat-rat suffix-rewrite system the nested-word language:

$$\mathfrak{D}_{\mathcal{A},\mathcal{B}_1,\ldots,\mathcal{B}_k,\wp(\mathcal{C}_1),\ldots,\wp(\mathcal{C}_k)}(\mathfrak{V}(\mathcal{L}(\mathcal{A})))$$

Recall that this set of nodes is in bijective correspondence with $\mathcal{L}(\mathcal{A})$ and that $\mathfrak{V}$ just adds decorations indicating whether a position is the source of a pointer, target or neither.

We then add multiple suffix rewrite rules labelled e_i of the form:

$$\mathcal{L}_{i\vec{a},b,(p,q)} \longrightarrow \mathcal{L}'_{i\vec{a},b',(p,q)}$$

for each $\vec{a}$ in the alphabet $\Sigma \times Q_{\mathcal{A}} \times Q_{\mathcal{A}}^{\perp} \times \mathbb{B}$ of the domain, $b \neq b' \in \mathfrak{V}(\Sigma)$ and each pair $p, q \in Q_{\mathcal{C}_i}$ such that there exists an $r \in \boldsymbol{st}_{\mathcal{C}_i}(\vec{a})$ such that $\mathcal{C}_i$ could branch to (p, q) from r taking into account $\boldsymbol{st}_{\mathcal{B}_i}(a)$.

The elements of $\mathcal{L}_{i\vec{a},b,(p,q)}$ are defined to be those of the form $\vec{a}\vec{b}w$ where $\boldsymbol{symb}(\vec{b}) = b$ and $\boldsymbol{symb}(\vec{b})\boldsymbol{symb}(w)$ could be recognised by $\mathcal{C}_i$ starting in state p and taking the $\boldsymbol{st}_{\mathcal{B}_i}$ decorations to be the state of $\mathcal{B}_i$. There is no other restriction placed on w. $\mathcal{L}'_{i\vec{a},b'(p,q)}$ is defined to be similar except that it considers $\mathcal{C}_i$ beginning in state q and we consider b' in place of b. We thereby simulate the corresponding forking of the convolution both in terms of the behaviour of automata at the fork and the fact that a fork actually does occur in this position (as enforced by $b \neq b'$, which due to $\mathfrak{V}$ takes into account pointers). Note also that since $\mathcal{C}_i$ is a finite non-nested automaton it must be the case that both of these languages are regular. Hence this is indeed a rat-rat suffix rewrite system, as required.

Now consider a *dendrisophilic* structure $\mathfrak{A}$, again with deterministic nested-word automaton $\mathcal{A}$ recognising its domain, and spine nested automata $\mathcal{B}_i^{\mathcal{C}_i}$ recognising its e_i-edge convolutions. Define ${\mathcal{B}_i^{\mathcal{C}_i}}^{\Cap}$ to be the deterministic nested-word automaton with state space $2^{Q_{\mathcal{B}_i} \times Q_{\mathcal{C}_i} \times Q_{\mathcal{B}_i} \times Q_{\mathcal{C}_i}}$ where an element $((q,p),(q',p'))$ belonging to the state means that if $\mathcal{B}_i^{\mathcal{C}_i}$ had started in state (q,p) at the most recent target not discharged by a source, then $\mathcal{B}_i^{\mathcal{C}_i}$ could now be in state (q',p') assuming no branching. This is the same construction as used in the determinisation proof of nested-word automata [3] and, as seen there, a $\mathcal{B}_i^{\Cap}$ can be constructed to deterministically maintain this state.

