Submitted to the Symposium on Theoretical Aspects of Computer Science
www.stacs-conf.org

COLLAPSIBLE PUSHDOWN GRAPHS OF LEVEL 2 ARE TREE-AUTOMATIC

ALEXANDER KARTZOW [1]

[1] TU Darmstadt, Fachbereich Mathematik, Schlossgartenstr. 7, 64289 Darmstadt

ABSTRACT. We show that graphs generated by collapsible pushdown systems of level 2 are tree-automatic. Even when we allow ε-contractions and add a reachability predicate for pairs of configurations, the structures remain tree-automatic. Hence, their FO + REACH-theories are decidable. As a corollary, we obtain the tree-automaticity of the second level of the Caucal-hierarchy.

1. Introduction

Higher-order pushdown systems were first introduced by Maslov [10, 11] as accepting devices for word languages. Later, Knapik et al. [8] studied them as generators for trees. They obtained an equi-expressivity result for higher-order pushdown systems and for higher-order recursion schemes that satisfy the constraint of *safety*, which is a rather unnatural syntactic condition. Recently, Hague et al. [6] introduced collapsible pushdown systems as extensions of higher-order pushdown systems and proved that these are equi-expressive to higher-order recursion schemes as a method for generating trees.

Both – higher-order and collapsible pushdown systems – also form interesting devices for generating graphs. Carayol and Wöhrle [3] showed that the graphs generated by higher-order pushdown systems of level l^1 coincide with the graphs in the l-th level of the Caucal-hierarchy, a class of graphs introduced by Caucal [4]. Every level of this hierarchy is obtained from the former level by applying graph-unfoldings and MSO-interpretations. Both operations preserve the decidability of the MSO-theory whence the Caucal-hierarchy forms a rather big class of graphs with decidable MSO-theories. If we use collapsible pushdown systems as generators for graphs we obtain a different situation. Hague et al. showed that even the second level of the hierarchy contains a graph with undecidable MSO-theory. But they showed the decidability of the $L\mu$-theories of all graphs in the hierarchy ($L\mu$ denotes the modal μ-calculus). This turns graphs generated by collapsible pushdown systems into an interesting class from a model theoretic point of view. There are few natural classes that share these properties. In fact, the author only knows one further example, viz. nested pushdown trees. Alur et al.[1] introduced these graphs for $L\mu$ model checking purposes. We proved in [7] that nested pushdown trees also have decidable first-order theories. We

Key words and phrases: tree-automatic structures, collapsible pushdown graphs, first-order decidability.

[1] The graph generated by a higher-order pushdown system is the graph of reachable configurations with ε-contraction.

gave an effective model checking algorithm using pumping techniques, but we also proved that nested pushdown trees are tree-automatic structures. Tree-automatic structures were introduced by Blumensath [2]. These structures enjoy decidable first-order theories due to the good computational behaviour of finite automata on trees.

In this paper, we are going to extend our previous result to the second level of the collapsible pushdown hierarchy. All graphs of the second level are tree-automatic. This subsumes our previous result as nested pushdown trees are first-order interpretable in collapsible pushdown graphs of level two. Furthermore we show that collapsible pushdown graphs of level 2 expanded by a reachability predicate, i.e., by the binary relation which contains all pairs of configurations such that there is a path from the first to the second configuration, are still tree-automatic. Hence first-order logic extended by reachability predicates is decidable on level 2 collapsible pushdown graphs.

Here is an outline of the paper. The next section introduces some basic notation, in particular with respect to tree-automatic structures. Section 3 then briefly reviews the definitions and our notation of collapsible pushdown systems and graphs. We explain the translation of configurations into trees in Section 4. We prove in Section 5 that this translation provides a tree-automatic representation of collapsible pushdown graphs of level 2. At the end of this section we state some extensions of this result concerning ε-contractions and first-order logic extended by reachability predicates. The last section contains some concluding remarks about questions arising from our results.

2. Preliminaries

We write MSO for monadic second order logic and FO for first-order logic.
For words $w_1, w_2 \in \Sigma^*$, we write $w_1 \sqcap w_2$ for the greatest common prefix of w_1 and w_2.
A Σ-*labelled tree* is a function $T : D \to \Sigma$ such that $D \subseteq \{0,1\}^*$ is finite and closed under prefixes.
For $d \in D$ we denote the *subtree rooted at* d by T_d. We denote the depth of the tree T by $\mathrm{dp}(T) := \max(|t| : t \in \mathrm{dom}(T))$.

Sometimes it is useful to define trees inductively by describing their left and right subtree. For this purpose we fix the following notation. Let $\hat{T}_0$ and $\hat{T}_1$ be Σ-labelled trees and $\sigma \in \Sigma$. Then we write $T := \sigma(\hat{T}_0, \hat{T}_1)$ for the Σ-labelled tree T with the following three properties

1. $T(\varepsilon) = \sigma$, 2. $T_0 = \hat{T}_0$, and 3. $T_1 = \hat{T}_1$.

In the rest of this section, we briefly present the notion of a tree-automatic structure as introduced by Blumensath [2].

The *convolution* of two Σ-labelled trees c_1 and c_2 is given by a function

$$c_1 \otimes c_2 : \mathrm{dom}(c_1) \cup \mathrm{dom}(c_2) \to (\Sigma \cup \{\Box\})^2$$

where $\Box$ represents undefined elements, and

$$(c_1 \otimes c_2)(d) := \begin{cases} (c_1(d), c_2(d)) & \text{if } d \in \mathrm{dom}(c_1) \cap \mathrm{dom}(c_2) \\ (c_1(d), \Box) & \text{if } d \in \mathrm{dom}(c_1) \setminus \mathrm{dom}(c_2) \\ (\Box, c_2(d)) & \text{if } d \in \mathrm{dom}(c_2) \setminus \mathrm{dom}(c_1) \end{cases}$$

With tree-automaton we mean a finite automaton that labels a finite tree in a bottom-up manner.

Definition 2.1. A structure $B = (B, E_1, E_2, \dots, E_n)$ with binary relations E_i is *tree-automatic* if there are tree-automata $A_B, A_{E_1}, A_{E_2}, \dots, A_{E_n}$ such that for the language accepted by A_B, denoted by $L(A_B)$ the following hold:

(1) There is a bijection $f : L(A_B) \to B$.
(2) For $c_1, c_2 \in C$, the automaton A_{E_i} accepts $c_1 \otimes c_2$ if and only if $(f(c_1), f(c_2)) \in E_i$.

Tree-automatic structures form a nice class because automata theoretic techniques may be used to decide first-order formulas on these structures:

Lemma 2.2 ([2])**.** *If B is tree-automatic, then its first-order theory is decidable.*

In the following, we use a well known characterisation of regular classes of trees in terms of MSO-definability.

Lemma 2.3 ([12], [5])**.** *For a set $\mathbb{T}$ of finite Σ-labelled trees, there is a tree automaton recognising $\mathbb{T}$ if and only if $\mathbb{T}$ is* MSO *definable.*

3. Definition of Collapsible Pushdown Graphs (CPG)

In this section, we define our notation of collapsible pushdown systems. For a more readable introduction, we refer the reader to [6].

3.1. Collapsible Pushdown Stacks

First, we provide some terminology concerning stacks of (collapsible) higher-order pushdown systems. We write Σ^{*1} for Σ^* and Σ^{+1} for Σ^+. Then we define inductively $\Sigma^{*(n+1)} := (\Sigma^{*n})^*$ and $\Sigma^{+(n+1)} := (\Sigma^{+n})^+$. We call an element of Σ^{*n} a level n word.

Let us fix a level $n+1$ word $s \in \Sigma^{*(n+1)}$. This word s consists of an ordered list $w_1, w_2, \dots, w_m$ with $w_i \in \Sigma^{*n}$. If we want to state this list of level n words explicitly, we separate them by colons writing $s = w_1 : w_2 : \dots : w_m$. By $|s|$ we denote the number of level n words s consists of, i.e., $|s| = m$.

If there is another word $s' \in \Sigma^{*(n+1)}$ with $s' = w'_1 : w'_2 : \dots w'_m$ then we write $s : s'$ for the concatenation $w_1 : w_2 : \dots w_m : w'_1 : w'_2 : \dots : w'_m$.

If $s \in \Sigma^{*n}$, we write $[s]$ for the $n+1$ word that only consists of a list of one n word which is s. We regularly omit the brackets.

A level n collapsible pushdown stack is a special element of $(\Sigma \times \{1, 2, \dots, n\} \times \mathbb{N})^{+n}$ that is generated by certain stack operations from an initial stack which we introduce in the following definitions. The natural numbers following the stack symbol represent the so-called collapse link: every element in a collapsible-pushdown stack has a pointer to some substack and applying the so-called collapse operation returns the substack where the topmost symbol of the stack points to. Here, the first number denotes the link-level. If it is k then the collapse-operation operates on the topmost level k word of the stack. The latter number indicates how the resulting topmost level k stack looks like: it is the number of level $k-1$ stacks the resulting topmost level k stack consists of.

The initial level 1 stack is $\bot_1 := (\bot, 1, 0)$. For the higher levels, we set inductively $\bot_n := [\bot_{n-1}]$.

Now we define the stack-operations. For $s = w_1 : w_2 : \dots : w_n \in (\Sigma \times \{1, 2, \dots l\} \times \mathbb{N})^{+l}$, we define the following stack operations for $1 \leq k \leq l$, and $\sigma \in \Sigma \setminus \{\bot\}$:

- The *topmost level $k-1$ word of s*, $\mathrm{top}_k(s) := \begin{cases} w_n & \text{if } k=l \\ \mathrm{top}_k(w_n) & \text{otherwise}\end{cases}$
- For $\mathrm{top}_1(s) = (\sigma, i, j) \in \Sigma \times \{1, 2, \dots, l\} \times \mathbb{N}$, we define the *topmost symbol* $\mathrm{Sym}(s) := \sigma$, the *collapse-level of the topmost element* $\mathrm{CLvl}(s) := i$, and the *collapse-link of the topmost element* $\mathrm{CLnk}(s) := j$.

Furthermore, for $1 \leq k \leq l$ and $2 \leq j \leq l$ let

$$\mathrm{clone}_j(s) := \begin{cases} w_1 : w_2 : \dots : w_{n-1} : w_n : w_n & \text{if } j = l \geq 2 \\ w_1 : w_2 : \dots : w_{n-1} : \mathrm{clone}_j(w_n) & \text{otherwise}\end{cases}$$

$$\mathrm{push}_{\sigma,k,m}(s) := \begin{cases} s(\sigma, k, m) & \text{if } l = 1 \\ w_1 : w_2 : \dots : w_{n-1} : \mathrm{push}_{\sigma,k,m}(w_n) & \text{otherwise}\end{cases}$$

$$\mathrm{push}_{\sigma,k}(s) := \begin{cases} \mathrm{push}_{\sigma,k,n}(s) & \text{if } k = l = 1 \\ \mathrm{push}_{\sigma,k,n-1}(s) & \text{if } k = l \geq 2 \\ w_1 : w_2 : \dots : w_{n-1} : \mathrm{push}_{\sigma,k}(w_n) & \text{otherwise}\end{cases}$$

$$\mathrm{pop}_k(s) := \begin{cases} w_1 : w_2 : \dots : w_{n-1} : \mathrm{pop}_k(w_n) & \text{if } k < l \\ w_1 : w_2 : \dots : w_{n-1} & \text{if } k = l, n > 1 \\ \text{undefined} & \text{otherwise, i.e.,} k = l, n = 1\end{cases}$$

$$\mathrm{collapse}(s) := \begin{cases} w_1 : w_2 : \dots : w_m & \text{if } \mathrm{CLvl}(s) = l, \mathrm{CLnk}(s) = m > 0 \\ w_1 : w_2 : \dots : w_{n-1} : \mathrm{collapse}(w_n) & \text{if } \mathrm{CLvl}(s) < l\end{cases}$$

$\mathrm{OP}_l := \{(\mathrm{push}_{\sigma,k})_{\sigma\in\Sigma, k\leq l}, (\mathrm{clone}_k)_{2\leq k\leq l}, (\mathrm{pop}_k)_{1\leq k\leq l}, \mathrm{collapse}\}$ is the *set of level l-operations*. The *set of level l stacks*, $\mathrm{Stck}_l(\Sigma)$, is the smallest set that contains $\bot_l$ and is closed under OP_l.

Note that collapse- and pop_k-operations are only allowed if the resulting stack is nonempty. This avoids the special treatment of empty stacks. Furthermore, any collapse that works on level 1 performs exactly one pop_1-operation. On the other hand a collapse that works on higher levels performs at least one pop_i-operation, but it may perform several of them. The easiest example is by just performing the sequence clone_2, $\mathrm{push}_{\sigma,2}$, clone_2, and collapse. This sequence first creates a level 2 stack that contains 3 level 1 stacks and then performs the collapse and ends in the initial stack again. From this example, one sees that clone_k-operations are responsible for the fact that a level k collapse-operation may remove more than one level $(k-1)$-word from the stack.

For $s, s' \in \mathrm{Stck}_l(\Sigma)$, we say s' is a substack of s if there are $n_i \in \mathbb{N}$ for $1 \leq i \leq l$ such that $s' = {\mathrm{pop}_1}^{n_1}({\mathrm{pop}_2}^{n_2}(\dots({\mathrm{pop}_l}^{n_l}(s))))$. We write $s' \leq s$ if s' is a substack of s.

3.2. Collapsible Pushdown Systems and Collapsible Pushdown Graphs

In this section we introduce collapsible pushdown systems and graphs which are analogs of pushdown systems and pushdown graphs using a collapsible pushdown stack instead of an ordinary stack.

Definition 3.1. A *collapsible pushdown system* of level l (l-CPS) is given as a tuple $S = (\Sigma, Q, \Delta, q_0)$ where Σ is a finite stack alphabet with $\bot \in \Sigma$, Q a finite state set, $q_0 \in Q$ the initial state, and $\Delta \subseteq Q \times \Sigma \times Q \times \mathrm{OP}_l$ the transition relation.

A level l *configuration* is a pair (q, s) where $q \in Q$ and $s \in \mathrm{Stck}_l(\Sigma)$. For $q_1, q_2 \in Q$ and $s, t \in (\Sigma \times \{1, 2, \dots, l\} \times \mathbb{N})^{*n}$ we define labelled transitions $(q_1, s) \vdash^{(q_2, op)} (q_2, t)$ if there is $(q_1, \sigma, q_2, op) \in \Delta$ such that $op(s) = t$ and $\mathrm{Sym}(s) = \sigma$. Now define the relation $\vdash := \bigcup \vdash^{(q,op)}$. We set $C(S)$ to be the set of all configurations that are reachable from $(q_0, \perp_l)$ via $\vdash$-paths. We call $C(S)$ the set of *reachable* or *valid* configurations. Then the *collapsible pushdown graph (*CPG*) generated by S* is

$$\mathrm{Grph}(S) := \big(C(S), (C(S)^2 \cap \vdash^{\ell})_{\ell \in Q \times \mathrm{OP}}\big)$$

Example 3.2. The following example of a collapsible pushdown graph of level 2 is taken from [6]. Let $Q := \{0, 1, 2\}, \Sigma := \{\perp, a\}$, and Δ is given by $(0, -, 1, \mathrm{clone}_2)$, $(1, -, 0, \mathrm{push}_{a,2})$, $(1, -, 2, \mathrm{push}_{a,2})$, $(2, a, 2, \mathrm{pop}_1)$, and $(2, a, 0, \mathrm{collapse})$, where $-$ denotes any letter from Σ. In our picture, the labels are abbreviated as follows: we set $\mathrm{cl} := (1, \mathrm{clone}_2)$, $a := (0, \mathrm{push}_{a,2})$, $a' := (2, \mathrm{push}_{a,2})$, $p := (2, \mathrm{pop}_1)$, and $\mathrm{co} := (0, \mathrm{collapse})$.

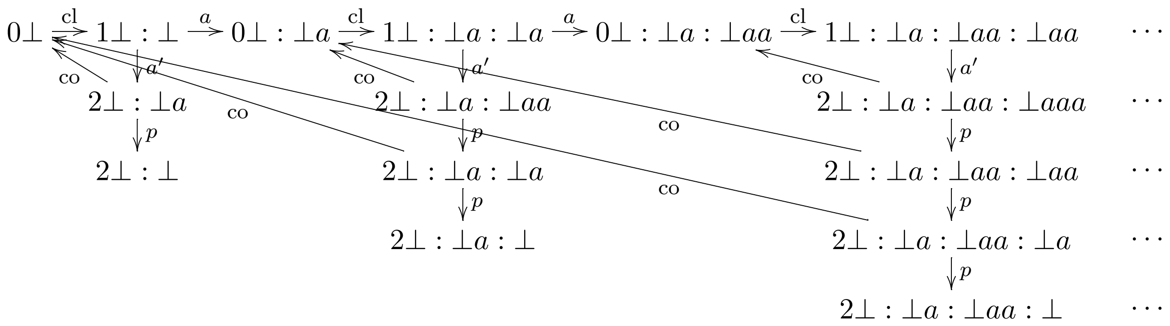

Figure 1: Example of Level 2 CPG

Remark 3.3. By a result of Hague et al. [6], deciding $L\mu$ sentences on level n collapsible pushdown graphs is n-EXPTIME complete. On the other hand, the graph in the previous example has undecidable MSO theory.

The next definition introduces the notion of a run of a CPS.

Definition 3.4. Let S be a CPS of level l. A run r of S of length n is a path in $Q \times (\Sigma \times \{1, 2, \dots, l\} \times \mathbb{N})^{*l}$, i.e., a function

$$r : \{0, 1, 2, \dots, n\} \to (\Sigma \times \{1, 2, \dots, l\} \times \mathbb{N})^{*l} \qquad \text{such that}$$
$$r(0) \vdash r(1) \vdash \dots \vdash r(n).$$

We write $\ln(r) := n$ and call r a run from $r(0)$ to $r(n)$.We say that r visits a stack s at i if $r(i) = (q, s)$.

For runs r, r' of length n and m, respectively, with $r(n) = r'(0)$, we define the composition of r and r' as

$$r \circ r' : \{0, 1, \dots, n + m\} \to Q \times (\Sigma \times \{1, 2, \dots l\} \times \mathbb{N})^{*l}$$
$$(r \circ r')(i) := \begin{cases} r(i) & \text{if } i \leq n \\ r'(i - n) & \text{otherwise} \end{cases}.$$

Remark 3.5. Note that we do not require runs to start in the initial configuration.

$$\begin{array}{cccccccc}
 & f & & & & & & \\
 & e & g & & i & & & \\
b & d & d & d & h & & j & l \\
a & c & c & c & c & c & c & k \\
\bot & \bot & \bot & \bot & \bot & \bot & \bot & \bot
\end{array}$$

Figure 2: Example of blocks of a stack

An important observation about collapsible pushdown systems is the fact that the reachability of some given configuration in a collapsible pushdown system is decidable by reduction to $L\mu$ model checking. The proof is based on the idea that a stack is uniquely determined by its top element and the information which substacks can be reached via collapse- and pop_i-operations.

Theorem 3.6. *Given some* CPS $S = (\Gamma, Q, \Delta, q_0)$*, some* $q \in Q$ *and some stack* s *then it is decidable whether* (q, s) *is a reachable configuration of* S*, i.e. whether* (q, s) *is a vertex of* $\mathrm{Grph}(S)$*.*

4. Encoding of 2-CPG in Trees

From now on, CPG and CPS stand only for those of level 2. We prove that these CPG are tree-automatic. For this purpose we have to encode stacks in trees. The idea is to divide a stack into blocks and to encode different blocks in different subtrees. The crucial observation is that every stack is a list of words that share the same first letter. A block then is a maximal list of words in the stack that share the same two first letters. If we remove the first letter of every word of such a block, the resulting level 2 word decomposes again as a list of blocks. Thus, we can inductively carry on to decompose parts of a stack into blocks and code every block in a different subtree. The roots of these subtrees are labelled with the first letter of the block. This results in a tree where every initial left-closed path in the tree represents one word of the stack. By left-closed, we mean that the last element of the path has no left successor.

It turns out that every stack operation corresponds to a simple tree-operation. The main difficulty is to provide a tree-automaton that verifies whether there is a run to the configuration represented by some tree. This problem is addressed in Section 5.

As we already mentioned, the encoding works by dividing stacks into blocks. Hence we start by making our notion of blocks precise. For $w \in \Sigma^*$ and $s = w_1 : w_2 : \ldots : w_n \in \Sigma^{*2}$, we write $s' := w \setminus s$ for $s' = [ww_1] : [ww_2] : \ldots : [ww_n]$ and say that s' is *s prefixed by w*. Let $t := t' : w$ be some stack. Then we also use the notation $t \setminus s$ as abbreviation for $t' : (w \setminus s)$.

Definition 4.1 (σ-block(line))**.** For $\sigma \in \Sigma$, we call $b \in \Sigma^{*2}$ a *σ-block* if $b = [\sigma]$ or $b = \sigma\tau \setminus s'$ for some $\tau \in \Sigma$ and $s' \in \Sigma^{*2}$. See Figure 2 for examples of blocks. If $b_1, b_2, \ldots, b_n$ are σ-blocks, then we call $b_1 : b_2 : \ldots : b_n$ a *σ-blockline.*

Note that every stack in $\mathrm{Stck}_2(\Sigma)$ forms a $(\bot, 1, 0)$-blockline. Furthermore, every blockline l decomposes uniquely as $l = b_1 : b_2 : \ldots : b_n$ of maximal blocks b_i in l. We will call these maximal blocks the blocks of l.

$$
\begin{array}{ccccc}
 & (c,2,1) & & (e,1,3) & \\
(b,2,0) & (b,2,0) & (c,1,2) & (d,2,3) & \\
(a,2,0) & (a,2,0) & (a,2,2) & (a,2,2) & (a,2,2) \\
(\bot,1,0) & (\bot,1,0) & (\bot,1,0) & (\bot,1,0) & (\bot,1,0)
\end{array}
\qquad
\begin{tikzcd}
 & c,2 & & e,1 & \\
b,2 \arrow[r] & \varepsilon \arrow[u] & c,1 & d,2 \arrow[u] & \\
a,2 \arrow[u] & & a,2 \arrow[u] \arrow[r] & \varepsilon \arrow[r] \arrow[u] & \varepsilon \\
\bot,1 \arrow[u] \arrow[rr] & & \varepsilon \arrow[u] & &
\end{tikzcd}
$$

Figure 3: A stack s and its Encoding $\mathrm{Enc}(s)$: right arrow lead to 1-successors (right successors), upward arrows lead to 0-successors (left successors).

Another crucial observation is that a σ-block $b \in \Sigma^{*2} \setminus \Sigma$ decomposes as $b = \sigma \setminus l$ for some blockline l and we say l is the induced blockline of b. For $b \in \Sigma$ the induced blockline of b is just the empty set.

Now we encode a (σ, n, m)-blockline l in a tree by labelling the root with (σ, n), by encoding the blockline induced by the first block of l in the left subtree, and by encoding the rest of the blockline in the right subtree . In order to avoid repetitions, we do not repeat the symbol (σ, n) in the right subtree, but replace it by the empty letter ε.

Definition 4.2. Let $s = w_1 : w_2 : \ldots : w_n \in (\Sigma \times \{1,2\} \times \mathbb{N})^{+2}$ be a (σ, l, k)-blockline. Let w'_i be words such that $s = (\sigma, l, k) \setminus [w'_1 : w'_2 : \ldots : w'_n]$ and set $s' := w'_1 : w'_2 : \ldots : w'_n$. As an abbreviation we write ${}_is_k := w_i : w_{i+1} : \ldots : w_k$. Furthermore, let $w_1 : w_2 : \ldots : w_j$ be a maximal block of s. Note that $j > 1$ implies $w_{j'} = (\sigma, l, k)(\sigma', l', k')w''_{j'}$ for all $j' \leq j$ and for some $(\sigma', l', k') \in \Sigma \times \{1,2\} \times \mathbb{N}$. Then for arbitrary $\rho \in (\Sigma \times \{1,2\}) \cup \{\varepsilon\}$ we define recursively the $(\Sigma \times \{1,2\}) \cup \{\varepsilon\}$-labelled tree $\mathrm{Enc}(s, \rho)$ via

$$
\mathrm{Enc}(s,\rho) := \begin{cases} \rho & \text{if } |w_1| = 1, n = 1 \\ \rho(\emptyset, \mathrm{Enc}({}_2s_n, \varepsilon)) & \text{if } |w_1| = 1, n > 1 \\ \rho(\mathrm{Enc}({}_1s'_n, (\sigma', l')), \emptyset) & \text{if } j = n, |w_1| > 1 \\ \rho(\mathrm{Enc}({}_1s'_j, (\sigma', l')), \mathrm{Enc}({}_{j+1}s_n, \varepsilon)) & \text{otherwise} \end{cases}
$$

$\mathrm{Enc}(s) := \mathrm{Enc}(s, (\bot, 1))$ is called the (tree-)encoding of the stack $s \in \mathrm{Stck}_2(\Sigma)$

Figure 3 shows a configuration and its encoding.

Remark 4.3. In this encoding, every (σ, l, k)-block is encoded in a subtree whose root d is labelled (σ, l). We can restore k from the position of d in the tree $\mathrm{Enc}(s)$ as follows: if $l = 1$ then $k = |d|_0$, i.e., the number of occurrences of 0 in d. This is due to the fact that level 1 links always point to the preceding letter. Note that we always introduce a left-successor tree in order to encode letters that are higher in the stack. If $l = 2$ then $k = |\{d' \in T \cap \{0,1\}^*1 : d' \leq_{lex} d\}|$. This is due to the fact that every right-successor corresponds to the separation of some block from some other. But every separation of blocks corresponds to the separation of two words, namely the last word of the first block and the first word of the second block. Note that the point of the separation of two blocks corresponds to the position of the blocks in the stack. Consider the encoding of some stack s. Assume that there are four blocks b_1, b_2, b_3, b_4 in the stack. Assume furthermore that we separate b_1 from b_2 at some point in the encoding and b_3 from b_4 at another point. We also assume that the first word in b_2 is a postfix of the i-th word of s and the first word in b_4 is a postfix of the j-th word of s. If $i < j$, then the separation of b_1 from b_2 is further to the left than the separation of b_3 from b_4. This means the element corresponding to the former separation is lexicographically smaller than the element corresponding to the latter.

This correspondence can be seen as a bijection as follows. Let $s = w_1 : w_2 : \ldots : w_n$ be some stack and let $R := \mathrm{dom}(\mathrm{Enc}(s)) \cap (\{\varepsilon\} \cup \{0,1\}^*1)$. Then there is a bijection $f : \{1,2,3,\ldots,n\} \to R$ such that i is mapped to the i-th element of R in lexicographic order. This map associates with the first word of s the root of $\mathrm{Enc}(s)$ and with every other word in s the element of $\mathrm{Enc}(s)$ that separates this word from its left neighbour in s.

If we interpret ε as empty word, the word from the root to $f(i)$ in $\mathrm{Enc}(s)$ is $w_{i-1} \sqcap w_i$. More precisely, the word read along this path is the projection onto the letters and link-levels of this prefix.

Furthermore, set $f'(i) := d0^m \in \mathrm{Enc}(s)$ such that $d = f(i)$ and m is maximal with this property, i.e., $f'(i)$ is the leftmost descendent of $f(i)$. Then the path from $f(i)$ to $f'(i)$ is the postfix w'_i such that $w_i = (w_{i-1} \sqcap w_i) \circ w'_i$ (here we set $w_0 := \varepsilon$). More precisely, the word read along this path is the projection onto the letters and link-levels of w'_i.

In order to encode configurations, we just add the state as new root of the tree and attach the encoding of the stack as left subtree, i.e., for some configuration (q, s) we set $\mathrm{Enc}(q, s) := q(\mathrm{Enc}(s), \emptyset)$.

The image of this encoding function contains only trees of a very specific type. We call this class $\mathbb{T}_{\mathrm{Enc}}$. In the next definition we state the characterising properties of $\mathbb{T}_{\mathrm{Enc}}$. This class is MSO-definable, whence automata-recognisable.

Definition 4.4. Let $\mathbb{T}_{\mathrm{Enc}}$ be the class of trees that satisfy the following conditions.

(1) The root of T is labelled by some element of Q ($T(\varepsilon) \in Q$).
(2) Every element of the form $\{0,1\}^*0$ is labelled by some $(\sigma, l) \in \Sigma \times \{1,2\}$, especially $T(0) = (\bot, 1)$.
(3) Every element of the form $\{0,1\}^*1$ is labelled by ε.
(4) $1 \notin \mathrm{dom}(T)$, $0 \in \mathrm{dom}(T)$.
(5) For all $t \in T$ we have that $T(t0) = (\sigma, 1)$ implies $T(t10) \neq (\sigma, 1)$.

Remark 4.5. Note that (5) holds as $T(t0) = T(t10) = (\sigma, 1)$ would imply that the subtree rooted at t encodes a blockline l such that the first block of l induces a $(\sigma, 1, n)$-blockline and the second one induces a $(\sigma, 1, m)$-blockline. But as level 1 links always point to the preceding letter, n and m are equal to the length of the prefix of l in the stack plus 1, i.e., if T encodes a stack s then $s = s_1 : [w \setminus l] : s_2$ and $n = m = |w| + 1$. This contradicts the maximality of the blocks in the encoding.

Remark 4.6. $\mathrm{Enc} : Q \times \mathrm{Stck}_2\Sigma \to \mathbb{T}_{\mathrm{Enc}}$ is a bijection and we denote its inverse by Dec.

5. Recognising Reachable Configurations

In this section, we show that Enc maps the reachable configurations of a given CPS to a regular set. For this purpose we introduce milestones of a stack s. These are exactly those substacks of s that every run to s has to visit. It turns out that the milestones are represented by the nodes of the encoding of s: with every $d \in \mathrm{Enc}(s)$, we can associate a subtree of s which encodes a milestone. It turns out that the substack relation on the milestones corresponds exactly to the lexicographical order of the elements of $\mathrm{Enc}(s)$. We show that we can guess for every $d \in \mathrm{Enc}(s)$ the state in which the corresponding milestone is visited for the last time by some run to s and that we can check the correctness of this guess using MSO or, equivalently, tree-automata.

We prove the fact that we can check the correctness of such a guess by introducing a special type of run, called *loop*. A run from one milestone to the next will mainly consist of loops combined with a finite number of stack operations.

We start by giving a precise notion of a loop and we analyse their occurrences for a given stack.

Definition 5.1. Let r be a run r of $\ln(r) = n$ and set $(q_i, s_i) := r(i)$. r is called a *simple high loop* if $s_0 = s_n$ and if s_0 is a proper substack of s_i for all $0 < i < n$.

The run r is called a *simple low loop* of s if $s_0 = s_n = s$, between 0 and n the stack s is never visited, $s_1 = \mathrm{pop}_1(s)$, $\mathrm{CLvl}(s) = 1$, $|s_i| \geq |s|$ for all $r(i) = (q_i, s_i)$, and $r{\restriction}_{[2,\ln(r)-1]}$ is the composition of simple low loops and simple high loops of $\mathrm{pop}_1(s)$.

We define a *loop* as a finite composition of low loops and high loops.

We also define *returns* which are runs from a stack $s : w$ to s without visiting substacks of s. Furthermore, we require that they are independent of the level 2 collapse links stored in the word w. Equivalently, we want that returns from $s : w$ to s are independent of s in the sense that if there is a return from $s : w$ to s then there is a return from $s' : w$ to s' for all nonempty 2-words s'.

Our interest for returns stems from the fact that they appear as subruns of high loops whence they play an important role in finding loops for a given stack. Now we give the precise definition.

Definition 5.2. Let $s' = s : w$ be some stack. A *return from s' to s* is a run r from s' to s such that r never visits a substack of s at $i < \ln(r)$ and such that one of the following hold:

(1) the last operation in r is pop_2,
(2) the last operation in r is a collapse and $w < \mathrm{top}_2(r(\ln(r) - 1))$, i.e., r pushes at first some new letters onto s' and then performs a collapse of one of these new letters, or
(3) the last operation in r is a collapse and there is some $i \in \mathrm{dom}(r)$ such that $r{\restriction}_{[i,\ln(r)]}$ is a return from $\mathrm{pop}_1(s')$ to s.

Returns and high loops are closely linked. Every return of a valid configuration $(q, s : w)$ occurs as subrun of a loop of the stack s. Every run r to $(q, s : w)$ passes s at some position i for the last time. Now compose $r{\restriction}_{[i,\ln(r)]}$ with the return. The resulting run is a loop of s. On the other hand returns play a crucial role in understanding the existence of high loops. Assume that r is a loop of the stack s and set $w := \mathrm{top}_2(s)$. It may happen that r visits at i some stack $s' : w$ such that s is a substack of s'. As r finally reaches s again it also has to pass s' (the loop cannot use a collapse in order to jump back to s without passing s' as the links in w are too small and such a collapse would reach a substack of $\mathrm{pop}_2(s)$). Restricting r to the interval between i and the first occurrence of s', we obtain a return of $s' : w$.

Later, we are interested in the initial and the final states of loops and returns of certain stacks s. For this purpose, we define $\mathrm{Loops}(s) \subseteq Q \times Q$ by $(q_1, q_2) \in \mathrm{Loops}(s)$ if there is a loop from (q_1, s) to (q_2, s). Analogously, we define $\mathrm{Returns}(s) \subseteq Q \times Q$ by $(q_1, q_2) \in \mathrm{Returns}(s)$ if there is a return from (q_1, s) to $(q_2, \mathrm{pop}_2(s))$.

The crucial observation is that $\mathrm{Loops}(s)$ and $\mathrm{Returns}(s)$ may be calculated by a finite automaton reading $\mathrm{top}_2(s)$.

Lemma 5.3. *For all stacks s with $s' := \mathrm{pop}_1(s)$ defined,* $\mathrm{Sym}(s')$*,* $\mathrm{Loops}(s')$*,* $\mathrm{Returns}(s')$*,* $\mathrm{CLvl}(s)$*, and* $\mathrm{Sym}(s)$ *effectively determine* $\mathrm{Loops}(s)$ *and* $\mathrm{Returns}(s)$.

5.1. Milestones

Now we define milestones of a stack s, which are those substacks of s that have to be visited by any run from the initial configuration to s. After the formal definition we show that this definition fits our informal description.

Definition 5.4 (Milestone)**.** A substack s' of $s = w_1 : w_2 : \ldots : w_n$ is a *milestone* if $s' = w_1 : w_2 : \ldots : w_i : w'$ such that $w_i \sqcap w_{i+1} \leq w' \leq w_{i+1}$. We denote by $\mathrm{MS}(s)$ the set of milestones of s.

Note that the substack relation linearly orders $\mathrm{MS}(s)$.

Lemma 5.5.

- *Let s_1, s_2, s be stacks such that $s \in \mathrm{MS}(s_2)$ but $s \not\leq s_1$. Then any run from s_1 to s_2 visits s.*
- *For every run r from the initial configuration to s, the function*

$$f : \mathrm{MS}(s) \to dom(r), \qquad s' \mapsto \max\{i \in dom(r) : r(i) = (q, s') \text{ for some } q \in Q\}$$

is an order embedding with respect to substack relation on the milestones and the natural order of $dom(r)$.

There is a close correspondence between milestones of a stack s and the elements of the tree $\mathrm{Enc}(s)$. The next definition associates to every element of $\mathrm{Enc}(s)$ a stack which turns out to be a milestone.

Definition 5.6. Let $T \in \mathbb{T}_{\mathrm{Enc}}$ be a tree and $d \in T \setminus \{\varepsilon\}$. Then the *left and downward closed tree of* d is $LT(d,T) := T{\restriction}_D$ where $D := \{d' \in T : d' \leq_{lex} d\} \setminus \{\varepsilon\}$. Then we denote $\mathrm{LStck}(d,T) := \mathrm{Dec}(LT(d,T))$ the *left stack induced by* d. If T is clear from the context, we omit it.

Remark 5.7. $\mathrm{LStck}(d,s)$ is a substack of s for all $d \in \mathrm{dom}(\mathrm{Enc}(s))$. This observation follows from Remark 4.3 combined with the fact that the left stack is induced by a lexicographically downward closed subset.

Lemma 5.8. *The map $g : \big(dom(\mathrm{Enc}(q,s)) \setminus \{\varepsilon\}, <_{lex}\big) \to \big(\mathrm{MS}(s), \leq\big)$ with $d \mapsto \mathrm{LStck}(d,s)$ is an order isomorphism.*

This lemma implies that every run r decomposes as $r = l_0 \circ o_1 \circ l_1 \circ o_2 \circ l_2 \ldots \circ o_n \circ l_n$ where l_i is a loop and o_i is a run from the last occurrence of the i-th milestone in r to the $(i+1)$-th milestone. If the $(i+1)$-th milestone differs from the i-th by exactly one $\mathrm{push}_{\sigma,k}$-operation, then o_1 only consists of one transition performing this operation. For the other case, the following lemma characterises the form of o_i.

Lemma 5.9. *Let r be a run from some stack s to the stack ${\mathrm{pop}_1}^n(\mathrm{clone}_2(s))$ that visits s only in the initial state. Then $r = o_0 \circ l_0 \circ o_1 \circ l_1 \circ o_2 \circ l_2 \ldots o_n \circ l_n$ where l_i are loops and o_0 is a run that only performs one clone-operations and for $j > 0$ the run o_j only performs a pop_1-operation.*

5.2. Detection of Reachable Configurations

We have already seen that every run to a valid configuration (q, s) passes all the milestones of s. Now, we use the last states in which a run r to (q, s) visits the milestones as a certificate for the reachability of (q, s). To be precise, *a certificate for the reachability of* (q, s) by a run r is a map $C_r : \mathrm{dom}(\mathrm{Enc}(q, s)) \setminus \{\varepsilon\} \to Q$ such that $d \mapsto q$ if and only if $r(i) = (q, \mathrm{LStck}(d))$ and i is the maximal position in r where $\mathrm{LStck}(d)$ is visited.

Lemma 5.10. *For every* CPG G*, one can effectively construct a tree-automaton that checks for any map*

$$f : dom(\mathrm{Enc}(q, s)) \setminus \{\varepsilon\} \to Q$$

whether f is a certificate of the reachability of (q, s), i.e., whether $f = C_r$ for some run r from the initial configuration to (q, s).

The proof of the lemma uses Lemma 5.3 and the fact that the path from the root to some $d \in \mathrm{Enc}(s)$ encodes the topmost word of $\mathrm{LStck}(d, \mathrm{Enc}(s))$. Hence, a tree automaton reading $\mathrm{Enc}(s)$ is able to calculate for each position $d \in \mathrm{Enc}(s)$ the pairs of initial and final states of loops of $\mathrm{LStck}(d)$. As every run decomposes as a sequence of loops separated by a single operation, knowing $\mathrm{Loops}(s')$ for each $s' \leq s$ enables the automaton to check the correctness of a candidate for a certificte of reachability.

Noting that a tree-automaton may non-deterministically guess a certificate of the reachability of a configuration, we obtain that the encodings of reachable configurations form a regular set. In order to finish our proof of the tree-automaticity of CPG, we have to show that Enc translates every stack operation into a regular tree operation on the set of convolutions of trees. Due to space limitations, we cannot give any details of this part of the proof which is rather easy but contains a lot of case distinctions.

Lemma 5.11. *Let C be the set of encodings of configurations of a* CPS S*. Then one can effectively compute automata $A_{(q,\mathrm{op})}$ for $q \in Q$ and $\mathrm{op} \in \mathrm{OP}$ such that for all $c_1, c_2 \in C$*

$$A_{(q,\mathrm{op})} \textit{ accepts } \mathrm{Enc}(c_1) \otimes \mathrm{Enc}(c_2) \qquad \textit{iff} \qquad c_1 \vdash^{(q,\mathrm{op})} c_2 \ .$$

Putting these lemmas together, we obtain our main result.