We form the equivalent sum-rat suffix rewrite system (which as before may be converted to a prefix rewrite system) taking the domain to be:

$$\mathfrak{D}_{\mathcal{B}_1^s,\ldots,\mathcal{B}_k^s,\wp(\mathcal{C}_1),\ldots,\wp(\mathcal{C}_k),{\mathcal{B}_1^{\mathcal{C}_1}}^{\Cap},\ldots,{\mathcal{B}_k^{\mathcal{C}_1}}^{\Cap}}(\mathfrak{V}(\mathcal{L}(\mathcal{A})))$$

where $\mathcal{B}_i^s$ is $\mathcal{B}_i$ restricted to acting (deterministically) on a spine (*i.e.* pretending no branching ever takes place). In a manner similar to before we take a family of rewrite rules for each e_i of the form: $\mathcal{L}_{i\vec{a},b,(p,q)} \longrightarrow \mathcal{L}'_{i\vec{a},b',(p,q)}$ whenever $b \neq b' \in \mathfrak{V}(\Sigma)$ and $\mathcal{C}_i$ could branch from a state in $\boldsymbol{st}_{\wp(\mathcal{C}_i)}(\vec{a})$ to p down the left-hand branch and q down the right-hand branch.

The language $\mathcal{L}'_{i\vec{a},b',(p,q)}$ may be defined exactly as before because the right-hand branch of the convolution is essentially the same as in the isophilic case, treating $\mathcal{B}_i^s$ like the path-nested automaton. This language must thus also be regular.

The language $\mathcal{L}_{i\vec{a},b,(p,q)}$ is different to before as the left-hand branch of the convolution may be read non-deterministically by the nested-word automaton. We first specify a nested-word language $\mathcal{L}^+_{i\vec{a},b,(p,q)}$ such that we want

$$\mathcal{L}^+_{i\vec{a},b,(p,q)} = \{\ w^{\curvearrowleft E} \in \mathfrak{D}_{\mathcal{B}_1^s,\ldots,\mathcal{B}_k^s,\wp(\mathcal{C}_1),\ldots,\wp(\mathcal{C}_k),{\mathcal{B}_1^{\mathcal{C}_1}}^{\Cap},\ldots,{\mathcal{B}_k^{\mathcal{C}_1}}^{\Cap}}(\mathfrak{V}(\mathcal{L}(\mathcal{A})))\ :\ \ulcorner w^{\curvearrowleft E} \urcorner \in \mathcal{L}_{i\vec{a},b,(p,q)}\ \}$$

It is defined to consist of all words of the form $\vec{a}\vec{b}w$ where $\boldsymbol{symb}(\vec{b}) = b$ and $\mathcal{B}_i^{\mathcal{C}_i}$ could recognise $\vec{b}w$:

- starting in some state (r,p) where $\mathcal{B}_i$ would be able to branch-left off the spine into state r from $\boldsymbol{st}_{\mathcal{B}_i^s}(\vec{a})$.
- treating all positions that are not pointer sources in $\vec{a}w$ but for which $\boldsymbol{st}^{\boldsymbol{pr}}_{\mathcal{B}_i^s} \neq \bot$ as being the source of a pointer with state $\boldsymbol{st}^{\boldsymbol{pr}}_{\mathcal{B}_i^s}$ at its target.

Given that $\vec{a}w$ is the suffix of a nested-word in the domain, the last item above ensures we give $\mathcal{B}_i$ the correct state at the target of pointers where the target lies outside of the suffix.

It is easy to derive a nested-word automaton from $\mathcal{B}_i^{\mathcal{C}_i}$ that would recognise $\mathcal{L}^+_{i\vec{a},(p,q)}$. However, we wish to present this in terms of the language of summaries $\mathcal{L}_{i\vec{a},b,(p,q)}$—*i.e.* we

need to be able to recognise those words of the for $\ulcorner\vec{a}\vec{b}w\urcorner$. We can do this with a finite-word automaton that reads $\ulcorner\vec{a}w\urcorner$ simulating $\mathcal{B}_i^{\mathcal{C}_i}$ starting in some state (r,p) as specified by the first point above. The second point above can also be achieved using a non-nested automaton since there is no need to actually look up a state at the target of a pointer in order to achieve this. So the only problem is simulating $\mathcal{B}_i^{\mathcal{C}_i}$ on pointers whose source and target both lie within the suffix $\vec{a}\vec{b}w$. This is enabled by looking at $\boldsymbol{st}_{\mathcal{B}_i{}^{\mathcal{C}_i{}^{\Uparrow}}}$ in $\ulcorner\vec{a}\vec{b}w\urcorner$. Suppose