Theorem 5.12. *Given a collapsible pushdown system S of level* 2*, one can effectively compute a tree-automatic representation of* $\mathrm{CPG}(S)$*, the collapsible pushdown graph generated by S.*

5.3. The Reachability Predicate is Tree-Automatic

Using similar techniques we obtain an even stronger result. Due to space restrictions we cannot give any details of the proofs. The proof is based on a similar analysis of runs, but this time runs in the reverse direction: runs from some stack s to some substack of s.

Theorem 5.13. *If G is a graph obtained by ε-contraction from a level* 2 CPG G' *then* (G, REACH) *is tree-automatic where* REACH *is the binary predicate that is true on configurations c_1, c_2 if there is a path from c_1 to c_2.*

Using ε-contractions on the level l collapsible pushdown graphs generates a class containing the l-th level of the Caucal-hierarchy. Hence, this theorem proves also the tree-automaticity of the second level of the Caucal-hierarchy.

6. Conclusion

We have seen that level 2 collapsible pushdown graphs are tree-automatic. This result holds also if we apply ε-contractions and if we add a reachability predicate. From this result there arise two further questions. Are level 3 collapsible pushdown graphs tree-automatic? We know an example of a level 5 CPG which is not tree-automatic. This raises the further question whether all CPG have decidable FO-theories. In order to solve this problem one has to come up with a completely new technique. A rather general question concerning our result aims at our knowledge about tree-automatic structures. Recent developments in the string case [9] show the decidability of rather large extensions of first-order logic for automatic structures. It would be interesting to clarify the status of the analogous claims for tree-automatic structures. Positive answers concerning decidability of extensions of first-order logic give us the corresponding decidability results for collapsible pushdown graphs of order 2.

References

[1] Rajeev Alur, Swarat Chaudhuri, and P. Madhusudan. Languages of nested trees. In *In Proc. 18th International Conference on Computer-Aided Verification*, volume 4144 of *LNCS*, pages 329–342. Springer, 2006.

[2] Achim Blumensath. Automatic structures. Diploma thesis, RWTH Aachen, 1999.

[3] Arnaud Carayol and Stephan Wöhrle. The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *Proceedings of the 23rd Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2003*, volume 2914 of *LNCS*, pages 112–123. Springer, 2003.

[4] Didier Caucal. On infinite terms having a decidable monadic theory. In *MFCS'02*, pages 165–176, 2002.

[5] John Doner. Tree acceptors and some of their applications. *J. Comput. Syst. Sci.*, 4(5):406–451, 1970.

[6] M. Hague, A. S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *LICS '08: Proceedings of the 2008 23rd Annual IEEE Symposium on Logic in Computer Science*, pages 452–461, 2008.

[7] A. Kartzow. Fo model checking on nested pushdown trees. In *MFCS'09*, volume 5734 of *LNCS*, pages 451–463, 2009.

[8] T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FOSSACS'02*, volume 2303 of *LNCS*, pages 205–222. Springer, 2002.

[9] Dietrich Kuske. Theories of automatic structures and their complexity. In *CAI'09, Third International Conference on Algebraic Informatics*, volume 5725 of *LNCS*, pages 81–98, 2009.

[10] A. N. Maslov. The hierarchy of indexed languages of an arbitrary level. *Sov. Math., Dokl.*, 15:1170–1174, 1974.

[11] A. N. Maslov. Multilevel stack automata. *Problems of Information Transmission*, 12:38–43, 1976.

[12] James W. Thatcher and Jesse B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

Appendix

As our paper does not contain any proofs at all, we give them in the following appendix. For better readability, the appendix is self contained, i.e., we repeat the crucial definitions which are already contained in the paper. Here is a list of the claims of the paper and how they follow from the statements in the appendix:

- Theorem 3.6 corresponds to Theorem A.1 in the appendix.
- Remark 4.6 follows from Lemma B.8 and Corollary B.10.
- Lemma 5.3 is a summary of the Lemmas C.7 and C.8
- Lemma 5.5 is a combination of Lemma C.11 and Corollary C.12
- Lemma 5.8 corresponds to Lemma C.16
- Lemma 5.9 corresponds to Lemma C.17
- Lemma 5.10 corresponds to Lemma C.18
- Lemma 5.11 corresponds to Lemma D.3
- Lemma 5.13 is proven in the Section E

Appendix A. Existence of a Given Configuration in a Given CPS

An important observation about collapsible pushdown systems is the fact that the reachability of some given configuration in a collapsible pushdown system is decidable by reduction to $L\mu$ model checking. The proof is based on the idea that a stack is uniquely determined by its top element and the information which substacks can be reached via collapse- and pop_i-operations. We can computably generate an extension of a given CPS S such that a certain modal $L\mu$ holds on this extension if a certain configuration is part of the graph generated by S. The new CPS will extend the old one by new states q_s for every substack s of the configuration we would like to check. We connect two such states by a pop_i-edge if the corresponding substack of the second one is generated by a pop_i-operation on the stack corresponding to the former state. Then we allow an identity-transition between "old configurations" of the original CPS and the state q_s for the stack s which we are looking for. By this we extend the CPS by a kind of testing device for the stack s: at any node in the graph, the new CPS may assume that the stack is s and switches to state q_s. Then the following operations delete the topmost information of the stack as long as the topmost symbol is in accordance to the symbol we would expect when really operating on s. If there occurs a wrong symbol the computation stops. Hence, there is a path from some configuration c via all states $q_{s'}$ for all substacks s' of s to the state $q_{\bot_l}$ if $c = (q, s')$ for some $q \in Q$ and s' a stack such that the (point-wise) projections to Σ^{+l} of s and s' agree. In order to also check the link levels and the links we also have to connect the states q_t and q'_t if $\mathrm{collapse}(t) = t'$ and have to formulate the requirement that the stack reachable via a collapse-operation is the stack we expect it to be. This can also be done in a $L\mu$-formula.

We fix some notation for the next theorem. As our edges are labelled by stack operations and states, we use a multi-modal $L\mu$ with the modal connectives $\langle q, op\rangle\varphi$ for "there is an edge labelled by state q and operation op to a vertex that satisfies φ", and analogously connectives of the form $[q, op]\varphi := \neg\langle q, op\rangle\neg\varphi$. We use $\Diamond$ for "there is a successor via an arbitrary labelled edge".

Theorem A.1. *Given some* CPS $S = (\Gamma, Q, \Delta, q_0)$*, some* $q \in Q$ *and some stack* s *then it is decidable whether* (q, s) *is a reachable configuration of* S*, i.e. whether* (q, s) *is a vertex of* $\mathrm{Grph}(S)$*.*

Proof. We only state the proof for the case of a level 2 CPS. For the general case there appear just more cases in terms of the applicable pop_k-operations at the stack we are looking for.

For every $q \in Q$ and every stack s, we define an automaton $S^{(q,s)}$ and a formula $\psi_{(q,s)} \in L\mu$, such that

$$S^{(q,s)}, (q_0, (\bot, 0)) \models \psi_{(q,s)} \quad \text{iff} \quad (q, s) \in \mathrm{Grph}(S).$$

We set

$$S^{(q,s)} := (\Gamma, Q', \Delta^{(q,s)}, q_0) \text{ with}$$
$$Q' := Q \cup \{q_t : t = {\mathrm{pop}_1}^m {\mathrm{pop}_2}^n(s) \text{ for } m, n \in \mathbb{N}\} \cup \{q_\emptyset\},$$

where the q_t are new states for every substack of the stack of the configuration we are looking for and $q_\emptyset$ is used for checking that certain operations can or cannot be performed on a configuration.

In the following we define $\Delta^{(q,s)} \supseteq \Delta$ by induction on the size of s such that $\Delta^{(q_t,t)} \subseteq \Delta^{(q,s)}$ for all proper substacks $t < s$.

(1) For $s = \bot_2$ we set
$$\Delta^{(q,s)} := \Delta \cup \big\{(q, \bot, q_\emptyset, \mathrm{clone}_2), (q, \bot, q_\emptyset, \mathrm{pop}_2)\big\}.$$
Additionally, we set $\varphi_{(q_0,s)} := \langle q_\emptyset, \mathrm{clone}_2\rangle \mathrm{True} \wedge [q_\emptyset, \mathrm{pop}_2]\mathrm{False}$.

(2) Assume $|s| > 1$ and $\mathrm{Sym}(s) = \bot$ for some stack s. Then let $t = \mathrm{pop}_2(s)$. Then we set
$$\Delta^{(q,s)} := \Delta^{(q_t,t)} \cup \big\{(q, \bot, q_t, \mathrm{pop}_2)\big\}$$
and $\varphi_{(q,s)} := \langle q_t, \mathrm{pop}_2\rangle \varphi_{(q_t,t)}$.

(3) The next case is $\mathrm{Sym}(s) \neq \bot$ and $\mathrm{CLnk}(s) = 0$. Then we set $t := \mathrm{pop}_1(s)$ and
$$\Delta^{(q,s)} := \Delta^{(q_t,t)} \cup \big\{(q, \mathrm{Sym}(s), q_t, \mathrm{pop}_1), (q, \mathrm{Sym}(s), q_\emptyset, \mathrm{collapse})\big\}$$
and
$$\varphi_{(q,s)} := \langle q_t, \mathrm{pop}_1\rangle \varphi_{(q_t,t)} \wedge [q_\emptyset, \mathrm{collapse}]\mathrm{False} \ .$$

(4) In all other cases we set $t := \mathrm{pop}_1(s)$ and $u := \mathrm{collapse}(s)$. We set
$$\Delta^{(q,s)} := \Delta^{(q_t,t)} \cup \big\{(q, \mathrm{Sym}(s), q_t, \mathrm{pop}_1), (q, \mathrm{Sym}(s), q_u, \mathrm{collapse})\big\} \text{ and}$$
$$\varphi_{(q,s)} := \langle q_t, \mathrm{pop}_1\rangle \varphi_{(q_t,t)} \wedge \langle q_u, \mathrm{collapse}\rangle \varphi_{(q_u,u)}.$$

We have to show by induction that for all $q \in Q$ and stacks s

$$\mathrm{Grph}(S^{(q,s)}), c \models \varphi_{(q,s)} \text{ iff } c = (q, s)$$

The initial stack $s = (\bot, 1, 0)$ is characterised by the facts that the top symbol of the stack is the bottom-of-stack symbol and that pop_2 is undefined. The first conjunct of $\phi_{(q_0,s)} = \langle q_\emptyset, \mathrm{clone}_2\rangle \mathrm{True} \wedge [q_\emptyset, \mathrm{pop}_2]\mathrm{False}$ is only satisfiable if the top symbol is $\bot$ and the second conjunct is satisfiable if and only if pop_2 is undefined. Thus, $\varphi_{(q,\bot_2)}$ and $S^{(q,\bot_2)}$ satisfy our claim.

$$
\begin{array}{cccccccc}
 & f & & & & & & \\
 & e & g & & i & & & \\
b & d & d & d & h & & j & l \\
a & c & c & c & c & c & c & k \\
\bot & \bot & \bot & \bot & \bot & \bot & \bot & \bot
\end{array}
$$

Figure 4: Example of blocks of a stack

For the induction step, note that collapse is defined if and only if the link of the top symbol is not 0. If collapse is defined for some stack s and $t := \text{collapse}(s), u = \text{pop}_1(s)$ then $\Delta^{(q_t,t)} \subseteq \Delta^{(q_s,s)}$ as $t = {\text{pop}_1}^{m_1}{\text{pop}_2}^{m_2}(u)$ and by induction hypothesis. With these observations the induction step is quite obvious.

Now we set $\psi_{(q,s)} := \mu Z.(\Diamond Z \vee \varphi_{(q,s)})$ which is just the formula asserting reachability of some point where $\varphi_{(q,s)}$ holds. For c_0 the initial configuration,

$$\text{Grph}(S^{(q,s)}), c_0 \models \psi_{(q,s)} \qquad \text{iff} \qquad (q,s) \in \text{Grph}(S) \ .$$

The former problem is decidable in 2-EXPTIME [6]. ∎

Appendix B. Encoding of 2-CPG in Trees

From now on, we say CPG, CPS, or stack and always mean only those of level 2. The aim of this section is to prove that level 2 CPG are tree-automatic.

For this purpose we have to encode stacks in trees. The idea is to divide a stack into blocks and to encode different blocks in different subtrees. The crucial observation is that every stack is a list of words that share the same first letter. A block then is a maximal list of words in the stack that share the same two first letters. If we remove the first letter of every word of such a block, the resulting level 2 word decomposes again as a list of blocks. Thus, we can inductively carry on to decompose parts of a stack into blocks and code every block in a different subtree. The roots of these subtrees are labelled with the first letter of the block. This results in a tree where every initial left-closed path in the tree represents one word of the stack. By left-closed, we mean that the last element of the path has no left successor.

It turns out that every stack operation corresponds to a simple tree-operation. The main difficulty is to provide a tree-automaton that verifies whether there is a run to the configuration represented by some tree. This problem is solved by guessing for every substack of a given configuration c the state with which this stack occurs as a configuration in some run to c. Checking the correctness of such a guess involves a technical analysis of *loops*: these are runs that start and end with the same stack.

Now we introduce formally the function Enc that translates between stacks and trees. As we already mentioned, this works by dividing stacks into blocks. Hence we start by making our notion of blocks precise. For $w \in \Sigma^*$ and $s = w_1 : w_2 : \ldots : w_n \in \Sigma^{*2}$, we write $s' := w \setminus s$ for $s' = [ww_1] : [ww_2] : \ldots : [ww_n]$ and say that s' is *s prefixed by w*. Let $t := t' : w$ be some stack. Then we also use the notation $t \setminus s$ as abbreviation for $t' : (w \setminus s)$.

$$\begin{array}{ccccc}
 & (c,2,1) & & (e,1,3) & \\
(b,2,0) & (b,2,0) & (c,1,2) & (d,2,3) & \\
(a,2,0) & (a,2,0) & (a,2,2) & (a,2,2) & (a,2,2) \\
(\bot,1,0) & (\bot,1,0) & (\bot,1,0) & (\bot,1,0) & (\bot,1,0)
\end{array}$$

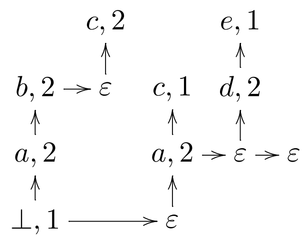

Figure 5: A stack s and its Encoding $\mathrm{Enc}(s)$: right arrow lead to 1-successors (right successors), upward arrows lead to 0-successors (left successors).

Definition B.1 (σ-block(line))**.** For $\sigma \in \Sigma$, we call $b \in \Sigma^{*2}$ a σ*-block* if $b = [\sigma]$ or $b = \sigma\tau \setminus s'$ for some $\tau \in \Sigma$ and $s' \in \Sigma^{*2}$. See Figure 4 for examples of blocks. If $b_1, b_2, \dots, b_n$ are σ-blocks, then we call $b_1 : b_2 : \dots : b_n$ a σ*-blockline*.

Note that every stack in $\mathrm{Stck}_2(\Sigma)$ forms a $(\bot, 1, 0)$-blockline. Furthermore every blockline l decomposes uniquely as $l = b_1 : b_2 : \dots : b_n$ of maximal blocks b_i in l. We will call these maximal blocks the blocks of l.

Another crucial observation is that a σ-block $b \in \Sigma^{*2} \setminus \Sigma$ decomposes as $b = \sigma \setminus l$ for some blockline l and we say l is the induced blockline of b. For $b \in \Sigma$ the induced blockline of b is just the empty set.

We encode a (σ, n, m)-blockline l in a tree by labelling the root with (σ, n), by encoding the blockline induced by the first block of l in the left subtree, and by encoding the rest of the blockline in the right subtree. Here we do not repeat the symbol (σ, n) in the right subtree, but replace it by the empty letter ε. Later on, we will see that ε represents cloned symbols.

Definition B.2. Let $s = w_1 : w_2 : \dots : w_n \in (\Sigma \times \{1,2\} \times \mathbb{N})^{+2}$ be a (σ, l, k)-blockline. Let w'_i be words such that $s = (\sigma, l, k) \setminus [w'_1 : w'_2 : \dots : w'_n]$ and set $s' := w'_1 : w'_2 : \dots : w'_n$. As an abbreviation we write

$$_is_k := w_i : w_{i+1} : \dots : w_k \ .$$

Furthermore let $w_1 : w_2 : \dots : w_j$ be a maximal block of s. Note that $j > 1$ implies $w_{j'} = (\sigma, l, k)(\sigma', l', k')w''_{j'}$ for all $j' \leq j$ and for some $(\sigma', l', k') \in \Sigma \times \{1,2\} \times \mathbb{N}$. Then for arbitrary $\rho \in (\Sigma \times \{1,2\}) \cup \{\varepsilon\}$ we define recursively the $(\Sigma \times \{1,2\}) \cup \{\varepsilon\}$-labelled tree $\mathrm{Enc}(s, \rho)$ via

$$\mathrm{Enc}(s,\rho) := \begin{cases} \rho & \text{if } |w_1| = 1, n = 1 \\ \rho(\emptyset, \mathrm{Enc}(_2s_n, \varepsilon)) & \text{if } |w_1| = 1, n > 1 \\ \rho(\mathrm{Enc}(_1s'_n, (\sigma', l')), \emptyset) & \text{if } j = n, |w_1| > 1 \\ \rho(\mathrm{Enc}(_1s'_j, (\sigma', l')), \mathrm{Enc}(_{j+1}s_n, \varepsilon)) & \text{otherwise} \end{cases}$$

$\mathrm{Enc}(s) := \mathrm{Enc}(s, (\bot, 1))$ is called the (tree-)encoding of the stack $s \in \mathrm{Stck}_2(\Sigma)$

Figure 5 shows a configuration and its encoding.

Remark B.3. In this encoding, every (σ, l, k)-block is encoded in a subtree whose root d is labelled (σ, l). We can restore k from the position of d in the tree $T := \mathrm{Enc}(s)$ as follows: if $l = 1$ then $k = |d|_0$, i.e., the number of occurrences of 0 in d. This is due to the fact that level 1 links always point to the preceding letter. Note that we always introduce a left-successor tree in order to encode letters that are higher in the stack. If $l = 2$ then $k = |\{d' \in T \cap \{0,1\}^*1 : d' \leq_{lex} d\}|$. This is due to the fact that every right-successor corresponds to the separation of some block from some other. But every separation of blocks corresponds to the separation of two words, namely the last word of the first block and the first word of the second block. Note that the point of the separation of two blocks corresponds to the position of the blocks in the stack. Consider the encoding of some stack s. Assume that there are four blocks b_1, b_2, b_3, b_4 in the stack. Assume furthermore that we separate b_1 from b_2 at some point in the encoding and b_3 from b_4 at another point. We also assume that the first word in b_2 is a postfix of the i-th word of s and the first word in b_4 is a postfix of the j-th word of s. If $i < j$, then the separation of b_1 from b_2 is further to the left than the separation of b_3 from b_4. This means the element corresponding to the former separation is lexicographically smaller than the element corresponding to the latter.

This correspondence can be seen as a bijection as follows. Let $s = w_1 : w_2 : \ldots : w_n$ be some stack. Then there is a bijection $f : \{1,2,3,\ldots,n\} \to \mathrm{dom}(\mathrm{Enc}(s)) \cap (\{\varepsilon\} \cup \{0,1\}^*1) =: R$ such that i is mapped to the i-th element of R in lexicographic order. This map associates with the first word of s the root of $\mathrm{Enc}(s)$ and with every other word in s the element of $\mathrm{Enc}(s)$ that separates this word from its left neighbour in s.

If we interpret ε as empty word, the word from the root to $f(i)$ in $\mathrm{Enc}(s)$ is $w_{i-1} \sqcap w_i$. More precisely, the word read along this path is the projection onto the letters and link-levels of this prefix.

Furthermore, set $f'(i) := d0^m \in \mathrm{Enc}(s)$ such that $d = f(i)$ and m is maximal with this property, i.e., $f'(i)$ is the leftmost descendent of $f(i)$. Then the path from $f(i)$ to $f'(i)$ is the postfix w'_i such that $w_i = (w_{i-1} \sqcap w_i) \circ w'_i$ (here we set $w_0 := \varepsilon$). More precisely, the word read along this path is the projection onto the letters and link-levels.