there is a segment $\cdots ab\cdots$ in the summary that originates from $\cdots\ a\ \cdots\ b\ \cdots$. The state $\boldsymbol{st}_{\mathcal{B}_i{}^{\mathcal{C}_i{}^{\Uparrow}}}(b)$ will specify pairs $((q_1,q_2),(q_1',q_2'))$ such that starting at a in state (q_1,q_2), $\mathcal{B}_i^{\mathcal{C}_i}$ could begin in state (q_1,q_2) and end in state (q_1',q_2') at b. Our finite automaton in reading the summary is thus able to recall its state at a when reading b and compute a possible state of $\mathcal{B}_i^{\mathcal{C}_i}$ to simulate at b. Thus $\mathcal{L}_{i\vec{a},(p,q)}$ is non-nested regular as required.

□

Remark 7.3. The fact that we will be able to go from rewrite systems to (dendr)isophilic structures and back again to rewrite systems means that the rewrite systems constructed in the proof above can be viewed as a kind of 'normal form'. In particular it is always possible to convert a suffix rewrite system to one generating the same graph whose rewrite rules always match a nested-word of the form aw_1 on the left-hand-side to one of the form aw_2, where w_1 and w_2 have no common prefix. In particular this tells us that it is in general impossible to take the *'symmetric closure'* of a sum-rat graph, where the symmetric closure of a graph adds an edge labelled $\bar{a}$ oriented in the opposite direction to an edge labelled a. After all, if it were possible and all sum-rat rules $a : \mathcal{L}_1 \longrightarrow \mathcal{L}_1'$ and $\bar{a} : \mathcal{L}_2 \longrightarrow \mathcal{L}_2'$ were put into such a normal form, it would be possible to construct a rat-rat system generating the original graph with rules $a : \mathcal{L}_2' \longrightarrow \mathcal{L}_1'$. In fact this allows us to conclude that if the symmetric closure of a given 2-CPDA graph is also a 2-CPDA graph, then the original graph must also be a 2-PDA graph. But since 2-CPDA graphs include those that are not 2-PDA [16], it must be the case that 2-CPDA graphs can be 'inherently asymmetric'.

7.2. **From Automata to Isophilic Structures.** This is inspired by Kartzow's work on 2-CPDA and tree-automaticity [17]. We reformulate it to show the progression from words to nested-words to nested-trees corresponds to the journey from order-1 to order-3 automata.

Given an automaton $\mathcal{A}$ we consider its *stack graph*, which represents its behaviour using edges annotated with pairs of control-states and whose nodes consist only of stacks.

Definition 7.4. Let $\mathcal{A}$ be an n-(C)PDA for some $n \in \mathbb{N}$. Its *stack graph* $\mathcal{G}_s(\mathcal{A})$ has *stacks* belonging to reachable configurations of $\mathcal{A}$ as nodes and a directed relation $\boldsymbol{R_{p,a,q}}$ between stacks s and s' whenever there is an a-edge from (p,s) to (q,s') in $\mathcal{G}(\mathcal{A})$. The *ϵ-closed stack-graph* $\mathcal{G}_s^{\epsilon}(\mathcal{A})$ is defined in the same way except there is a directed relation $\boldsymbol{R_{p,a,q}}$ between stacks s and s' whenever there is an a-edge from (p,s) to (q,s') in $\mathcal{G}^{\epsilon}(\mathcal{A})$ and the nodes of the graph are the stacks belonging to configurations in the ϵ-closure.

We are naturally able to make both $\mathcal{G}_s(\mathcal{A})$ and $\mathcal{G}_s^{\epsilon}(\mathcal{A})$ a $\mathcal{Q}$-decorated structure, assigning pairs of control-states of $\mathcal{A}$ to each relation. In both cases we set $\mathcal{Q}(\boldsymbol{R_{p,a,q}}) := (p,q)$.

It is easy to see that for any order-1 automaton $\mathcal{A}_1$ the graph $\mathcal{G}_s(\mathcal{A})$ is flat-isophilic. If Γ is the stack alphabet, then stacks can be represented by finite-words in Γ^*. Since an individual stack operation changes the height of the stack by at most one, convolutions

representing single transitions have branches of length at most one eminating from the tip of the trunk and so can easily be recognised by a finite tree automaton. Given the judicious choice of $\mathcal{Q}$-decoration, Theorem 6.12 (the proof applied to flat-isophilic gives flat-isophilic) then gives us a flat-isophilic representation of $\mathcal{G}^{\epsilon}_{s}(\mathcal{A})$—we first get a representation of ϵ^* labelled paths using the theorem, and can then concatenate a single non-ϵ edge. This is the same as the previously known fact that ϵ-closures of 1-PDA graphs coincide with rational prefix rewrite systems [21]. We express this result as:

Lemma 7.5. *Let $\mathcal{A}$ be a 1-PDA. Then $\mathcal{G}^{\epsilon}_{s}(\mathcal{A})$ is flat-isophilic.*

Using derivatives and flat-isophilic chains we can continue the hierarchy with:

Lemma 7.6. *Let $\mathcal{A}$ be a 2-PDA, then $\mathcal{G}^{\epsilon}_{s}(\mathcal{A})$ is isophilic. Let $\mathcal{A}$ be a 2-CPDA, then $\mathcal{G}^{\epsilon}_{s}(\mathcal{A})$ is dendrisophilic.*

Proof. We may assume that $\mathcal{A}^{\uparrow}$ is *slow* (recall that this means it does not perform order-2 operations when making ϵ-transitions); since isophilic and dendrisophilic structures are closed under reachability (Theorem 6.12), we can always extend to the case when the automata are not slow.

First consider the case when $\mathcal{A}$ is a 2-PDA. Then $\partial(\mathcal{A})$ is a 1-PDA and so by Lemma 7.5 it must be the case that $\mathcal{G}_{s}(\partial(\mathcal{A}))$ is isophilic. We add the relation: $\boldsymbol{R_{q,r^p(\epsilon+\Sigma)^*,q'}}$ relating (q,s) and (q',s') such that $(q,s)\boldsymbol{r}_{r^p(\epsilon+\Sigma)^*}(q',s')$ where r ranges over all non-ϵ edge labels (so r^p ranges over every a^p in the derivative). Give this relation the $\mathcal{Q}$-decoration (q,q').

Let us further add a unary predicate $\boldsymbol{c_0 R_{\epsilon^*,q}}$ that is satisfied by precisely those nodes s such that (q,s) is reachable from the initial configuration of $\partial(\mathcal{A})$ via an ϵ^*-labelled path. All of these additions preserve flat-isophilicity by Theorem 6.12. Let us give the graph modifed to include *only* $\boldsymbol{R_{q,r^p(\epsilon+\Sigma)^*,q'}}$ and $\boldsymbol{c_0 R_{\epsilon^*,q}}$ the name $\mathcal{G}^{\epsilon}_{s}(\partial(\mathcal{A}))^{+}$.

We can represent precisely the nodes of $\mathcal{G}^{\epsilon}_{s}(\mathcal{A})$ by the elements of the set:

$$S \;:=\; \{\, C \in \boldsymbol{Ch_{\blacktriangleleft}}(\mathcal{G}^{\epsilon}_{s}(\partial(\mathcal{A}))^{+}) \;:\; \text{the initial item in } C \text{ satisfies } \boldsymbol{c_0 R_{\epsilon^*,q}} \text{ and has decoration } q \text{ for some } q \,\}$$