Up to now, we have only encoded stacks in trees, but we want to encode configurations, i.e., a stack together with a state information. For this purpose we just add the state as new root of the tree and attach the encoding of the stack as left subtree.

Definition B.4. For $(q, s) \in \mathrm{Grph}(S)$, we set $\mathrm{Enc}(q, s) := q(\mathrm{Enc}(s), \emptyset)$.

The image of this encoding function contains only trees of a very specific type. We call this class $\mathbb{T}_{\mathrm{Enc}}$. In the next definition we state the characterising properties of $\mathbb{T}_{\mathrm{Enc}}$. Note that this class is MSO-definable, whence automata-recognisable.

Definition B.5. Let $\mathbb{T}_{\mathrm{Enc}}$ be the class of trees that satisfy the following conditions.

(1) The root of T is labelled by some element of Q ($T(\varepsilon) \in Q$).
(2) Every element of the form $\{0,1\}^*0$ is labelled by some $(\sigma, l) \in \Sigma \times \{1,2\}$, especially $T(0) = (\bot, 1)$.
(3) Every element of the form $\{0,1\}^*1$ is labelled by ε.
(4) $1 \notin \mathrm{dom}(T)$, $0 \in \mathrm{dom}(T)$.
(5) For all $t \in T$ we have that $T(t0) = (\sigma, 1)$ implies $T(t10) \neq (\sigma, 1)$ (see the following remark).

Remark B.6. Note that 5. holds as $T(t0) = T(t10) = (\sigma, 1)$ would imply that the subtree rooted at t encodes a blockline l such that the first block of l induces a $(\sigma, 1, n)$-blockline and the second one induces a $(\sigma, 1, m)$-blockline. But as level 1 links always point to the preceding letter, n and m are equal to the length of the prefix of l in the stack plus 1, i.e., if T encodes a stack s then $s = s_1 : [w \setminus l] : s_2$ and $n = m = |w| + 1$. This contradicts the fact that every subtree encodes a maximal block as the first and the second block of l together would form one (bigger) block of l.

Now we want to show that $Q \times \mathrm{Stck}_2(\Sigma)$ is in bijection with $\mathbb{T}_{\mathrm{Enc}}$. For this purpose we define the inverse of Enc on $\mathbb{T}_{\mathrm{Enc}}$ explicitly. Thus, we have to restore the collapse-links of the letters encoded in the tree. Let $T \in \mathbb{T}$ be a $(\Sigma \times \{1,2\}) \cup \{\varepsilon\}$-labelled tree. Then we use the following auxiliary function

$$f_g^h(t,T) := \begin{cases} (T(t), h) & \text{if } T(t) \in \Sigma \times \{1\} \\ (T(t), g) & \text{if } T(t) \in \Sigma \times \{2\} \\ \varepsilon & \text{if } T(t) = \varepsilon \end{cases}$$

for $h, g \in \mathbb{N}$. h and g stand for the height and the width, respectively, of the stack decoded so far. We omit the parameter T if the tree is clear from the context.

Definition B.7. Define $\mathrm{Dec} : \mathbb{T} \times \mathbb{N} \times \mathbb{N} \to (\Sigma \times \{1,2\} \times \mathbb{N})^{*2}$ recursively as

$$\mathrm{Dec}(T,h,g) = \begin{cases} f_g^h(\varepsilon) & \text{if } \mathrm{dom}(T) = \{\varepsilon\} \\ f_g^h(\varepsilon) \setminus \mathrm{Dec}(T_0, h+1, g) & \text{if } 1 \notin \mathrm{dom}(T) \\ f_g^h(\varepsilon) \setminus (\varepsilon : \mathrm{Dec}(T_1, h, g+1)) & \text{if } 0 \notin \mathrm{dom}(T) \\ f_g^h(\varepsilon) \setminus (\mathrm{Dec}(T_0, h+1, g) : \mathrm{Dec}(T_1, h, g + G(T_1))) & \text{otherwise,} \end{cases}$$

where $G(T) := |\mathrm{Dec}(T, 0, 0)|$ is the width of the stack encoded in T. For a tree $T \in \mathbb{T}_{\mathrm{Enc}}$, we set its decoding to be

$$\mathrm{Dec}(T) := (T(\varepsilon), \mathrm{Dec}(T_0, 0, 0)) \in Q \times (\Sigma \times \{1,2\} \times \mathbb{N})^{+2} \ .$$

We are now going to prove that Dec is injective on $\mathbb{T}_{\mathrm{Enc}}$. Afterwards we prove that $\mathrm{Dec} \circ \mathrm{Enc}$ is the identity on the set of all configurations. This implies that Dec is a surjective map from $\mathbb{T}_{\mathrm{Enc}}$ to the configurations as the image of Enc is a subset of $\mathbb{T}_{\mathrm{Enc}}$ by definition.

Lemma B.8. *The function* Dec *is injective on* $\mathbb{T}_{\mathrm{Enc}}$.

Proof. Assume that there are trees $T', U' \in \mathbb{T}_{\mathrm{Enc}}$ with $\mathrm{Dec}(T') = \mathrm{Dec}(U')$. Then the labels of their roots coincide as these define the state of the configuration. Furthermore, we only have to compare the subtrees rooted at 0, i.e., $T := T'_0$ and $U := U'_0$. From our assumption it follows that $\mathrm{Dec}(T, 0, 0) = \mathrm{Dec}(U, 0, 0)$. Now we proceed by induction on the depth of the trees U and T. If both trees have depth 0 they are uniquely determined by the label of their root. But then $\mathrm{Dec}(U, n, m) = \mathrm{Dec}(T, n, m)$ if and only if the labels of the roots agree because the resulting stack is the label of the root plus a link which is n or m depending on the link-level of the label of the roots.

We assume that for all trees $\bar{U}, \bar{T} \in \mathbb{T}_{\mathrm{Enc}}$ such that $\mathrm{dp}(\bar{U}) \leq \mathrm{dp}(\bar{T}) \leq k$ the function Dec is injective. Let $\mathrm{dp}(U) \leq \mathrm{dp}(T) \leq k+1$. Heading for a contradiction, assume that $U \neq T$ but $\mathrm{Dec}(U, n, m) = \mathrm{Dec}(T, n, m)$. There are the following cases.

(1) If $U(\varepsilon) \neq T(\varepsilon)$ then the decodings cannot agree for the same reasons as in the initial step.
(2) We may assume that there is some $\rho \in (\Sigma \times \{1,2\}) \cup \{\varepsilon\}$ such that $U = \rho(U_0, U_1)$ and $T = \rho(T_0, T_1)$. We distinguish the following sub-cases
    (a) If $U_0 = \emptyset$ and $T_0 \neq \emptyset$ then $\mathrm{Dec}(U, n, m) = f_m^n(\rho) : u'$ for some level 2 word u' and $\mathrm{Dec}(T, n, m) = f_m^n(\rho) \setminus f_m^{n+1}(T(0)) \setminus t'' : t'$ for some level 2 words t' and $t'' \neq \emptyset$. But by definition of $\mathbb{T}_{\mathrm{Enc}}$ $f_m^{n+1}(T(0)) \neq \varepsilon$ which leads to the fact that the two stacks disagree on the first word.

(b) If $U_1 = \emptyset$ and $T_1 \neq \emptyset$ then $\mathrm{Dec}(U, n, m) = f^n_m(\rho) \setminus \mathrm{Dec}(U_0, n+1, m)$ and $\mathrm{Dec}(T, n, m) = f^n_m(\rho) \setminus \big(\mathrm{Dec}(T_0, n+1, m) : \mathrm{Dec}(T_1, n, m+m')\big)$ for some $m' \in \mathbb{N} \setminus \{0\}$. if $U_0 = T_0 = \emptyset$ then the two stacks disagree in their number of words. Hence we may assume that

$$\mathrm{Dec}(U_0, n+1, m) = (\sigma, l, k) \setminus u'$$

for some level 2 word u'. Note that $l = 2$ implies $k = m$. But for all letters in $t' := \mathrm{Dec}(T_1, n, m+m')$ of link level 2 the link is greater or equal to $m + m'$. Hence, t' does not contain any symbol $(\sigma, 2, m)$ which contradicts the equality of the decodings of U and T. Thus, we have to assume $l = 1$ and $k = n+1$. But then $\mathrm{Dec}(T_0, n+1, m) = (\sigma, 1, n+1) \setminus t'$ and $\mathrm{Dec}(T_{10}, n+1, m+m') = (\sigma, 1, n{+}1) \setminus t''$ for some nonempty level 2 words t' and t''. But this implies that the labels of 0 and of 10 in T are both $(\sigma, 1)$ which contradicts the definition of $\mathbb{T}_{\mathrm{Enc}}$.

(c) The last case that we have to consider is $U_0 \neq \emptyset$, $U_1 \neq \emptyset$, $T_0 \neq \emptyset$, $T_1 \neq \emptyset$, and $U \neq T$. Then we have

$$\mathrm{Dec}(U, n, m) = f^n_m(\rho) \setminus \big(\mathrm{Dec}(U_0, n+1, m) : \mathrm{Dec}(U_1, n, m+m')\big) \text{ and}$$
$$\mathrm{Dec}(T, n, m) = f^n_m(\rho) \setminus \big(\mathrm{Dec}(T_0, n+1, m) : \mathrm{Dec}(T_1, n, m+m'')\big)$$

for some natural numbers $m', m'' > 0$. By induction hypothesis, the decodings of the subtrees cannot agree. Hence, without loss of generality we may assume that $\mathrm{Dec}(U_0, n{+}1, m) = \mathrm{pop}_2{}^z\big(\mathrm{Dec}(T_0, n{+}1, m)\big)$ for some $z > 0$. But then the first letter in the first word of $\mathrm{Dec}(U_1, n, m{+}m')$ has to agree with the first letter of the words in $\mathrm{Dec}(U_0, n+1, m)$. But this leads to the same contradictions as in the previous case.

∎

As the next step we prove that Dec is a surjective map from $\mathbb{T}_{\mathrm{Enc}}$ to the set of configurations. This is done by induction on the blocklines used to encode a stack. In this proof we use the notion of left-maximal blocks and blocklines. Let $s : w \setminus (w' : b) : s'$ be a stack where s and s' are level 2 words, w, and w' are words, and b is a (σ, l, k)-block. We say b is left maximal in this stack if either $b = (\sigma, l, k)$ or if $b = (\sigma, l, k)(\tau, l', k') \setminus b'$ and w' does not start with $(\sigma, l, k)(\tau, l', k')$. We call a blockline in some stack good, if its first block is left maximal. Furthermore, we call the blockline starting with the block b left maximal if w' does not start with (σ, l, k). Note that the encoding of stacks works on left maximal blocks and good blocklines.

Lemma B.9. *For arbitrary* $c \in Q \times \mathrm{Stck}_2(\Sigma)$*, we have* $\mathrm{Dec}(\mathrm{Enc}(c)) = c$.

Corollary B.10. Dec *is a surjective map from* $\mathbb{T}_{\mathrm{Enc}}$ *to* $Q \times \mathrm{Stck}_2(\Sigma)$.

Proof of Lemma. As Dec and Enc encode and decode the state of c in the root of $\mathrm{Enc}(c)$, it suffices to show that

$$\mathrm{Dec}(\mathrm{Enc}(s), 0, 0) = s$$

for arbitrary $s \in \mathrm{Stck}_2(\Sigma)$. We proceed by induction on blocklines of the stack s. For this purpose we reformulate the lemma in the following claim.

Claim. Let s' be some stack which decomposes as $s' = s'' : [w \setminus b] : s'''$ such that $b \in (\Sigma \times \{1, 2\} \times \mathbb{N})^{+2}$ is a good (σ, l, k)-blockline. Then

(1) $\mathrm{Dec}(\mathrm{Enc}(b,\varepsilon),|w|,|s''|)=b'$ for the unique b' such that $b=(\sigma,l,k)\setminus b'$ and
(2) if b is left maximal, then $\mathrm{Dec}(\mathrm{Enc}(b,(\sigma,l)),|w|,|s''|)=b$.

Note that for the second part of the claim we have $k=|s''|$ if $l=2$ and $k=|w|$ if $l=1$. The lemma follows from the second part of the claim as every stack is a left maximal $(\perp,1,0)$-blockline.

We prove both claims in parallel by induction on the size of b. As abbreviations we set $h:=|w|$ and $g:=|s''|$. The arguments for the first claim are as follows.

- If $b=[(\sigma,l,k)]$ the claim is trivially true, as $\mathrm{Enc}(b,\tau)$ only consists of a root without successors for any τ.
- If there are $b_1,b_1'\in(\Sigma\times\{1,2\}\times\mathbb{N})^{*2}$ such that
$$b=[(\sigma,l,k)]:b_1=[(\sigma,l,k)]:\big((\sigma,l,k)\setminus b_1'\big)\text{ then}$$
$$\begin{aligned}&\mathrm{Dec}(\mathrm{Enc}(b,\varepsilon),h,g)=\mathrm{Dec}(\varepsilon(\emptyset,\mathrm{Enc}(b_1,\varepsilon)),h,g)\\&=f_g^h(\varepsilon)\setminus[\varepsilon:\mathrm{Dec}(\mathrm{Enc}(b_1,\varepsilon),h,g+1)]\\&\overset{(1)}{=}\varepsilon\setminus[\varepsilon:b_1']=\varepsilon:b_1'=b'\ .\end{aligned}$$
- Assume there is some $(\sigma',l',k')\in\Sigma\times\{1,2\}\times\mathbb{N}$ and some $b_1\in(\Sigma\times\{1,2\}\times\mathbb{N})^{*2}$ such that
$$b=(\sigma,l,k)(\sigma',l',k')\setminus b_1\ .$$
Then by assumption on b, the blockline $(\sigma',l',k')\setminus b_1$ is maximal whence
$$\begin{aligned}&\mathrm{Dec}(\mathrm{Enc}(b,\varepsilon),h,g)=\mathrm{Dec}(\varepsilon(\mathrm{Enc}((\sigma',l',k')\setminus b_1,(\sigma',l')),\emptyset),h,g)\\&=f_g^h(\varepsilon)\setminus\mathrm{Dec}(\mathrm{Enc}((\sigma',l',k')\setminus b_1,(\sigma',l')),h+1,g)\\&\overset{(2)}{=}(\sigma',l',k')\setminus b_1=b'\ .\end{aligned}$$
- The last case is that
$$b=(\sigma,l,k)\setminus[((\sigma',l',k')\setminus b_1):b_2]$$
for b_2 a blockline of s such that b_2 is not of the form $b_2\neq(\sigma',l',k')w':b_2'$. Our assumption on b implies that $(\sigma',l',k')\setminus b_1$ is a left maximal blockline. Thus,
$$\begin{aligned}&\mathrm{Dec}(\mathrm{Enc}(b,\varepsilon),h,g)\\&=\mathrm{Dec}(\varepsilon(\mathrm{Enc}((\sigma',k',l')\setminus b_1,(\sigma',l')),\mathrm{Enc}((\sigma,l,k)\setminus b_2,\varepsilon)),h,g)\\&=f_g^h(\varepsilon)\setminus[\mathrm{Dec}(\mathrm{Enc}((\sigma',l',k')\setminus b_1,(\sigma',l')),h+1,g):\mathrm{Dec}(\mathrm{Enc}((\sigma,l,k)\setminus b_2,\varepsilon),h,g+f)],\end{aligned}$$
$$\text{where }f=|\mathrm{Dec}(\mathrm{Enc}((\sigma',l',k')\setminus b_1,(\sigma',l')),h+1,g)|\overset{(2)}{=}|b_1|\ .$$
From this, we obtain
$$\begin{aligned}&\mathrm{Dec}(\mathrm{Enc}(b,\varepsilon),h,g)\\&=\varepsilon\setminus\big(((\sigma',l',k')\setminus b_1):\mathrm{Dec}(\mathrm{Enc}((\sigma,l,k))\setminus b_2,\varepsilon),h,g+f)\big)\\&\overset{(1)}{=}((\sigma',l',k')\setminus b_1):b_2\end{aligned}$$
as $(\sigma,l,k)\setminus b_2$ is a good blockline of s.

For the proof of the second claim, note that the calculations are basically the same, but $f_g^h(\varepsilon)$ is replaced by $f_g^h(\sigma, l)$ We claim that in this case $f_g^h(\sigma, l) = (\sigma, l, k)$. For $l = 1$ this is always true as level 1 links always point to the length of their prefix in the stack which is h. In the case that $l = 2$, $h = k = |s''|$ due to the left maximality of b. Thus, we may do the same calculations as in the first part and observe that every stack is prefixed by (σ, l, k).

∎

Appendix C. Recognising Reachable Configurations

The aim of this section is to develop the automata-recognisability of the valid, i.e., the reachable configurations (q, s) of a given 2-CPS S. For this purpose we introduce *milestones* of a stack s. These milestones are exactly those substacks of s that every run to s has to visit. It turns out that milestones are represented by the nodes of the encoding of s: for every element $d \in \mathrm{Enc}(s)$ the set of lexicographically smaller elements in $\mathrm{Enc}(s)$ form the encoding of a milestone of s. In the same sense, every milestone has such a corresponding element $d \in \mathrm{Enc}(s)$. It turns out that the order of the milestones induced by the substack relation is isomorphic to the lexicographical order of the elements of $\mathrm{Enc}(s)$. Furthermore, it turns out that the last occurrence of a milestone m_1 in a run to s is before the last occurrence of another milestone m_2 if and only if m_1 is a substack of m_2. We are going to show that we can guess for every $d \in \mathrm{Enc}(s)$ the state in which the corresponding milestone is visited for the last time by some run to s and that we can check the correctness of this guess using MSO or, equivalently, tree-automata.

We prove the fact that we can check the correctness of such a guess by introducing a special type of run, called *loop*. A run from one milestone to the next will mainly consist of loops combined with a finite number of stack operations.

We start by giving a precise notion of a loop and we analyse their occurrences.

C.1. Loops

A *loop* is a run from a stack s to the same stack s again that never "looks into" $\mathrm{pop}_2(s)$ and is only allowed to visit ${\mathrm{pop}_1}^n(s)$ if the topmost n symbols of s are of link-level 1. Such *loops* decompose into minimal loops which then divide into two groups: *high loops* are runs that start and end with some stack s and in between never read information below s, i.e., they never visit a substack of s (this implies not to visit s in between). *Low loops* are runs that start and end with a stack s, the first operation is a pop_1 and they never visit a substack of $\mathrm{pop}_2(s)$. For low loops we require that $\mathrm{CLvl}(s) = 1$.

Definition C.1. Let r be a run of length $\ln(r) = n$ and set $(q_i, s_i) := r(i)$. r is called a *simple high loop* if $s_0 = s_n$ and if s_0 is a proper substack of s_i for all $0 < i < n$.

The run r is called a *simple low loop* of s if $s_0 = s_n = s$, between 0 and n the stack s is never visited, $s_1 = \mathrm{pop}_1(s)$, $\mathrm{CLvl}(s) = 1$, $|s_i| \geq |s|$ for all $r(i) = (q_i, s_i)$, and $r{\restriction}_{[2,\ln(r)-1]}$ is the composition of simple low loops and simple high loops of $\mathrm{pop}_1(s)$.

We define a *loop* as a finite composition of simple low loops and simple high loops in arbitrary order.

We also define *returns* which are runs from a stack $s : w$ to s without visiting substacks of s. Furthermore, we require that they are independent of the level 2 collapse links stored in the word w. Equivalently, we want that returns from $s : w$ to s are independent of s in the sense that if there is a return from $s : w$ to s then there is a return from $s' : w$ to s' for all 2-words s'.

Our interest for returns stems from the fact that they appear as subruns of high loops whence they play an important role in finding loops for a given stack. Now we give the precise definition.

Definition C.2. Let $s' = s : w$ be some stack. A return from s' to s is a run r from s' to s such that r never visits a substack of s at $i < \ln(r)$ and such that one of the following hold:

(1) the last operation in r is pop_2,
(2) the last operation in r is a collapse and $w < \mathrm{top}_2(r(\ln(r) - 1))$, i.e., r pushes at first some new letters onto s' and then performs a collapse of one of these new letters, or
(3) the last operation in r is a collapse and there is some $i \in \mathrm{dom}(r)$ such that $r{\restriction}_{[i,\ln(r)]}$ is a return from $\mathrm{pop}_1(s')$ to s.

Returns and high loops are closely linked. Every return from a valid configuration $(q, s : w)$ occurs as subrun of a loop of the stack s. Every run r to $(q, s : w)$ passes s at some position i for the last time. Now compose $r{\restriction}_{[i,\ln(r)]}$ with the return. The resulting run is a loop of s. On the other hand returns play a crucial role in understanding the existence of high loops. Assume that r is a loop of the stack s and set $w := \mathrm{top}_2(s)$. It may happen that r visits at i some stack $s' : w$ such that s is a substack of s'. As r finally reaches s again it also has to pass s'. Restricting r to the interval between i and the first occurrence of s' again, we obtain a return from $s' : w$ to s'.

As loops and returns starting at some stack s never "look into" the information below the topmost word of s, the existence of a loop starting at some stack only depends on the topmost word w of this stack. Furthermore, we defined returns and loops in such a way that they do not use the level 2 collapse-links stored in w. Hence the existence of a loop only depends on the topmost word and is independent of the level 2 links of the elements in this word. The next lemma states an exact version of this claim.

Definition C.3. The word $w{\downarrow_0}$ is obtained from $w \in (\Sigma \times \{1, 2\} \times \mathbb{N})^*$ by replacing every occurrence of $(\sigma, 2, j)$ by $(\sigma, 2, 0)$ for all $\sigma \in \Sigma$ and all $j \in \mathbb{N}$.

Lemma C.4. *Let $(q, s) \in \mathrm{Grph}(A)$ be a configuration of a* CPG *with $\mathrm{top}_2(s) = w$. There is a loop r from (q, s) to (q', s) if and only if there is a loop r' from $(q, w{\downarrow_0})$ to $(q', w{\downarrow_0})$.*

Proof. Let $s = s_1 : w$. Assume r is a loop from (q, s) to (q', s). Then we define r' starting with $r'(0) := (q, w{\downarrow_0})$. We now define r' step by step on $\mathrm{dom}(r)$ such that for $r(i) = (q_i, t_i)$ and $r'(i) = (q_i, t'_i)$ we preserve the following:

- $|t_i| - |s_1| = |t'_i|$
- $\big(\mathrm{top}_2(t_i)\big){\downarrow_0} = \big(\mathrm{top}_2(t'_i)\big){\downarrow_0}$

Assume r' is defined up to $i < \ln(r)$ such that $r(i + 1) = (q_{i+1}, \mathrm{op}(t_i))$ for $\mathrm{op} \in \mathrm{OP}$ then we set $r'(i + 1) := (q_{i+1}, \mathrm{op}(t'_i))$. This defines a run r' as the topmost symbols and the states of $r(i)$ and $r'(i)$ agree. Furthermore r never uses a collapse-operation when the topmost word of t_i is a prefix of w because $|t_{i+1}| \geq |s|$. The other direction works by the same arguments: every operation that may be performed at $r'(i)$ can also be performed at $r(i)$. ∎

As returns and loops are closely linked, a similar result holds for returns:

Lemma C.5. *Let $(q, s : w) \in \mathrm{Grph}(S)$ for nonempty s be a configuration of some* CPG. *There is a return r from $(q, s : w)$ to (q', s) if and only if there is a return r' from $(q, w{\downarrow_0}\Box : w{\downarrow_0})$ to $(q', w{\downarrow_0}\Box)$ for some new stack symbol $\Box$ such that S stops whenever the topmost symbol is $\Box$.*

The proof is analogously to the loop case.

Now, we want to prove that there is an algorithm that checks the existence of certain loops, respectively, returns for all stacks: reading the topmost word of the stack letter by letter it determines all possible pairs of initial and final states of loops, respectively returns of this stack. From this we obtain that for a fixed CPS S there is a finite automaton that reads a word $w \in (\Sigma \times \{1, 2\} \times \mathbb{N})^*$ and calculates the possible initial and final states of loops and returns of stacks with topmost word w. Before we state the lemma, we have to introduce some notation.

Definition C.6. We denote as $\mathrm{Loops}(w)$ the set of possible state combinations in a loop of a word w, i.e., $\mathrm{Loops}(w) \subseteq Q \times Q$ such that

$$(q_1, q_2) \in \mathrm{Loops}(w) \quad \text{:iff} \quad \text{there is a loop from } (q_1, w{\downarrow_0}) \text{ to } (q_2, w{\downarrow_0}) \ .$$

Analogously, we denote as $\mathrm{Returns}(w)$ the set of possible state combinations in a return from some stack with topmost word w, i.e., $\mathrm{Returns}(w) \subseteq Q \times Q$ such that

$$(q_1, q_2) \in \mathrm{Returns}(w) \text{ :iff } \text{there is a return from } (q_1, w{\downarrow_0}\Box : w{\downarrow_0}) \text{ to } (q_2, w{\downarrow_0}\Box) \ .$$

Lemma C.7. *For all stacks s with $s' := \mathrm{pop}_1(s)$ defined,* $\mathrm{Returns}(\mathrm{top}_2(s'))$, $\mathrm{CLvl}(s)$, *and* $\mathrm{Sym}(s)$ *effectively determine* $\mathrm{Returns}(\mathrm{top}_2(s))$.

Proof. For the calculation of $\mathrm{Returns}(\mathrm{top}_2(s))$ we make the following observation about a return r from (q_1, s) to $(q_2, \mathrm{pop}_2(s))$.

If there is an i such that $r(i) = (q_i, s_i)$ with $w := \mathrm{top}_2(s_i) = \mathrm{top}_2(s')$, then there is a minimal $j > i$ such that $r(j) = (q_j, \mathrm{pop}_2(s_i))$ for some $q_j \in Q$. Hence, r restricted to $[i, j]$ is a return from s_i to $\mathrm{pop}_2(s_i)$. As we have seen before, this return exists if and only if $(q, q') \in \mathrm{Returns}(\mathrm{top}_2(s'))$.

This leads to the following decision process for $(q_1, q_2) \in \mathrm{Returns}(\mathrm{top}_2(s))$ for a pushdown system $S = (Q, \Sigma, q_0, \Delta)$. While calculating returns of a word w, we may replace the prefix $\mathrm{pop}_1(w)$ by a new stack symbol $\top$. From the considerations above we know that any return of w that reaches a configuration (q, s) with $s = t : \mathrm{pop}_1(w)$ for some stack t goes on with a return from s to t. After performing this return w is again a prefix of the topmost word of the stack. Now we simulate a return from $(q_1, t : \mathrm{pop}_1(w))$ to (q_2, t) by adding a transition $(q_1, \top, q_2, \mathrm{pop}_2)$. Then any return from some stack $t : \mathrm{pop}_1(w)$ to t is simulated by performing one pop_2-operation on $(\bot\top \setminus t') : [\bot\top]$ where t' is defined such that $t = w \setminus t'$.

The details are as follows. We set $\bar{S} := (\bar{Q}, \Sigma \cup \{\top, \Box\}, \bar{q}_0, \bar{\Delta})$ for $Q \subseteq \bar{Q}$, with a new initial state $\bar{q}_0 \in \bar{Q}$ and with $\Delta \subseteq \bar{\Delta}$. $\bar{\Delta}$ extends Δ by:

(1) transitions that initialise the automaton with the configuration (q_1, u) where

$$u := [\bot\top(\mathrm{Sym}(s), \mathrm{CLvl}(s), k)\Box] : [\bot\top(\mathrm{Sym}(s), \mathrm{CLvl}(s), k)],$$

where $k = 2$ if $\mathrm{CLvl}(s) = 1$ and $k = 0$ otherwise, and $\bot$ stands for $(\bot, 1, 0), \top$ for $(\top, 1, 1)$, and $\Box$ for $(\Box, 1, 3)$. $\Box$ is a new symbol that just stops the calculation of the pushdown system, while $\top$ represents the word $\mathrm{top}_2(s')$.

(2) for every $(q, q') \in \mathrm{Returns}(\mathrm{top}_2(s'))$ the transition $(q, \top, q', \mathrm{pop}_2)$.

From the observations we made before, we see that $(q_1, q_2) \in \text{Returns}(\text{top}_2(s))$ if and only if there is a return from $(q_1, s : \text{top}_2(s))$ to (q_2, s) in S if and only if there is a return from (q_1, u) to $(q_2, \text{pop}_2(u))$ in $\bar{S}$. But this problem is equivalent to the existence of $(q_2, \text{pop}_2(u))$ in $\text{Grph}(\bar{S})$ which we have already shown to be decidable. ∎

Lemma C.8. *Let s be a stack such that $s' := \text{pop}_1(s)$ is defined. Then* $\text{CLvl}(s)$, $\text{Sym}(s)$, $\text{Loops}(\text{top}_2(s'))$, $\text{Returns}(\text{top}_2(s'))$, *and* $\text{Sym}(s')$ *effectively determine* $\text{Loops}(\text{top}_2(s))$.

Proof. For the calculation of $\text{Loops}(\text{top}_2(s))$, we focus at first on a simple low loop r from (q_1, s) to (q_2, s). Note that r only exists if $\text{CLvl}(s) = 1$ by definition of a low loop. Then there is some $q' \in Q$ such that $r(1) = (q', s')$ with $(q_1, \text{Sym}(s), q', \text{pop}_1) \in \Delta$. Furthermore, we know that for $n := \text{ln}(r)$ we have $r(n-1) = (q'', s')$ and $r{\restriction}_{[1,n-1]}$ is a loop of $\text{top}_2(s')$.

Secondly, we now analyse a simple high loop r from (q_1, s) to (q_2, s). If there is an $i \in \text{dom}(r)$ such that $r(i) = (q_i, s_i)$ with $|s_i| > |s|$ and $w := \text{top}_2(s_i) = \text{top}_2(s')$, then there is a minimal $j > i$ such that $r(j) = (q_j, \text{pop}_2(s_i))$ for some $q_j \in Q$. Hence, r restricted to $[i, j]$ is a return from s_i to $\text{pop}_2(s_i)$. As we have seen before, this return exists if and only if $(q, q') \in \text{Returns}(\text{top}_2(s'))$.

These observations lead to the following decision process whether there is a loop from (q_1, s) to (q_2, s): let $S = (Q, \Sigma, q_0, \Delta)$. We set $\bar{S} := (\bar{Q}, \Sigma \cup \{\top\}, \bar{q}_0, \bar{\Delta})$ for $Q \subseteq \bar{Q}$ with a new initial state $\bar{q}_0 \in \bar{Q}$ and $\Delta \subseteq \bar{\Delta}$. $\bar{\Delta}$ extends Δ by

(1) transitions that initialise the automaton in the configuration (q_1, u) where $u := (\bot, 1, 0)(\top, 1, 1)(\text{Sym}(s), \text{CLvl}(s), l)$ and $l = 2$ if $\text{CLvl}(s) = 1$ and $l = 0$ otherwise.
(2) for every $(q, q') \in \text{Returns}(\text{top}_2(s'))$ the transition $(q, \top, q', \text{pop}_2)$.
(3) for every $(q, q') \in \text{Loops}(\text{top}_2(s'))$ the transition $(q, \top, q', \text{id})$.
(4) If $\text{CLvl}(s) = 1$, then we add for every $(q, \text{Sym}(s'), q', \text{push}_{\text{Sym}(s),1}) \in \Delta$ the transition $(q, \top, q', \text{push}_{\text{Sym}(s),1})$.

From the observations about simple high and low loops, we see that $\bar{S}$ simulates any loop on s by simulating returns and loops of $\text{top}_2(\text{pop}_1(s))$ in one step (see Figure 6 and 7). Hence there is a loop from (q_1, s) to (q_2, s) in S if and only if there is a loop from (q_1, u) to (q_2, u) in $\bar{S}$. But this problem is equivalent to the existence of (q_2, u) in $\text{Grph}(\bar{S})$ which we have already shown to be decidable. ∎

C.2. Milestones

Now we define milestones of a stack s, which turn out to be those substacks of s that have to be visited by any run from the initial configuration to s. It is a simple observation that for $s = w_1 : \ldots : w_i : \ldots : w_n$ every run from the initial configuration to s has to visit $w_1 : \ldots : w_i$. This is due to the fact that the only possibility to create new words on the stack is the clone_2-operation. But there are more substacks that have to be visited on such a run.

Definition C.9 (milestone)**.** A substack s' of $s = w_1 : w_2 : \ldots : w_n$ is a *milestone* if $s' = w_1 : w_2 : \ldots : w_i : w'$ such that $w_i \sqcap w_{i+1} \leq w' \leq w_{i+1}$.

We denote as $\text{MS}(s)$ the set of milestones of s.

Remark C.10. The substack relation induces a linear order on the milestones of a given stack s.

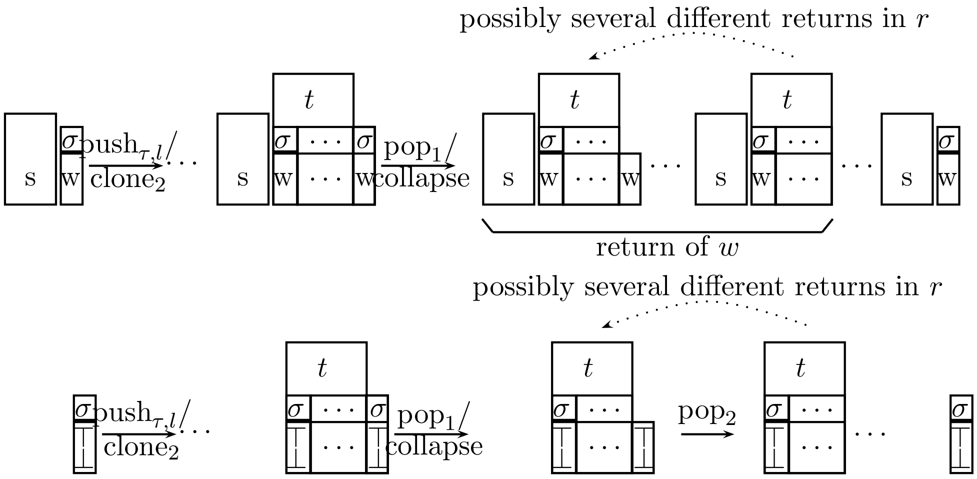

Figure 6: Simple high loop r in S (upper picture) and its simulation in $\bar{S}$ (lower picture) (see. Lemma C.8)

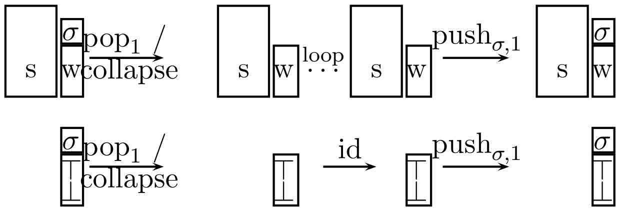

Figure 7: Simple low loop r in S (upper picture) and its simulation is $\bar{S}$ (lower picture) (see. Lemma 6).

This definition fits our informal definition in terms of necessarily reached substacks because of the following lemma:

Lemma C.11. *Let s_1, s_2 be stacks and s a milestone of s_2 but no substack of s_1. Then any run from s_1 to s_2 passes s.*

Proof. Without loss of generality, we may assume that $s_1 \leq s \leq s_2$ as every run from s_1 to s_2 must pass a common substack of both stacks. If $s = s_2$, there is nothing to show. Now assume that the lemma holds for all milestones of s_2 that contain s as a proper substack. Hence, we may restrict r to an initial part that ends with the successor s' of s in $\mathrm{MS}(s_2)$. There are the following cases.

- For $s = \mathrm{pop}_2(s')$, it is clear that r passes a common substack of s_1 and s'. This substack must also be a substack of s. Hence, there is some $n \in \mathrm{dom}(r)$ such that $|r(n)| \leq |s|$. Now let m be the maximal element such that $|r(m)| = |s| < |s'|$. But then the stack at $r(m)$ is s.
- Otherwise $s = \mathrm{pop}_1(s')$. Note that in this case the greatest common prefix of the last two words in s' are a proper prefix of $\mathrm{top}_2(s')$. In this case s' is not reachable via clone_2 and pop_1 operations from any of its substacks. Hence, at some point s' is created by r via a $\mathrm{push}_{\sigma,i}$-operation. But then the previous stack was s.

■

Corollary C.12. *For an arbitrary run r from the initial configuration to s, the function*

$$f : \mathrm{MS}(s) \to dom(r)$$
$$s' \mapsto \max\{i \in dom(r) : r(i) = (q, s') \text{ for some } q \in Q\}$$

is an order embedding with respect to substack relation on the milestones and the natural order of $dom(r)$.

As already indicated, there is a close correspondence between milestones of a stack s and the elements of the tree $\mathrm{Enc}(s)$. The next definition associates to every element of $\mathrm{Enc}(s)$ a stack which turns out to be a milestone.

Definition C.13. Let $T \in \mathbb{T}_{\mathrm{Enc}}$ be a tree and $d \in T \setminus \{\varepsilon\}$. Then the *left and downward closed tree of* d is $LT(t, T) := T{\restriction_D}$ where $D := \{d' \in T : d' \leq_{lex} d\} \setminus \{\varepsilon\}$. Then we denote as $\mathrm{LStck}(t, T) := \mathrm{Dec}(LT(t, T), 0, 0)$ the *left stack induced by t*. If T is clear from the context, we omit it.

Remark C.14. Due to the definition of the encoding, for every $d \in \mathrm{dom}(\mathrm{Enc}(s))$, $w := \mathrm{top}_2(\mathrm{LStck}(d, s)){\downarrow_0}$ is determined by the path from the root to d: interpreting ε as empty word, the word along this path contains the pairs of stack symbols and collapse-link-levels of the elements in w. In every word $w{\downarrow_0}$ the links are determined by the link-levels in w.

Remark C.15. $\mathrm{LStck}(d, s)$ is a substack of s for all $d \in \mathrm{dom}(\mathrm{Enc}(s))$. This observation follows from Remark B.3 combined with the fact that the left stack is induced by a lexicographically downward closed subset.

The next lemma states the tight connection between milestones of a stack (with substack relation) and elements in the encoding of this stack (with lexicographic order).

Lemma C.16. *There is an order isomorphism $g : dom(\mathrm{Enc}(q, s)) \setminus \{\varepsilon\} \to \mathrm{MS}(s)$ given by $d \mapsto \mathrm{LStck}(d, s)$.*

We start the proof of the lemma with the following claim.

Claim. $\mathrm{LStck}(d, s) \in \mathrm{MS}(d)$ for every stack s and every $d \in \mathrm{dom}(\mathrm{Enc}(q, s)) \setminus \{\varepsilon\}$.

Proof of claim. From observation B.3 we know that for every $d = d'1$, the left-tree induced by d contains the first symbol of some block and all subtrees encoding blocks to the left of this block. This forms an encoding of a substack of s and actually of a milestone as the first symbol of a block always represents the last common symbol of two words in s that are next to each other. For the other case, assume that $d = d'0 \in \mathrm{dom}(\mathrm{Enc}(q, s))$. We set $s' := \mathrm{LStck}(d, s)$ and assume that this is a milestone of s. It is easily seen that $\mathrm{LStck}(d0, s)$ extends s' by one letter and the claim is clear from Remark B.3: if $|s'| = i$, then $\mathrm{top}_2(s')$ is a prefix of the i-th word of s. From the remark we obtain that $\mathrm{LStck}(d0, s)$ is s' extended by one further letter of the i-th word of s. Note that the greatest common prefix of the i-th and the $(i-1)$-st word of s is a prefix of $\mathrm{top}_2(s')$ whence it is also a prefix of $\mathrm{top}_2(\mathrm{LStck}(d0, s))$. ■

Proof of lemma. If the successor of d in lexicographic order is $d0$, then the left-stack of the latter extends the former by just one letter. Otherwise, the left-tree of the successor of d contains more elements ending in 1, whence it encodes a stack of larger width. As all these elements induce milestones, the lemma follows immediately.