This set is over approximated by $\boldsymbol{Ch_{\blacktriangleleft}}(\mathcal{G}^{\epsilon}_{s}(\partial(\mathcal{A}))^{+})$. Note how the automaton recognising a chain in $\boldsymbol{Ch_{\blacktriangleleft}}(\mathcal{G}^{\epsilon}_{s}(\partial(\mathcal{A}))^{+})$ can be adapted to recognise S; we simply send down the left-most branch of the chain a copy of the automaton for the unary predicate $\boldsymbol{c_0 R_{\epsilon^*,q}}$ for each state q and require that the instance for a state q accepts where q is used as the decoration for the first element of the chain. In order to show that $\mathcal{G}^{\epsilon}_{s}(\mathcal{A})$ is isophilic, it thus suffices to show that the graph $\boldsymbol{Ch_{\blacktriangleleft}}(\mathcal{G}^{\epsilon}_{s}(\partial(\mathcal{A}))^{+})$, together with the relations for stack operations on the 2-PDA stacks encoded as chains, is isophilic.

We appeal to Lemma 6.15. Recall that we are considering the slow version of a 2-PDA and so we are not treating any pop_2 or $push_2$ operations as ϵ-transitions. First observe that order-1 operations only affect the top stack—*i.e.* the final element of the stack in the chain representation thereof. We can therefore represent these by flat-isophilic relations of the form $R_{q,\epsilon^*a,q'}$ for each $a \in \Sigma$ relating stacks s and s' when $(q,s)\boldsymbol{r}_{\epsilon^*a}(q',s')$ and assigning these the decoration (q,q'). The relation $R_{q,\epsilon^*a,q'}{}_{\blacktriangleleft}$ on chains then does the job.

For a pop_2 we just need to discard the final element of the chain. We can have a relation $R_{q,a\downarrow,q'}$ that relates a stack s to s' just in case a pop_2 could be performed by $\mathcal{A}$ via an edge a when in control-state q with top stack symbol $top_1(s)$ and transitioning into control-state q' (stack s' is ignored). We can then represent pop_2 operations by relations of the form $R_{q,a\downarrow,q'\,pop}$, assigning $\mathcal{Q}_{pop}(R_{q,a\downarrow,q'}) := (q,q')$.

For $push_2$ we can have a relation $R_{q,a\uparrow,q'}$ that relates a stack s to s' just in case a $push_2$ could be performed by $\mathcal{A}$ via an edge a when in control-state q with top stack symbol $top_1(s)$ and transitioning into control-state q' *and additionally* $s = s'$. (Equality is flat-isophilic so this is possible). We then represent $push_2$ operations by relations of the form $R_{q,a\uparrow,q'push}$, assigning $\mathcal{Q}_{push}(R_{q,a\uparrow,q'}) := (q, q')$. Thus $\mathcal{G}^{\epsilon}_{s}(\mathcal{A})$ is isophilic by Theorem 6.17.

When $\mathcal{A}$ is a 2-CPDA it is sufficient to show that $\mathcal{G}^{\epsilon}_{s}(\mathsf{Trail}(\mathcal{A}))$ is dendrisophilic. We do this in the same way as above, working with $\partial(\mathsf{Trail}(\mathcal{A}))$. Item two of Lemma 3.7 ensures that equality of chains from the derivative implies equality of the stacks being represented (in the absence of links). Due to the third item of Lemma 3.7 this means that we can extend the proof to work with *collapse* by recognising such transitions by predicates of the form $(P_1, P_2)_{panic}$ where P_1 checks that the top element of the starting stack is such that the transition function permits *collapse* and P_2 is the universal predicate, which should be satisfied by all stacks. The use of *panic* means that we obtain a *dendrisophilic* structure. □

We can go from a representation of $\mathcal{G}^{\epsilon}_{s}(\mathcal{A})$ to a representation of $\mathcal{G}^{\epsilon}(\mathcal{A})$ by simply adding to the right-most leaf of a representation of the stack a control-state of $\mathcal{A}$. Each element of the domain then represents a *configuration* rather than just a stack. We can then construct a representation of an a-labelled edge R_a by constructing an automaton recognising the same binary convolutions as $R_{q,a,q'}$ in the stack-graph whenever the first configuration has right-most-leaf labelled q and the second has right-most-leaf labelled q'. Thus we get:

Lemma 7.7. $\mathbf{Pd}_{\epsilon 2} \subseteq \boldsymbol{Iso_2}$ *and* $\mathbf{Pd}^{\mathcal{C}}_{\epsilon 2} \subseteq \boldsymbol{dIso_2}$. *This completes the proof of Theorem 3.8.*

8. First-Order Logic on 3_2-CPDA Graphs

Since 2-CPDA graphs are dendrisophilic, we can extend Kartzow's [17] first-order decidability result to the ϵ-closures of 2-CPDA using Theorem 6.13:

Theorem 8.1. *Let $\mathcal{A}$ be a 2-CPDA. Then $\mathcal{G}^{\epsilon}(\mathcal{A})$ has decidable* $\boldsymbol{FO(TC[\Delta_0])}$.

In fact this is, in some sense, optimal ($\boldsymbol{FO(TC[\Delta_1])}$, which allows for the transitive closure of properties expressible as both a universal and existential formula (Δ_1)) is undecidable on such graphs [6, 9, 7]. The theorem also has applications to proving some decidability results for the existential fragment of first-order logic on restricted CPDA of higher-orders [6, 7]. A secondary result gives new information beyond order-2:

Theorem 8.2. *Let $\mathcal{A}$ be a* slow 3_2*-CPDA. Then $\mathcal{G}^{\epsilon}(\mathcal{A})$ has a decidable first-order theory.*

This lies at the fringe of decidability [9]; in particular one cannot drop slowness.

We begin by showing that the tree-isophilic and tree-nondisophilic structures subsume the ϵ-closures of 3-PDA and 3_2-CPDA respectively (*without* any slowness assumption). Restricting this to slow 3_2-CPDA would be sufficient for the purposes of the theorem above, however it is interesting in itself that the technique introduced in the previous section can continue to build the hierarchy up to order-3 for unrestricted ϵ-closure.

Lemma 8.3. *Let $\mathcal{A}$ be a 3-PDA, then $\mathcal{G}^{\epsilon}(\mathcal{A})$ is tree-isophilic. Let $\mathcal{A}$ be a 3_2-CPDA, then $\mathcal{G}^{\epsilon}(\mathcal{A})$ is tree-nondisophilic.*

Proof. As before we need only prove that $\mathcal{G}^{\epsilon}_{s}(\mathcal{A})$ is tree-isophilic (*resp.* tree-nondisophilic) if $\mathcal{A}$ is a 3-PDA (*resp.* 3_2-CPDA). If we were to restrict $\mathcal{A}$ to being a slow 3-PDA (*resp.* 3_2-CPDA) then we could perform exactly the same construction as for the proof of the 2-PDA

component of Lemma 7.6 by constructing isophilic (*resp.* dendrisophilic) chains over an analogous modification of $\partial(\mathcal{A})$. The fact (Theorem 6.18) that graphs from isophilic (*resp.* dendrisophilic) chains are tree-isophilic (*resp.* tree-nondisophilic) then yields the result.

In order to extend the construction to the slow case we use $\mathcal{A}^{\uparrow}$, recalling that $\mathcal{G}^{\epsilon}(\mathcal{A}) \cong \mathcal{G}^{\epsilon}(\mathcal{A}^{\uparrow})$. Observe that the special property enjoyed by runs of $\mathcal{A}^{\uparrow}$ means that if it has a sequence of ϵ-transitions $(q,s) \xrightarrow{\epsilon^*} (q',s')$, then it must have a run of the following form:

$$\begin{aligned}
&(q, [\ s_1\ s_2\ \cdots\ s_{m-1}\ s_m\ ] =: s) \xrightarrow{\epsilon^*} (q_1, [\ s_1\ s_2\ \cdots\ s_{m-1}\ ]) \xrightarrow{\epsilon^*} \cdots \\
&\xrightarrow{\epsilon^*} (q_{m-l}, [\ s_1\ s_2\ \cdots\ s_{l-1}\ s_l\ ]) \xrightarrow{\epsilon^*} (q_{m-l+1}, [\ s_1\ s_2\ \cdots\ s_{l-1}\ s'_l\ ]) \xrightarrow{\epsilon^*} (q_{m-l+2}, [\ s_1\ s_2\ \cdots\ s_{l-1}\ s'_l\ s'_{l+1}\ ]) \\
&\xrightarrow{\epsilon^*} \cdots \xrightarrow{\epsilon^*} (q_{m-l+(m'-l)+1}, [\ s_1\ s_2\ \cdots\ s_{l-1}\ s'_l\ s'_{l+1}\ \cdots\ s'_{m'}\ ] =: s')
\end{aligned}$$

where the first half never uses $push_3$ (but does use pop_3) and the second half never uses pop_3 (but does use $push_3$). This form of run is called a *bounce* with the first half ending in the final pop_3 operation called a *fall* and the second half a *climb*. A climb can be represented in the graph induced from a chain in exactly the same manner as reachability from the origin is defined (reachability from the origin is a climb from the empty stack). This will involve a path-nested automaton for the 3-PDA (isophilic chain) but an unrestricted nested-tree automaton for the 3-CPDA (dendrisophilic chain).

To represent a fall, note that reachability in $\partial(\mathcal{A}^{\uparrow})$ allows us to construct a nested-word automaton recognising a predicate $\downarrow_{q,q'}$, for each pair of control-states q, q', which recognises (representations of) order-2 stacks s such that from (q,s) $\mathcal{A}^{\uparrow}$ could perform order-2 transitions followed by a single pop_3 into control-state q'. A path-nested automaton $\mathcal{B}^{\mathcal{C}}$ can be used in both cases to recognise binary convolutions of an isophilic/dendrisophilic chain representing an order-3 stack together with one representing a stack obtained via a fall. $\mathcal{B}$ runs the determinised automaton for every $\downarrow_{q,q'}$ along each branch and $\mathcal{C}$ uses this to check for the existence of the fall.

By combining the automata for falls and climbs one can recognise binary convolutions for $\xrightarrow{\epsilon^*}$ in $\mathcal{A}^{\uparrow}$ and from this obtain a representation of the unrestricted ϵ-closure. □

By Theorem 6.8, the following is sufficient to establish Theorem 8.2

Lemma 8.4. *If $\mathcal{A}$ is a* slow 3_2*-CPDA, then $\mathcal{G}^{\epsilon}(\mathcal{A})$ is* compact *nondisophilic.*

Proof. We just need to show that $\mathcal{G}^{\epsilon}_s(\mathcal{A})$ is compact nondisophilic. Since only the two right-most elements of the dendrisophilic chain need be compared in order to establish whether a pop_3 or $push_3$ relation holds (under the assumption that all other branches of the two chains are the same), all transition relations must be compact. We also modify the representations of stacks slightly. At the tip of ith element s_i of the chain we must record a set S_i. The set S_i contains precisely those control-states belonging to reachable configurations associated with the stack encoded by the first i elements of the chain. Note that $S_{i+1} = \{\ q' \in Q\ :\ (q,s_i)\boldsymbol{r}_{r^p(\epsilon+\Sigma)^*}(q',s_{i+1})\ \}(*)$ and so each step in the chain remains dendrisophilic and so the chain as a whole is still tree-nondisophilic. Moreover, since each S_i decoration is uniquely determined by the chain, equality is preserved.