■

We have seen that all milestones of some stack s are visited by a run r to s and their last occurrence in the run is ordered according to the substack order on the milestones. This implies that r decomposes as $r = l_0 \circ o_1 \circ l_1 \circ o_2 \circ l_2 \ldots \circ o_n \circ l_n$ where l_i is a loop and o_i is a run that starts at the last occurrence of the i-th milestone in r and ends at some occurrence of the $i+1$-th milestone. If the $i+1$-th milestone differs from the i-th by exactly one $\text{push}_{\sigma,k}$-operation, then o_1 only consists of one transition performing this operation. For the other case, we now analyse the form of o_i and show that it is a clone-operation followed by a sequence of loops and pop_1-operations.

Lemma C.17. *Let r be a run from some stack s to the stack $s' := \text{pop}_1{}^n(\text{clone}_2(s))$ that visits s only in the initial state. Then $r = o_0 \circ l_0 \circ o_1 \circ l_1 \circ o_2 \circ l_2 \ldots o_n \circ l_n$ where l_i are loops and o_0 is a run that only performs one clone-operation and for $j > 0$ the run o_j only performs a pop_1-operation.*

Proof. If r does not start with a clone-operation, then $|r(1)| \leq |s|$ whence it would pass s again. As r never passes s again, for each $s^m := \text{pop}_1{}^m(\text{clone}_2(s))$, $m \leq n$ the following hold: if $r(j) = (q, s^m)$ and $i \geq j$ such that s^m occurs at i for the last time, r restricted to the interval $[j, i]$ is a loop of s^m. Let $j-1$ be the last occurrence of s^{m-1}. The operation at s^{m-1} has to be a pop_1-operation as the final stack of r has a smaller $|s'|$-th word than s^{m-1} and by assumption we cannot remove this word from the stack via collapse or pop_2 as we never visit s again. ∎

C.3. Detection of Reachable Configurations

We have already seen that every run to a valid configuration (q, s) passes all the milestones of s. Now, we use the last states in which a run r to (q, s) visits the milestones as a certificate for the reachability of (q, s). To be precise, a certificate for the reachability of (q, s) by a run r is a map $C_r : \text{dom}(\text{Enc}(q, s)) \setminus \{\varepsilon\} \to Q$ such that $d \mapsto q$ if and only if $r(i) = (q, \text{LStck}(d))$ and i is the maximal position in r where $\text{LStck}(d)$ is visited.

Lemma C.18. *There is a tree-automaton that checks for any map*

$$f : \mathit{dom}(\text{Enc}(q, s)) \setminus \{\varepsilon\} \to Q$$

whether f is a certificate for some run r, i.e., whether $f = C_r$ for some run r from the initial configuration to (q, s).

Proof. First of all note that due to Remark C.14 and due to the inductive computability of the pairs of initial and final states of loops of some stack, the mapping that assigns $d \mapsto \text{Loops}(\text{LStck}(d, \text{Enc}(q, s)))$ is MSO-definable on $\text{Enc}(q, s)$.

Assume that $d, d0 \in \text{dom}(f)$. By definition, we know that f can only be a certificate if there is a run r' from $(f(d), \text{LStck}(d))$ to $(f(d0), \text{LStck}(d0))$ that starts with some push-operation followed by a loop of $\text{LStck}(d0)$. But as the pairs of initial and final states of loops of $\text{LStck}(d0)$ are MSO-definable, this requirement is MSO-definable for all such pairs d and $d0$.

Now we have to consider the case where $d \in \text{dom}(f)$ but $d0 \notin \text{dom}(f)$. This implies that the direct successor of $\text{LStck}(d)$ in the milestones is of the form $s' := \text{pop}_1{}^m(\text{clone}_2(\text{LStck}(d)))$. In this case there is a maximal prefix $d_0 \leq d$ such that $d = d_0 0 d_1$ and $d' := d_0 1 \in \text{dom}(f)$. Due to Lemma C.16, we know that $s' = \text{LStck}(d')$.

From the previous section, we know that we have to verify that there is some run $r' := o_0 \circ l_0 \circ o_1 \circ l_1 \circ o_2 \circ l_2 \dots o_m \circ l_m$ where l_i are loops and o_0 is a run that only performs one clone-operation and for $j > 0$ the run o_j only performs a pop_1-operation such that r' starts in $(f(d), \mathrm{LStck}(d))$ and ends in state $f(d')$.

An automaton can verify this because the path from d' to d encode the topmost stack symbols and link-levels of ${\mathrm{pop}_1}^{m'}(\mathrm{clone}_2(\mathrm{LStck}(d)))$ for $m' \leq m$. As the existence of loops only depends on the topmost word, an automaton can check the existence of r' while processing the path from d_0 to d.

∎

As a certificate of reachability is a function of finite range, it can be encoded in a finite set of subsets of its domain. Thus we obtain:

Corollary C.19. *For every 2-CPS S, there is a tree-automaton A that accepts a tree $T \in \mathbb{T}$ if and only if $T = \mathrm{Enc}(q, s)$ for a reachable configuration (q, s) of S.*

Appendix D. Automata Recognising Stack Operations

In the previous section, we have seen that the function Enc translates the reachable configurations of a 2-CPS into a regular tree language. In order to prove that $\mathrm{Grph}(S)$ is tree-automatic, we finally have to define tree-automata recognising the different transition relations. For this purpose we start with an analysis of the relationship between $\mathrm{Enc}(q, s)$ and $\mathrm{Enc}(q', \mathrm{pop}_2(s))$.

Lemma D.1. *Let s and s' be stacks of a pushdown system S such that $s' := \mathrm{pop}_2(s)$ and let t be the maximal element such that t is in the rightmost path of $\mathrm{Enc}(s')$ and $t1 \in dom(\mathrm{Enc}(s))$. For $D := \{d \in dom(\mathrm{Enc}(s))) : t1 \not\leq d\}$, it holds that*

$$dom(\mathrm{Enc}(s)) \setminus dom(\mathrm{Enc}(s')) \subseteq t10^* \text{ and}$$
$$\mathrm{Enc}(s', (\bot, 1)) = \mathrm{Enc}(s){\restriction_D} \text{ (see Figure 14).}$$

Proof. At first, note that for some stack consisting of just one word w_1, its encoding $\mathrm{Enc}(w_1, \varepsilon)$ is a path with left successor edges, i.e., $\mathrm{dom}(\mathrm{Enc}(w_1, \varepsilon)) \subseteq \{0\}^*$.

Let $s := w_1 : w_2 : \dots : w_n : w_{n+1}$ and correspondingly $s' := w_1 : w_2 : \dots : w_n$. Now we compare $\mathrm{Enc}(s, \rho)$ and $\mathrm{Enc}(s', \rho)$ for some $\rho \in \Sigma \times \{1, 2\}$ and distinguish the following cases.

1. For all $i \leq n, w_i = (\sigma_1, l_1, i_1)(\sigma_2, l_2, i_2)w'_i$, but $(\sigma_1, l_1, i_1)(\sigma_2, l_2, i_2) \not\leq w_{n+1}$: then the root in $\mathrm{Enc}(s', (\sigma, l))$ has only a left successor and $\mathrm{Enc}(s, (\sigma, l))$ extends $\mathrm{Enc}(s', \sigma)$ by a right subtree of the root which is $\mathrm{Enc}(w_{n+1}, \varepsilon)$. In this case the lemma holds.
2. For all $i \leq n{+}1, w_i = (\sigma_1, l_1, i_1)(\sigma_2, l_2, i_2)w'_i$: in this case $\mathrm{Enc}(s, (\sigma, l))$ and $\mathrm{Enc}(s', (\sigma, l))$ coincide on their root, they do not have right successors and the left successor is $\mathrm{Enc}((\sigma_2, l_2, i_2) \setminus (w'_1 : \dots : w'_n : w'_{n+1}))$ and $\mathrm{Enc}((\sigma_2, l_2, i_2) \setminus (w'_1 : \dots : w'_n))$, respectively.
3. There is some $j < n$ such that for all $i \leq j, w_i = (\sigma_1, l_1, i_1)(\sigma_2, l_2, i_2)w'_i$, but for $i > j$ we have $(\sigma_1, l_1, i_1)(\sigma_2, l_2, i_2) \not\leq w_i$: in this case the claim of the lemma reduces to the claim that the lemma holds for $s := w_{j+1} : w_{j+2} : \dots : w_{n+1}$ and $s' := w_{j+1} : w_{j+2} : \dots : w_n$.

(4) For $|w_1| = 1$. In this case the problem reduces to the claim that the lemma holds for $s := w_2 : w_3 : \ldots : w_{n+1}$ and $s' := w_2 : w_3 : \ldots : w_n$. Note that for $n = 1$ this leads to the fact that $\mathrm{Enc}(s', \sigma)$ is only a tree of one element and $\mathrm{Enc}(s, \sigma)$ extends this root by a right subtree, namely $\mathrm{Enc}(w_{n+1}, \varepsilon)$. In this case the lemma holds.

Eventually the claim always reduces to the first case or the last case and $n = 1$. ∎

As in the pop_2-case, one proves a similar result for the collapse-case with similar arguments:

Lemma D.2. *Let s, s' be configurations such that* $\mathrm{CLvl}(s) = 2$ *and* $s' := \mathrm{collapse}(s)$*. Let t' be the maximal element in the rightmost path of of* $\mathrm{Enc}(s)$ *which is labelled by some* $(\sigma, 2)$ *for $\sigma \in \Sigma$. Then let t be the maximal ancestor of t' such that $t1 \leq t'$. Then* $\mathrm{Enc}(s', (\bot, 1)) = \mathrm{Enc}(s){\restriction}_D$*, where $D := \{d \in dom(\mathrm{Enc}(s) : t1 \not\leq d\}$ (see Figure 13).*

With these lemmas we can now prove that the relations of CPG are tree-automatic.

Lemma D.3. *Let C be the set of Encodings of configurations of a* CPS *S. Then there are automata $A_{(q,\mathrm{op})}$ for $q \in Q$ and* $\mathrm{op} \in \mathrm{OP}$ *such that*

$$A_{(q,\mathrm{op})} \text{ accepts } \mathrm{Enc}(c_1) \otimes \mathrm{Enc}(c_2) \quad \text{iff} \quad c_1 \vdash^{(q,\mathrm{op})} c_2$$

for all $c_1, c_2 \in C$.

Proof. Whether c_2 has state q may easily be checked directly at the root of $\mathrm{Enc}(c_1) \otimes \mathrm{Enc}(c_2)$. As $c_1 \vdash^{(q,\mathrm{op})} c_2$ if and only if $c_1 \vdash^{(\mathrm{op})} c_2$ and $c_2 = (q, s)$ for some stack s, the remaining problem is to give automata which recognise the convolution of encodings of stacks that are related via a stack operation op. In order to prove this, we assume that the stack of c_i is s_i.

It is easy to see that if we obtain a stack s_2 by applying a $\mathrm{push}_{\sigma,2}$- or clone_2-transition to some configuration s_1 then $\mathrm{Enc}(s_1)$ and $\mathrm{Enc}(s_2)$ differ only in one node, which is the rightmost leaf of $\mathrm{Enc}(s_2)$ (cf. Figures 10 and 8). This can easily be checked by an automaton.

For the $\mathrm{push}_{\sigma,1}$- and pop_1-operation, recognisability of the operations is slightly more difficult as we have different cases. The different possibilities are depicted in the Figures 9 – 12. The different cases are uniquely determined by the rightmost leaves of $\mathrm{Enc}(s_1)$ and $\mathrm{Enc}(s_2)$: it depends on whether these leaves are right- or left-successors. As you can see from the figures, for each of the different cases, it is easy to define an automaton recognising convolutions of two trees which differ in such a local way. Furthermore, the case distinction whether the rightmost leaf is a left- or right-successor is clearly tree-automata recognisable. Hence, there are automata recognising the pop_1- and the $\mathrm{push}_{\sigma,1}$-transitions.

For the pop_2-case we use the lemma we just proved. An automaton recognising the pop_2-operation only has to guess the set D from Lemma D.1 and check whether the second tree is the restriction of the first tree to D. Note that the last element of D along the rightmost path may be guessed non-deterministically and then the automaton may check that its guess was right. Compare also figure 14.

For the collapse-case, we have to distinguish according to the level of the collapse. Either the level of the link of the first configuration is 1, then the collapse is a pop_1-operation, or it is of level 2 and then we may proceed as in the pop_2-case using the previous lemma. The only difference to the pop_2-case is the definition of the set D. As the case distinction only depends on the label of the greatest element in the rightmost path which is not labelled ε, both cases may be checked with one single automaton. Compare also figure 13.

∎

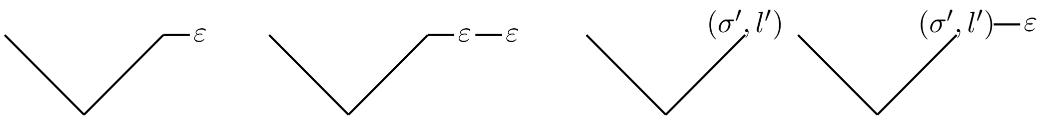

Figure 8: The two versions of clone_2-operations

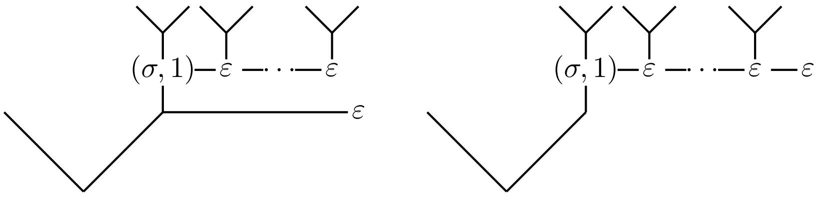

Figure 9: $\text{push}_{\sigma,1}$-operation with $\text{top}_2(s)(\sigma, 1, |\text{top}_2(s)|) \leq \text{top}_2(\text{pop}_2(s))$

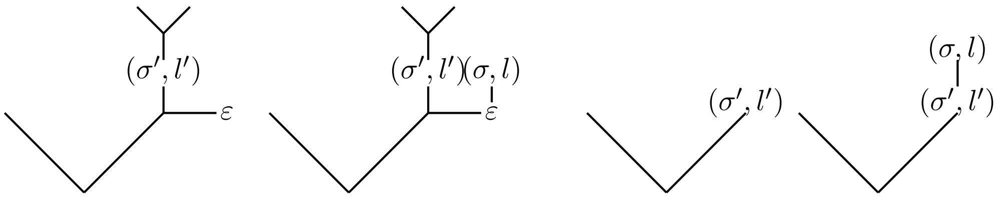

Figure 10: The two versions of $\text{push}_{\sigma,l}$-operation otherwise

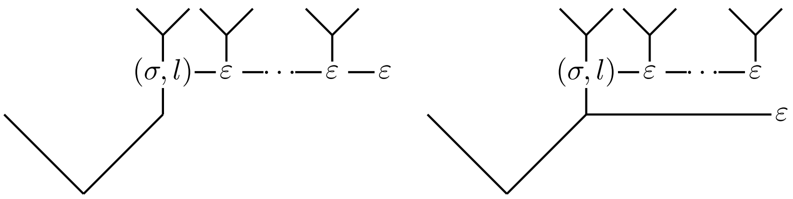

Figure 11: pop_1-operation on cloned element

Figure 12: pop_1-operation otherwise

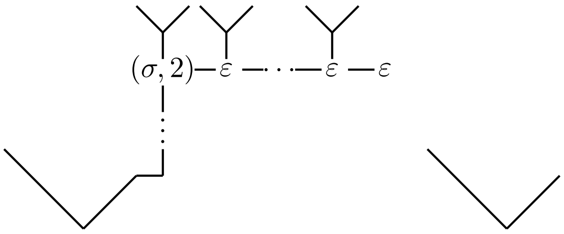

Figure 13: collapse-operation of level 2 (If the collapse is of level 1 then it is identical to the pop_1-operation).

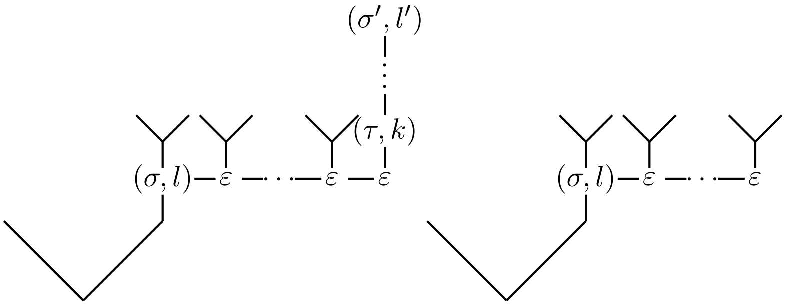

Figure 14: pop_2-operation

We have seen that for each 2-CPS the class of encodings of valid configurations of this system is a set of MSO-definable trees. Hence, this set is automata-recognisable. Furthermore, all operations of a collapsible pushdown system are tree-automata recognisable in this encoding. Putting these facts together we obtain our first main result.

Theorem D.4. *For every collapsible pushdown system S of order 2, the collapsible pushdown graph generated by S, $\text{CPG}(S)$, is tree-automatic.*

One obtains immediately the first-order decidability of level 2 CPG as all tree-automatic structures have decidable first-order theories.

Appendix E. Reachability in 2-CPG is Tree-Automatic

In this section we extend our result about the decidability of the first-order theories of 2-CPG to the decidability of first-order with reachability. We expand a CPG G by a new binary relation REACH that is true for two configurations c_1 and c_2 if there is a path from c_1 to c_2 in G. We start with an observation about the general form of such a path. Let $c_1 = (q_1, s_1)$ and $c_2 = (q_2, s_2)$, then for every run r from c_1 to c_2 there are configuration (q_3, s_3), (q_4, s_4), (q_5, s_5), positions $i_3 \leq i_4 \leq i_5$ in the run r, and numbers $m_2, m_3, m_5 \in \mathbb{N}$ such that the following hold:

(1) $r(i_3) = (q_3, s_3)$ and $s_3 = \text{pop}_2{}^{m_2}(s_1)$,
(2) $r(i_4) = (q_4, s_4)$, $s_4 = \text{pop}_1{}^{m_3}(s_3)$, and s_4 is a common substack of s_1 and s_2,
(3) $r(i_5) = (q_5, s_5)$, $s_5 = \text{pop}_2{}^{m_5}(s_2)$ and at the same time $s_4 = \text{pop}_1{}^{n_5}(s_5)$, and
(4) r does not visit any substack of $\text{pop}_2(s_3)$.

i_4 is easily found as the minimal substack of s_1 that is visited by r. The existence from i_5 follows directly from the fact that s_5 is a milestone of s_2. The existence of i_3 is clear from the fact that the run r first has to reach a stack of width $|s_4|$ before it can change the $|s_4|$-th word of the stack.

We decompose the proof that REACH is a tree-automatic predicate according to the decomposition of r along the intervals $[i_1, i_3], [i_3, i_4], [i_4, i_5]$, and $[i_5, i_2]$.

We start by an analysis of the first part: a run from a stack s to some substack $\mathrm{pop}_2{}^n(s)$. At a first glance one might think that such a run only consists of a sequence of loops and returns. But this is only true if we do not use the collapse-operation. A CPS might start by writing a lot of information with clone_2- and $\mathrm{push}_{\sigma,l}$-operations onto the stack, then use a couple of pop_1-operations to come to an element with a small collapse link and then use the collapse to jump to a very small substack of s without using any substack of s in between.

For this reason we introduce the notion of a *level-1-loop* which is a kind of loop of the topmost word but which increases the number of level 1 words on the level 2 stack. We prove that the pairs of initial and final states of these new loops are computable in a similar way as for the ordinary loops we introduced before. Furthermore, we show that every run from s to $\mathrm{pop}_2{}^n(s')$ decomposes mainly into parts that are basically returns, loops or 1-loops. These parts are connected by application of either a pop_1- or a collapse-operation.

Definition E.1. A run r is called a *level-1-loop* if for level 2 words s, s', where s' is nonempty, and a word w we have $r(0) = s : w$, $r(\ln(r)) = s : (w \setminus s') : w$ and for all $0 < i < \ln(r), r(i) = (q_i, s : s_i)$ for some nonempty level 2-word s_i such that whenever $\mathrm{top}_2(s_i) = \mathrm{pop}_1(w)$ then there is some $i < j < \ln(r)$ such that $r{\restriction}_{[i,j]}$ is a return.

For a fixed CPS S and some stack s we denote as $1\text{-Loops}_S(s)$ the set

$$\{(q_1, q_2) \in Q \times Q : \text{ there is } s' \in \mathrm{Stck}(\Sigma) \text{ and a level 1 loop from } (q_1, s) \text{ to } (q_2, s')\} \ .$$

If S is clear from the context, we omit it.

This rather technical definition of a level 1-loop has two main properties. At first, we obtain again good computational behaviour of this notion in the same spirit as for loops and returns: the possible combinations of initial and final states of 1-loops of a stack s depend on the returns of $\mathrm{pop}_1(s)$, on $\mathrm{CLvl}(s)$, and on $\mathrm{Sym}(s)$. We prove this in the next lemma. Furthermore, this notion is strong enough to capture all parts of a run from a stack s to $\mathrm{pop}_2{}^n(s)$ that are not captured by the notions of loops and returns. This idea is made precise in Lemma E.3.

Lemma E.2. *There is an algorithm that determines for every stack s the set* $1\text{-Loops}(s)$ *from the input* $\mathrm{Sym}(s)$*,* $\mathrm{CLvl}(s)$*, and* $\mathrm{Returns}(\mathrm{top}_2(\mathrm{pop}_1(s)))$*.*

Proof. As in the calculation of returns and loops we are again going to replace the topmost word of $\mathrm{pop}_1(s)$ by a symbol $\top$ and to remove all words below the topmost word of s. Then we will look for configurations where the topmost word of the stack is of the form $\bot\top\mathrm{Sym}(s)$ and the width of the stack is at least 2. All runs from $\bot\top\mathrm{Sym}(s)$ to a configuration of this type are 1-loops of $\bot\top\mathrm{Sym}(s)$. Similar to Lemmas C.7 and C.8, the set $1\text{-Loops}_{\bar{S}}(\bot\top\mathrm{Sym}(s))$ on a extended CPS $\bar{S}$ that simulates for symbol $\top$ the returns of $\mathrm{pop}_1(s)$ equals the set $1\text{-Loops}_S(s)$ with respect to the original CPS S.

For some 2-CPS $S = (Q, \Sigma, q_0, \Delta)$ and fixed $q, q' \in Q$ we construct a new pushdown system $\bar{S}$ that detects whether $(q, q') \in 1\text{-Loops}_S(s)$: we define $\bar{S} = (\bar{Q}, \bar{\Sigma}, \bar{q}_0, \bar{\Delta})$ to be a pushdown system such that $Q \subseteq \bar{Q}$, $\bar{\Sigma} = \Sigma \cup \{\top\}$ for $\top$ a new stack symbol, $\bar{q}_0$ a new initial state and $\bar{\Delta}$ extends Δ by the following transitions:

- transitions that initialise the collapsible pushdown system $\bar{S}$ in the configuration $(q, [(\bot, 1, 0)(\top, 1, 1)(\mathrm{Sym}(s), \mathrm{CLvl}(s), i)])$ for $i = 2$ if $\mathrm{CLvl}(s) = 1$ and $i = 0$ otherwise.
- For $(q_1, q_2) \in \mathrm{Returns}(\mathrm{top}_2(\mathrm{pop}_1(s)))$ the transition $(q_1, \top, q_2, \mathrm{pop}_2)$.
- For two new states q_{pop_1} and q_{pop_2} the transitions $(q', \mathrm{Sym}(s), q_{\mathrm{pop}_1}, \mathrm{pop}_1)$ and $(q_{\mathrm{pop}_1}, \top, q_{\mathrm{pop}_2}, \mathrm{pop}_2)$.

We claim that there is a configuration $c \in \mathrm{CPG}(\bar{S})$ such that

$$\mathrm{CPG}(\bar{S}), c \models \langle \mathrm{pop}_1, q_{\mathrm{pop}_1} \rangle \langle \mathrm{pop}_2, q_{\mathrm{pop}_2} \rangle \mathrm{True}$$

if and only if $(q, q') \in 1\text{-}\mathrm{Loops}_S(s)$. As we can reach the states q_{pop_1} and q_{pop_2} only through the newly introduced transitions, we know that the state of c is q' and the topmost word of the stack has $(\top, 1, 1)(\mathrm{Sym}(s), l, k)$ as postfix. But as there is no transition that pushes some element onto $(\top, 1, 1)$ except for the initialisation of $\bar{S}$ where we push $(\mathrm{Sym}(s), \mathrm{CLvl}(s))$ onto the stack and as we never push $\top$ onto the stack except for the initialisation we know that the topmost word of c is $(\bot, 1, 0)(\top, 1, 1)(\mathrm{Sym}(s), \mathrm{CLvl}(s), i)$. But now observe that any return from some stack s' occurring in a 1-loop of s such that $\mathrm{top}_2(s') = \mathrm{pop}_1(\mathrm{top}_2(s))$ is summarised in one of the new pop_2-transitions we introduced for symbol $\top$. Hence $\bar{S}$ simulates every step that is allowed to perform in the computation of a 1-loops of s. We conclude as $L\mu$ is decidable for $\mathrm{CPG}(\bar{S})$. ∎

Lemma E.3. *Let r be a run from some stack s to some substack $s' \leq s$ such that r does not visit a substack of* $\mathrm{pop}_2(s')$. *Then r decomposes as $r_1 \circ r_2 \circ \ldots \circ r_n \circ l$ where l is a loop of s' and each r_i is of one of the following forms.*

1. *r_i is a return,*
2. *r_i is 1-loop followed by a* collapse *of link level 2,*
3. *r_i is a loop followed by a* collapse *of link level 2,*
4. *r_i is a loop followed by a* pop_1 *(or a* collapse*-operation of link level 1) and for each $i < k$ such that r_k is of form (1) there is some $i < j < k$ such that r_j is of form (2) or (3), or*
5. *r_i is a 1-loop followed by a* pop_1*-operation (or a* collapse *of link level 1) and for each $i < k$ such that r_k is of form (1) there is some $i < j < k$ such that r_j is of form (2) or (3).*

Proof. If $|s| = |s'|$ then $s' = {\mathrm{pop}_1}^n(s)$. As r does not pass a substack of $\mathrm{pop}_2(s')$, it follows immediately that r decomposes into r_i of the fourth form.

Let us now assume that $s' = {\mathrm{pop}_2}^n(s)$ for $n > 0$. The general case then follows by combination of this with the case $|s| = |s'|$. We proceed by induction on the length of r. The first case is when there is an initial segment $r{\restriction}_{[0,m]}$ that is a return. Then we set $r_1 := r{\restriction}_{[0,m]}$ which is of form (1) and the claim follows by induction. Otherwise, assume that there is no such initial segment. Then there is an initial segment r_1 of r such that for all $i < \ln(r_1)$ and $r_1(i) = (q_i, s_i)$ we have that $|s_i| \geq |s|$ and $|s_{\ln(r_1)}| < |s|$. As we are not in the first case, the last operation of r_1 is a collapse such that the topmost word w before the collapse is a prefix of $\mathrm{top}_2(s)$.

Again, we have to distinguish two cases. At first, assume that $w = \mathrm{top}_2(s)$. Note that this assumption implies $\mathrm{CLvl}(s) = 2$. There are the following possibilities:

- Assume that the stack before the collapse is s. Then r_1 is of the form (3). Note that r_1 cannot visit $\mathrm{pop}_1(s)$ because then it would be a return from s by definition as the last part of r_1 would be a return from $\mathrm{pop}_1(s)$. This contradicts the assumption that r_1 is no return from s.

- The stack before the collapse is s' such that $s' = s : (w \setminus s'') : w$ for some nonempty 2-word s''. Then r_1 is a 1-loop plus a collapse-operation: for the same reasons as in the previous case, r_1 never visits $\mathrm{pop}_1(s)$. Before the last operation of r_1 the topmost word is equal to w. Hence, whenever we pass some stack $\hat{s}$ at $i < \ln(r)$ such that $\mathrm{top}_2(\hat{s}) = \mathrm{pop}_1(w)$ there is a $i > j$ such that the restriction of r_1 to $[i, j]$ is a return. This is due to the fact that we cannot restore $\mathrm{top}_2(s)$ as topmost word from $\hat{s}$ by $\mathrm{push}_{\sigma,2}$-operations because $\hat{s} \neq \mathrm{pop}_1(s)$ whence the collapse-links would not agree.

As last case, assume that $w < \mathrm{top}_2(s)$. Then there is a minimal initial segment r_1 of r such that r_1 ends with stack $\mathrm{pop}_1(s)$ or $s : (w \setminus s') : \mathrm{pop}_1(\mathrm{top}_2(s))$ such that r does not continue with a return. In the former case r_1 is of form (4), in the latter it is of form (5). In order to see this, we have to check the side condition on the segments following r_1. By induction hypothesis r without the initial segment r_1 decomposes as a sequence of subruns of the forms mentioned above. Note that this decomposition cannot start with a return by the assumption on r_1. But then there has to be a segment of form (3) or (2) before the first occurrence of a segment which is a return: either the first segment is of one of these forms or it is of a form that requires such a segment before the first segment that is a return. But this shows that r_1 is of the form (4) or (5). ∎

The previous lemma tells us that any run to a substack decomposes into three different classes of subruns: returns, subruns that decrease the topmost word by one letter but possibly increase the width of the stack, and subruns that end up with a collapse-operation performed on a stack with equal topmost word as their initial stack. Note that in the case of decreasing the topmost word we eventually perform a collapse of level 2. Hence, it does not really matter that we increase the width. Eventually we perform a collapse-operation on a prefix of the initial topmost word. This collapse then deletes all the new words we created on the stack. If the first segment of the run is a 1-loop plus a pop_1-operation then the resulting stack is the same whether we let $r_2 \circ r_3 \circ \ldots \circ r_n$ start at the last stack of r_1 or at the stack $\mathrm{pop}_1(s)$ for s the initial stack of r_1.

The decomposition of a run according to Lemma E.3 is the starting point for deciding whether a run from some stack s to $\mathrm{pop}_2{}^n(s)$ exists. The basic idea is to guess the form and the initial and final state of each segment the run consists of. As in the case of the existence of a configuration in a given graph we want to attach these data to certain elements in the encoding of the stack s. Unfortunately, this time we do not find the stacks that our run visits encoded as subtrees. Especially the case where the run has a segment which is a 1-loop followed by some pop_1-operation may lead to a stack that is much larger than the one we were starting from. But it turns out that we are only interested in the topmost word and a small initial part of such a stack. And in every case, we will find a corresponding milestone of s with the same properties. We will use an element $t \in T$ in the encoding T of some configuration c_1 as a representative for the stack $\mathrm{LStck}(t)$ or some stack s' such that $\mathrm{LStck}(t) \leq s'$ and $\mathrm{top}_2(s') = \mathrm{top}_2(\mathrm{LStck}(t))$. Assume that there is some run r from $c_1 = (q_1, s_1)$ to some substack $\mathrm{pop}_2{}^n(s_1)$. Then we will label the rightmost leaf of $\mathrm{Enc}(c_1)$ with a marker of the form of the first segment of this run. According to the form of this segment we MSO-definably find a second element in the tree, which corresponds to the stack reached by the first segment. Then we will continue to label this element with the form and the initial and final state of the next segment of the run. We continue with this idea until we reach the last stack of the run. As we label every element of the tree with a finite amount of data and as the next representative of a stack is MSO-definable, this procedure may be done by a tree-automaton.

The labelling with forms of segments and initial and final states is done by the use of functions which we call reachability guesses.

Definition E.4. Let $c_1 = (q_1, s_1), c_2 = (q_2, s_2)$ be configurations such that $s_2 = \mathrm{pop}_2{}^n(s_1)$ for some $n \in \mathbb{N}$. We call a function

$$\lambda : \mathrm{dom}(\mathrm{Enc}(c_1)) \setminus \mathrm{dom}(\mathrm{Enc}(c_2)) \to \{1, 2, 3, 4, 5\} \times Q \times Q$$

a reachability guess for c_1 and c_2.

Remark E.5. Note that using MSO we can existentially quantify over reachability guesses for c_1 and c_2 in the structure $\mathrm{Enc}(c_1) \otimes \mathrm{Enc}(c_2)$.

As explained before we are going to use a reachability guess in order to check the existence of a run from some stack to a substack. The checking is done locally between representatives of some stacks visited during the run. Here we mean by local that every representative has exactly one successor which represents the next stack that we visit. But where we find this next representative depends on the form of the segment of the run that starts at the current representative. In the next definition we formally define how we can find the next representative. But before, we give an informal description. Let r be a run from some stack s to $\mathrm{pop}_2{}^m(s)$ and assume that there is some $t \in \mathrm{Enc}(s)$ such that r visits some stack s' that contains $\mathrm{LStck}(t)$ as substack and whose topmost word is equal to the topmost word of $\mathrm{LStck}(t)$. The reachability guess corresponding to r labels t with one of the forms according to Lemma E.3. When we assume that the segment of r starting at s' is a return or ends in a collapse of level 2, the representative for the next segment of the run is the rightmost leaf t' of the encoding of the stack obtained by performing a pop_2 or a collapse of level 2, respectively, on $\mathrm{LStck}(t, \mathrm{Enc}(s))$. Note that $\mathrm{LStck}(t', \mathrm{Enc}(s))$ is the stack that r reaches at the end of the segment corresponding to t. This is due to the fact that a loop or 1-loop followed by a collapse reaches exactly the same final stack as just performing the collapse directly. In all other cases the segment starting at s' ends with a pop_1- or a collapse-operation of link level 1. Then there is a minimal t' such that $t = t'1^k01^l$ for some $l, k \in \mathbb{N}$. This t' will be the representative of the next segment of the run. Note that $\mathrm{top}_2(\mathrm{LStck}(t))$ and $\mathrm{top}_2(\mathrm{LStck}(t'))$ differ in one pop_1-operation and $\mathrm{LStck}(t')$ is a substack of the stack s'' we reach by the segment of r starting at s'.

If we ensure that every occurrence of the last case has a following occurrence of the second case, we assure that we always reach eventually a representative t'' such that the run r reaches $\mathrm{LStck}(t'', \mathrm{Enc}(s))$.

Definition E.6. Let $t \in \mathrm{dom}(\mathrm{Enc}(c_1)) \setminus \mathrm{dom}(\mathrm{Enc}(c_2))$ and λ a reachability guess for c_1 and c_2. Then the successor of t with respect to λ is defined depending on the first component of $\lambda(t)$, denoted as $\pi_1(\lambda(t))$. There are the following cases.

(1) $\pi_1(\lambda(t)) = 1$: Let t' be the ancestor of t such that $t = t'10^m$ for some $m \in \mathbb{N}$. Then there is a lexicographically maximal word $t'' \in \{\varepsilon\} \cup \{0\{0,1\}^*\}$ such that $t't'' \in \mathrm{dom}(\mathrm{Enc}(c_1))$ ($t't''$ is the maximal element in the subtree rooted at $t'0$ if $t'0$ is in the tree, otherwise it is just t'). We say $t't''$ is the successor of t with respect to λ.

(2) $\pi_1(\lambda(t)) \in \{2,3\}$: The successor of t with respect to λ is the rightmost leaf of the encoding of collapse(LStck(t)), i.e., the rightmost leaf of the subtree rooted at $t'0$ for t' the element such that $t = t'10^m1^n$ where $m > 0$ and $n \in \mathbb{N}$. In case that $t'0$ is not in the domain of the tree, the successor of t is just t'.
(3) $\pi_1(\lambda(t)) \in \{4,5\}$: Let t' be the unique element such that $t = t'01^n$ for some $n \in \mathbb{N}$. Then t' is the successor of t with respect to λ.

Remark E.7. The motivation of the previous definition are the following observations.
(1) $\pi_1(\lambda(t)) = 1$: The successor t' of t is chosen such that $\mathrm{LStck}(t') = \mathrm{pop}_2(\mathrm{LStck}(t))$.
(2) $\pi_1(\lambda(t)) \in \{2,3\}$: The successor t' is chosen such that $\mathrm{LStck}(t') = \mathrm{collapse}(\mathrm{LStck}(t))$ (under the assumption that $\mathrm{CLvl}(\mathrm{LStck}(t)) = 2$).
(3) $\pi_1(\lambda(t)) \in \{4,5\}$: The successor t' of t is chosen such that $\mathrm{LStck}(t')$ is the maximal milestone of $\mathrm{LStck}(t)$ such that $\mathrm{top}_2(\mathrm{LStck}(t')) = \mathrm{top}_2(\mathrm{pop}_1(\mathrm{LStck}(t)))$.

Note that the successor with respect to λ is MSO-definable. With this notion of successor we are able to define what a valid reachability guess is. It is basically a guess that represents a decomposition of a run according to Lemma E.3. Note that we only consider the sequence of elements induced by the rightmost leaf and the successors with respect to λ. All other assignments of the reachability guess do not matter for its validity.

Before we state the definition, we have to fix some notation.

Remark E.8. From Lemma E.2 and the analogous lemmas in Section C.1, one easily derives the MSO-definability of the following functions for each pushdown system S of level 2 and arbitrary $c \in \mathrm{CPG}(S)$

$$\begin{aligned}
\mathrm{f}_1^{\circlearrowleft} : \mathrm{Enc}(c) \setminus \{\varepsilon\} &\to 2^{Q\times Q} \\
t &\mapsto \text{1-Loops}(\mathrm{LStck}(t)) \ , \\
\mathrm{f}^{\circlearrowleft} : \mathrm{Enc}(c) \setminus \{\varepsilon\} &\to 2^{Q\times Q} \\
t &\mapsto \mathrm{Loops}(\mathrm{LStck}(t)) \ , \\
\mathrm{f}^{\mathrm{Rt}} : \mathrm{Enc}(c) \setminus \{\varepsilon\} &\to 2^{Q\times Q} \\
t &\mapsto \mathrm{Returns}(\mathrm{LStck}(t)) \ .
\end{aligned}$$

Definition E.9. Let λ be a reachability guess for the configurations $\bar{c} = (\bar{q}, \bar{s})$ and $\hat{c} = (\hat{q}, \hat{s})$ where $\hat{s} = {\mathrm{pop}_2}^n(\bar{s})$ for some $n \in \mathbb{N}$. We call λ *valid* if the following hold: there is a finite sequence $t_1, t_2, \dots, t_n$ such that t_1 is the rightmost leaf of $\mathrm{Enc}(\bar{c} \otimes \hat{c})$, t_{i+1} is the successor of t_i with respect to λ for every $i < n$, and the successor of t_n with respect to λ is the rightmost leaf l of $\mathrm{Enc}(\hat{c})$. For each $1 \leq i \leq n$ we write $\lambda(t_i) = (k_i, q_i, q'_i)$. Then we require that $q_1 = \bar{q}$, $(q'_n, \hat{q}) \in \mathrm{f}^{\circlearrowleft}(l)$, $q'_i = q_{i+1}$, and
(1) if $k_i = 1$ then $(q_i, q'_i) \in \mathrm{f}^{\mathrm{Rt}}(t_i)$,
(2) if $k_i = 2$ then $\mathrm{CLvl}(\mathrm{LStck}(t_i)) = 2$ and there is a $q \in Q$ such that

$$(q_i, q) \in \mathrm{f}_1^{\circlearrowleft}(t_i), \text{ and } (q, \mathrm{Sym}(\mathrm{LStck}(t_i)), q'_i, \mathrm{collapse}) \qquad \in \Delta \ ,$$

(3) if $k_i = 3$ then $\mathrm{CLvl}(\mathrm{LStck}(t_i)) = 2$ and there is some $q \in Q$ such that

$$(q_i, q) \in \mathrm{f}^{\circlearrowleft}(t_i) \text{ and } (q, \mathrm{Sym}(\mathrm{LStck}(t_i)), q'_i, \mathrm{collapse}) \qquad \in \Delta \ ,$$

(4) if $k_i = 4$ then there is some $q \in Q$ such that $(q_i, q) \in \mathrm{f}^{\circlearrowleft}(t_i)$ and one of the following hold: either

$$(q, \mathrm{Sym}(\mathrm{LStck}(t_i)), q'_i, \mathrm{pop}_1) \in \Delta \text{ or}$$
$$\mathrm{CLvl}(\mathrm{LStck}(t_i)) = 1 \text{ and } (q, \mathrm{Sym}(\mathrm{LStck}(t_i)), q'_i, \mathrm{collapse}) \in \Delta\ .$$

(5) if $k_i = 5$ then there is a $q \in Q$ such that $(q_i, q) \in \mathrm{f}_1^{\circlearrowleft}(t_i)$ and the following hold: either

$$(q, \mathrm{Sym}(\mathrm{LStck}(t_i)), q'_i, \mathrm{pop}_1) \in \Delta \text{ or}$$
$$\mathrm{CLvl}(\mathrm{LStck}(t_i)) = 1 \text{ and } (q, \mathrm{Sym}(\mathrm{LStck}(t_i)), q'_i, \mathrm{collapse}) \in \Delta\ .$$

Finally, we require that for all i with $k_i \in \{4, 5\}$ there is some $j > i$ such that $k_j \in \{2, 3\}$ and $k_l \neq 1$ for all $i < l < j$.