Given $a \in \mathcal{G}^{\epsilon}_s(\mathcal{A})$, by slowness any $b \in B_r(a)$ (for any r) must be represented by a tree coinciding with the tree for a on every branch that is not amongst its right-most r branches. Moreover, using (*), a nested-tree automaton can tell whether any tree agreeing with a on all but its right-most r branches also belongs to a by examining the S_i on the $(r+1)$th

branch from the right of b (which it has in common with a) together with its remaining right-most r branches. Thus we can construct automata acting on a bounded number of branches recognising a suitable D_r^k. Thus $\mathcal{G}_s^\epsilon(\mathcal{A})$ is compact nondisophilic. □

Acknowledgements

We thank the anonymous reviewers of our ICALP paper [8] for detailed and very helpful comments.

References

[1] R. Alur, S. Chaudhuri, and P. Madhusudan. Languages of nested trees. In *CAV*, pages 329–342. Springer, 2006.
[2] R. Alur and P. Madhusudan. Visibly Pushdown Languages. In *STOC*, pages 202–211. ACM, 2004.
[3] R. Alur and P. Madhusudan. Adding nesting structure to words. *Journal of the ACM (JACM)*, 56(3):1–43, 2009.
[4] M. Arenas, P. Barcelo, and L. Libkin. Regular Languages of Nested Words: Fixed Points, Automata, and Synchronization. *Theory Comput. Syst.*, 49(3):639—-670, 2011.
[5] A. Blumensath and E. Gradel. Automatic structures. In *LICS*, pages 51–62. IEEE Comput. Soc, 2000.
[6] C. Broadbent. *On Collapsible Pushdown Automata, their Graphs and the Power of Links*. PhD thesis, 2011.
[7] C. Broadbent. On First-Order Logic and CPDA Graphs. *Submitted to Theory of Computing Systems*, (STACS Special Issue), 2012. URL: http://mjolnir.cs.ox.ac.uk/∼chrb/focpda.ps
[8] C. Broadbent. Prefix Rewriting for Nested-Words and Collapsible Pushdown Automata. In *ICALP*, 2012.
[9] C. Broadbent. The Limits of Decidability for First Order Logic on CPDA Graphs. In *STACS*, 2012.
[10] A. Carayol. Regular Sets of Higher-Order Pushdown Stacks. In *Proc. MFCS*, volume 3618 of *LNCS*, pages 168–179. Springer, 2005.
[11] A. Carayol and S. Wöhrle. The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *FSTTCS*, pages 112–123. Springer, 2003.
[12] D. Caucal. On infinite transition graphs having a decidable monadic theory. In *ICALP*, volume 6317, pages 194–205. Springer, 1996.
[13] D. Caucal. On Infinite Terms having a Decidable Monadic Theory. In *Mathematical Foundations of Computer Science, 2002 : 27th Internation Symposium*, volume 24, pages 165–176. Springer, 2002.
[14] C. Delhommé. Automaticité des ordinaux et des graphes homogčnes. *Comptes Rendus Mathematique*, 339(1):5–10, 2004.
[15] H. Gaifman. On Local and Non-Local Properties. In J. Stern, editor, *Proceedings of the Herbrand Symposium*, volume 107 of *Studies in Logic and the Foundations of Mathematics*, pages 105–135. Elsevier, 1982.
[16] M. Hague, A. S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible Pushdown Automata and Recursion Schemes. In *LICS*. IEEE Computer Society, 2008.
[17] A. Kartzow. Collapsible Pushdown Graphs of Level 2 are Tree-Automatic. In *STACS*, pages 501–512, 2010.
[18] B. Khoussainov and A. Nerode. Automatic presentations of structures. In D. Leivant, editor, *Logic and Computational Complexity*, volume 960 of *Lecture Notes in Computer Science*, pages 367–392. Springer Berlin / Heidelberg, 1995.
[19] T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-Order Pushdown Trees are Easy. In *Proc. FoSSaCS*, volume 2303 of *LNCS*, pages 205–222. Springer, 2002.
[20] T. Knapik, D. Niwinski, P. Urzyczyn, and I. Walukiewicz. Unsafe Grammars and Panic Automata. In *ICALP*, volume 3580, pages 1450–1461, 2005.
[21] C. Stirling. Decidability of Bisimulation Equivalence for Pushdown Processes, 2000.