Example E.10. Consider the stack s in figure 15 and the following transitions:

(1) $(q_1, a, q_2, \mathrm{clone}_2)$
(2) $(q_2, a, q_3, \mathrm{collapse})$
(3) $(q_3, a, q_2, \mathrm{clone}_2)$
(4) $(q_3, b, q_2, \mathrm{push}_{b,2})$
(5) $(q_1, c, q_4, \mathrm{push}_{b,1})$
(6) $(q_2, b, q_1, \mathrm{collapse})$
(7) $(q_2, c, q_1, \mathrm{collapse})$
(8) $(q_4, b, q_2, \mathrm{pop}_1)$

As these transitions form a deterministic relation, there is a unique run starting in (q_1, s). This run r is generated by using the transitions in the following order: (1), (2), (3), (2), (4), (6), (5), (8), (7), (5), (8), (7) and r ends in the configuration $(q_1, \perp_2)$. According to the decomposition of Lemma E.3, $r{\restriction}_{[0,2]}$ is of form (5), $r{\restriction}_{[2,4]}$ is of form (2), $r{\restriction}_{[4,6]}$ is of form (1), $r{\restriction}_{[6,9]}$ is of form (4), and $r{\restriction}_{[9,12]}$ is of form (3). A valid reachability guess corresponding to this decomposition for $\mathrm{Enc}(q_1, s) \otimes \mathrm{Enc}(q_1, \perp_2)$ is depicted in 15 (the bold labels are the values of the reachability guess; we omitted the values on irrelevant nodes of the tree).

Remark E.11. There is a formula in MSO that is valid on $\mathrm{Enc}(c_1) \otimes \mathrm{Enc}(c_2)$ if and only if there is a valid reachability guess for c_1 and c_2.

The next lemma shows that a valid reachability guess represents indeed a run from a stack to a substack and that we can extract a valid reachability guess from every such run. This shows that we can detect the reachability of substacks of the form ${\mathrm{pop}_2}^n(s)$ for $n \in \mathbb{N}$ and s some stack.

Lemma E.12. *For configurations $c_1 = (q_1, s_1), c_2 = (q_2, s_2) \in \mathrm{CPG}(S)$ such that $s_2 = {\mathrm{pop}_2}^n(s_1)$ for some $n \in \mathbb{N}$ there is a run from c_1 to c_2 if and only if there is a valid reachability guess for c_1 and c_2.*

Proof. The proof from left to right is by decomposition of a a run r from c_1 to c_2 in subruns according to Lemma E.3. We label inductively the elements t_i. t_1 is the rightmost leaf of $\mathrm{Enc}(c_1) \otimes \mathrm{Enc}(c_2)$ and we label it with $\lambda(t_1) := (k_1, q_1, q'_1)$ if there is an initial segment r_1 of r starting in state q_1, ending in state q'_1 and r_1 is of form k_1 according to the enumeration of the Lemma E.3. We call the successor of t_1 with respect to λ t_2 and note that the last stack visited by r_1 contains $\mathrm{LStck}(t_2)$ as substack and its topmost word agrees with

$$s := \begin{matrix} & & & (a,1,3) & (a,1,3) \\ & (c,1,2) & (b,1,2) & (a,2,3) & (a,2,3) \\ & (c,2,1) & (c,2,1) & (c,2,1) & (c,2,1) \\ (\bot,1,0) & (\bot,1,0) & (\bot,1,0) & (\bot,1,0) & (\bot,1,0) \end{matrix}$$

$$\begin{tikzcd}
 & & & (a,1),\Box, \arrow[r] & \begin{matrix}\varepsilon,\Box \\ (\mathbf{5},\mathbf{q_1},\mathbf{q_3})\end{matrix} \\
 & \begin{matrix}(c,1),\Box \\ (\mathbf{4},\mathbf{q_1},\mathbf{q_1})\end{matrix} & \begin{matrix}(b,1),\Box \\ (\mathbf{1},\mathbf{q_3},\mathbf{q_1})\end{matrix} & \begin{matrix}(a,2),\Box \\ (\mathbf{2},\mathbf{q_3},\mathbf{q_3})\end{matrix} \arrow[u] & \\
 & \begin{matrix}(c,2),\Box \\ (\mathbf{3},\mathbf{q_1},\mathbf{q_1})\end{matrix} \arrow[u] \arrow[r] & \varepsilon,\Box \arrow[u] \arrow[r] & \varepsilon,\Box \arrow[u] & \\
 (\bot,1),(\bot,1) \arrow[r] & \varepsilon,\Box \arrow[u] & & & \\
 (q_1,q_1) \arrow[u] & & & &
\end{tikzcd}$$

Figure 15: Example of a valid reachability guess

$\mathrm{top}_2(\mathrm{LStck}(t_2))$. If the last operation of r_1 was a pop_2 or a collapse of link level 2 then the last stack of r_1 agrees with $\mathrm{LStck}(t_2)$. Now we may proceed inductively labelling t_i for $1 \leq i$ with the data obtained from r_i until the successor t_{n+1} of some t_n with respect to λ is the rightmost leaf of $\mathrm{Enc}(c_2)$. This happens for the last segment r_{n-1} in the decomposition of r. r continues with a sequence of high loops l, whence the last state of r_{n-1} which is the third component of $\lambda(t_n)$ together with the last state of r form a pair in $\mathrm{f}^{\circlearrowleft}(t_{n+1})$ which contains by definition all initial and final states of sequences of high loops of the stack of c_2.

For the proof from right to left assume that λ is a valid reachability guess for c_1 and c_2. We prove the following

Claim. Let t_1 and t_2 be the first two elements of the sequence witnessing the validity of λ.

If $\lambda(t_1) = (1, q_1, q_1')$ then there is a run from $(q_1, \mathrm{LStck}(t_1))$ to $(q_1', \mathrm{LStck}(t_2))$

If $\lambda(t_1) = (j, q_1, q_1')$ for $j \in \{2, 3\}$ and (q_1, s') is some configuration such that $\mathrm{LStck}(t_1)$ is a substack of s' and the topmost words of these two stacks agree, then there is a run from (q_1, s') to $(q_1', \mathrm{LStck}(t_2))$.

If $\lambda(t_1) = (j, q_1, q_1')$ for $j \in \{4, 5\}$ and (q_1, s') is some configuration such that $\mathrm{LStck}(t_1)$ is a substack of s' and the topmost words of these two stacks agree, then there is some stack s'' such that $\mathrm{LStck}(t_2)$ is a substack of s'' and the topmost words of these two stacks agree and there is some run from (q_1, s') to (q_1', s'')

With the claim we prove the lemma just by induction: note that for any valid λ such that $\lambda(t_1) = (j, q_1, q_1')$ for $j \in \{1, 2, 3, 4, 5\}$ it holds that $\lambda{\restriction}_{\mathrm{dom}(\mathrm{Enc}(q_1', \mathrm{LStck}(t_2)))}$ is a valid labelling with respect to $(q_1', \mathrm{LStck}(t_2))$ and c_2. Note also that we ruled out the case that $\lambda(t_1) =$

$(4, q_1, q_1')$ or $\lambda(t_1) = (5, q_1, q_1')$ and $\lambda(t_2) = (1, q_2, q_2')$ for arbitrary states $q_1, q_2, q_1', q_2' \in Q$. Hence, in the inductive proof we only reach an element t_j with $\lambda(t_i) = (1, q_j, q_j')$ if the run reaches $(q_j, \mathrm{LStck}(t_j))$

The proof of the claim is just by noting that $\mathrm{Loops}(\mathrm{LStck}(t_1))$ and $\mathrm{Loops}(s')$ as well as 1-$\mathrm{Loops}(\mathrm{LStck}(t_1))$ and 1-$\mathrm{Loops}(s')$) agree due to lemmas C.8 and E.2, respectively. ∎

Now, we have to deal with the pop_1-case: how can we decide that a substack $s' = {\mathrm{pop}_1}^m(s)$ is reachable from a configuration (q, s)? Here we only have to focus on runs that do not pass $\mathrm{pop}_2(s)$. Hence, such a run is a composition loops and pop_1-operations.

Lemma E.13. *Let $A \subseteq \mathrm{CPG}(S) \times \mathrm{CPG}(S)$ be the set of configurations such that $(c_1, c_2) \in A$ if*

$$c_1 = (q_1, s_1), c_2 = (q_2, s_2), s_2 = {\mathrm{pop}_1}^n(s_1),$$
$$\textit{and there is a run from } c_1 \textit{ to } c_2 \textit{ that never passes } \mathrm{pop}_2(s_1) \ .$$

Then the set

$$\{\mathrm{Enc}(c_1) \otimes \mathrm{Enc}(c_2) : (c_1, c_2) \in A\}$$

is regular.

Proof. Reading in $\mathrm{Enc}(c_1) \otimes \mathrm{Enc}(c_2)$ the path from the rightmost leaf of $\mathrm{Enc}(c_2)$ to the rightmost leaf of $\mathrm{Enc}(c_1)$ as a word, we obtain the postfix of the topmost word of s_1 which is removed from s_1 by r. We may label every letter with the initial and the finial state of a subrun of r that consists of a loops extended by one pop_1-operation. An automaton may easily check that these subruns are locally consistent. On the other hand, any locally consistent such labelling gives rise to a corresponding run from c_1 to c_2. ∎

Combining the previous lemmas, we obtain that the existence of a path from a configuration to another configuration whose stack is a substack of the former one can be checked by an automaton. Now we want to prove the other direction: the set of pairs of configurations such that the first configuration contains a substack of the second configuration and the second one is reachable by a run form the first one is also regular.

Again, this proof decomposes into two parts: at first we have to deal with the combination of high loops with $\mathrm{push}_{\sigma,l}$-operations in order to come from some stack ${\mathrm{pop}_1}^n(s)$ to the stack s. Afterwards, we have to deal with runs from some stack ${\mathrm{pop}_2}^m(s)$ to s.

The first part is a kind of inverse of the previous lemma and is proved using the same technique: the path leading from the rightmost leaf of the encoding of the smaller stack to the rightmost leaf of the encoding of the larger stack is MSO-definable in the convolution of the two encodings. Then we label every element along the path with the initial and final state of a sequence of high loops followed by the appropriate $\mathrm{push}_{\sigma,l}$-operation and check that the labelling is correct. This gives us the following result:

Lemma E.14. *Let $A \subseteq \mathrm{CPG}(S) \times \mathrm{CPG}(S)$ be the set of configurations of a pushdown system S such that (c_1, c_2) for $c_1 = (q_1, s_1), c_2 = (q_2, s_2)$ is in A if and only if the following hold:*

- *s_2 can be generated from s_1 by application of a sequence of* push*-operations*
- *There is a run from c_1 to c_2 not passing any proper substack of s_1.*

Then $\mathrm{Enc}(A) := \{\mathrm{Enc}(c_1) \otimes \mathrm{Enc}(c_2) : (c_1, c_2) \in A)$ *is regular.*

Now we have to deal with the second part. This is the case where the second configuration consists of more words than the first one, i.e., the stack s' of the first configuration is of the form $s' = \mathrm{pop}_2{}^n(s)$ where s is the stack of the second configuration. This implies that s' is a milestone of s. Hence the encoding of s' is a subtree of the encoding of s and the rightmost leaf in the encoding of s' does not have a left successor in the encoding of s.

Lemma E.15. *There is an* MSO*-formula defining all convolutions of encodings of configurations* $c_1 = (q_1, s_1)$ *and* $c_2 = (q_2, s_2)$ *such that* $s_1 = \mathrm{pop}_2{}^n(s_2)$, $n \in N \setminus \{0\}$ *and there is a run that connects* c_1 *with* c_2 *but does not pass a proper substack of* s_1*.*

Proof. We are basically going to use the same ideas as for the proof of the reachability of some configuration from the initial configuration. Note that s_1 is by definition a milestone of s_2. An automaton reading $\mathrm{Enc}(c_1) \otimes \mathrm{Enc}(c_2)$ can determine where the rightmost leaf of the encoding of c_1 is. Now the automaton can perform the algorithm that guesses for each lexicographic greater element in the domain of $\mathrm{Enc}(c_2)$ the last state of a run from c_1 to c_2. Checking the correctness of the guesses may be done as in the case of the reachability of c_2 from the initial configuration. Finally, the automaton has to check that the guess of a state for s_1 is valid, i.e., for s_1 labelled by some state q, it checks whether $(q_1, q) \in \mathrm{Loops}(s_1)$. ∎

Now we can summarise the previous lemmas about reachability in the following

Corollary E.16. *For each level* 2 CPS S *the structure* $(\mathrm{CPG}(S), \mathrm{REACH})$ *is tree automatic and the* $\mathrm{FO} + \mathrm{REACH}$*-theory of* $\mathrm{CPG}(S)$ *is decidable.*

Proof. Let r be a run from a configuration $c_1 = (q_1, s_1)$ to a configuration $c_5 = (q_5, s_5)$. Then this run decomposes as $r = r_1 \circ r_2 \circ r_3 \circ r_4$ where r_1 is a run from c_1 to some (q_2, s_2) with $s_2 = \mathrm{pop}_2{}^n(s_1)$ and $n \in \mathbb{N}$ maximal with this property for configurations in r. Then r_2 is a run from (q_2, s_2) to (q_3, s_3) for $s_3 = \mathrm{pop}_1{}^m(s_2)$ and $m \in \mathbb{N}$ maximal with this property for configurations in r. r_3 is a run from (q_3, s_3) to some (q_4, s_4) such that $\mathrm{pop}_1{}^l(s_4) = s_3$ and l is maximal with this property in the run r. and finally r_4 is a run from (q_4, s_4) to c_2. For given configurations $(q_2, s_2), (q_3, s_3)$ and (q_4, s_4) we already proved that the existence of r_i $1 \leq i \leq 4$ is automata recognisable. But then existential quantification over these three configurations leads to the desired result. ∎

Furthermore, note that we can also define a restricted reachability predicate $\mathrm{REACH}_{\Delta'}$ for $\Delta' \subseteq \Delta$ which expresses that there is a path between two configurations consisting only of transitions in Δ'. The proof of decidability of $\mathrm{FO} + \mathrm{REACH}_{\Delta'}$ is the same but we have to calculate loops, returns, etc. with respect to the set Δ'. The reachability via Δ' can also be seen as edges between configurations in the ε-contraction of the given pushdown graph where exactly those edges from Δ' are labelled by ε.

Corollary E.17. *If* G *is a graph obtained by* ε*-contraction from a level* 2 CPG G' *then* G *is tree-automatic and the* $\mathrm{FO} + \mathrm{REACH}$*-theory of this graph is decidable.*

Remark E.18. Note that the class of graphs obtained by application of ε-contractions to collapsible pushdown graphs of order 2 subsumes the graphs in the second level of the Caucal-hierarchy.

Appendix F. Reachability under regular expressions

In the previous section, we proved that the reachability predicate on collapsible pushdown graphs is decidable. Variating the pushdown system that we consider we can even prove that reachability by a path that satisfies a regular expression is decidable. Consider some collapsible pushdown system S with transition relation Δ. Let r be a regular expression over the alphabet Δ. We write REACH_r for the binary predicate that is true for those configurations $c_1, c_2 \in \mathrm{CPG}(S)$ where c_2 is reachable from c_1 using a path whose transitions form a word in the language of r. We show that REACH_r is decidable by constructing some kind of product of S with the finite automaton A_r corresponding to r. We show that $\mathrm{CPG}(S)$ is first-order interpretable in this product and we show that the reachability predicate REACH can be used to decide REACH_r by starting in appropriate configurations of the product we constructed. Before we state the lemma, we introduce some abreviations. For x a variable and q a state of some $\mathrm{CPS}(S)$ we write $x \in q$ for the FO formula stating that x is a configuration with state q. This is definable as all incoming edges are labelled by the state of x and all configurations but the initial one have an incoming edge. Furthermore, the state of the initial configuration is the initial state. As the set of states is finite, we also use the abbreviation $x \in Q$ for a set of states Q and mean that $x \in q$ for some $q \in Q$.

Lemma F.1. *Let $S = (Q, \Sigma, q_i, \Delta)$ be a 2-CPS and $r_1, r_2, \dots, r_n$ regular expressions. Then $(\mathrm{CPG}(S), \mathrm{REACH}, (\mathrm{REACH}_{r_i})_{1\leq i\leq n})$ is tree-automatic.*

Proof. Without loss of generality, we assume that $n = 1$ and write r for r_1. The general case follows by iterating our construction. Let $A = (F, \Delta, f_i, f_f, \Delta_r)$ be the finite automaton corresponding to r where F are the states of A, f_i the initial configuration, f_f the final configuration and Δ_r the transition relation.

We define a new 2-CPS $\bar{S} = (\bar{Q}, \Sigma, q_i, \bar{\Delta})$ as follows

- $\bar{Q} := Q \cup (Q \times F)$ and
- $\bar{\Delta}$ is the union

$$\begin{aligned}
&\Delta \cup \{(q, \sigma, (q, f_i), \mathrm{id}) : \sigma \in \Sigma, q \in Q\}\\
&\cup \{(q, \sigma, (q, f_f), \mathrm{id}) : \sigma \in \Sigma, q \in Q\}\\
&\cup \{((q, f), \sigma, (q', f'), \mathrm{op}) : (q, \sigma, q', \mathrm{op}) \in \Delta \text{ and } (f, (q, \sigma, q', \mathrm{op}), f') \in \Delta_r\} \ .
\end{aligned}$$

Note that $\mathrm{CPG}(S)$ is FO definable in $\mathrm{CPG}(\bar{S})$: both graphs have the same initial configuration and $\bar{S}$ extends S only by transitions that lead to configurations with states in $Q \times F$. Hence, the restriction of $\mathrm{CPG}(\bar{S})$ to the set $\{x \in \mathrm{CPG}(\bar{S}) : x \in Q\}$ is isomorphic to $\mathrm{CPG}(S)$.

On the other hand, the construction is such that there is a path from $((q, f_i), s)$ to $((q', f_f), s')$ in $\mathrm{CPG}(\bar{S})$ if and only if there is a path from (q, s) to (q', s') in $\mathrm{CPG}(\bar{S})$ whose path corresponds to an accepting word of A. Hence, deciding $(x, y) \in \mathrm{REACH}_r$ on $\mathrm{CPG}(S)$ corresponds to deciding

$$\exists x' \exists y' \big(x \vdash^{(q,f_i),\mathrm{id}} x' \wedge y \vdash^{(q,f_f),\mathrm{id}} y' \wedge (x', y') \in \mathrm{REACH}\big)$$

on $\mathrm{CPG}(S')$. ∎