Collapsible Pushdown Graphs of Level 2 are Tree-Automatic

Alexander Kartzow

Symposium on Theoretical Aspects of Computer Science 2010 (Nancy, France), pp. 501-512
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, Germany

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 (with regular constraints) for pairs of configurations, the structures remain tree-automatic. Hence, their FO theories are decidable, even when expanded by a reachability predicate. 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 have exactly the same power as higher-order recursion schemes as methods 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[1] of level l 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 preceding level by applying graph unfoldings and MSO interpretations. Both operations preserve the decidability of the MSO theory whence the Caucal-hierarchy forms a rather large 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 modal μ-calculus theories of all graphs in the hierarchy. 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 μ-calculus model checking purposes. We

1998 ACM Subject Classification: F.4.1[Theory of Computation]:Mathematical Logic.

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

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

SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

proved in [7] that nested pushdown trees also have decidable first-order theories. We 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 closure properties 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 are still tree-automatic when 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. Thus, first-order logic extended by reachability predicates is decidable on level 2 collapsible pushdown graphs.

In the next section, we introduce the necessary notions concerning tree-automaticity and in Section 3 we define collapsible pushdown graphs. We explain the translation of configurations into trees in Section 4. Section 5 is a sketch of the proof that this translation yields tree-automatic representations of collapsible pushdown graphs, even when enriched with certain regular reachability predicates. The last section contains some concluding remarks about questions arising from our result.

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 $T : D \to \Sigma$ for a finite $D \subseteq \{0,1\}^*$ which is closed under prefixes.
For $d \in D$ we denote by T_d the *subtree rooted at* d.

Sometimes it is useful to define trees inductively by describing their left and right subtrees. 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

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

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 T and T' is given by a function

$$T \otimes T' : \operatorname{dom}(T) \cup \operatorname{dom}(T') \to (\Sigma \cup \{\Box\})^2$$

where $\Box$ is a new symbol for padding and

$$(T \otimes T')(d) := \begin{cases} (T(d), T'(d)) & \text{if } d \in \operatorname{dom}(T) \cap \operatorname{dom}(T') \\ (T(d), \Box) & \text{if } d \in \operatorname{dom}(T) \setminus \operatorname{dom}(T') \\ (\Box, T'(d)) & \text{if } d \in \operatorname{dom}(T') \setminus \operatorname{dom}(T) \end{cases}$$

By "tree-automata" we mean a nondeterministic finite automaton that labels a finite tree top-down.

Definition 2.1. A structure $\mathfrak{B} = (B, E_1, E_2, \ldots, E_n)$ with domain B and binary relations E_i is *tree-automatic* if there are tree-automata $A_B, A_{E_1}, A_{E_2}, \ldots, A_{E_n}$ and a bijection

$f : L \to B$ for L the language accepted by A_B such that the following hold. For $T, T' \in L$, the automaton A_{E_i} accepts $T \otimes T'$ if and only if $\big(f(T), f(T')\big) \in E_i$.

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.*

We will use the classical result that regular sets of trees are MSO definable.

Theorem 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 comprehensive 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^{*2} for $(\Sigma^*)^*$ and Σ^{+2} for $(\Sigma^+)^+$. We call an $s \in \Sigma^{*2}$ a 2-word.

Let us fix a 2-word $s \in \Sigma^{*2}$ which consists of an ordered list $w_1, w_2, \ldots, w_m \in \Sigma^*$. We separate the words of this list by colons writing $s = w_1 : w_2 : \ldots : w_m$. By $|s|$ we denote the number of words s consists of, i.e., $|s| = m$.

For another word $s' = w'_1 : w'_2 : \ldots : w'_n \in \Sigma^{*2}$, we write $s : s'$ for the concatenation $w_1 : w_2 : \ldots : w_m : w'_1 : w'_2 : \ldots : w'_n$.

If $w \in \Sigma^*$, we write $[w]$ for the 2-word that consists of a list of one word which is w.

A level 2 collapsible pushdown stack is a special element of $(\Sigma \times \{1, 2\} \times \mathbb{N})^{+2}$ 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 pointer*: every element in a collapsible pushdown stack has a pointer to some substack and applying the collapse operation returns the substack to which the topmost symbol of the stack points. Here, the first number denotes the *collapse level*. If it is 1 the collapse pointer always points to the symbol below the topmost symbol and the collapse operations just removes the topmost symbol. The more interesting case is when the collapse level of the topmost symbol of the stack s is 2. Then the stack obtained by the collapse contains the first n words of s where n is the second number in the topmost element of s.

The initial level 1 stack is $\bot_1 := (\bot, 1, 0)$ and the initial level 2 stack is $\bot_2 := [\bot_1]$.

For $k \in \{1, 2\}$ and for a 2-word $s = w_1 : w_2 : \ldots : w_n \in (\Sigma \times \{1, 2\} \times \mathbb{N})^{+2}$ such that $w_n = a_1 a_2 \ldots a_m$ with $a_i \in \Sigma \times \{1, 2\} \times \mathbb{N}$ for all $1 \leq i \leq m$:

- we define the *topmost $(k-1)$-word of s* as $\mathrm{top}_k(s) := \begin{cases} w_n & \text{if } k = 2 \\ a_m & \text{if } k = 1 \end{cases}$
- for $\mathrm{top}_1(s) = (\sigma, i, j) \in \Sigma \times \{1, 2\} \times \mathbb{N}$, 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* $\mathrm{CLnk}(s) := j$.

For s, w_n and k as before, $\sigma \in \Sigma\backslash\{\bot\}$, and $w'_n := a_1 \dots a_{m-1}$, we define the stack operations

$$\mathrm{pop}_k(s) := \begin{cases} w_1 : w_2 : \ldots : w_{n-1} & \text{if } k = 2, n \geq 2 \\ w_1 : w_2 : \ldots : w_{n-1} : w'_n & \text{if } k = 1, m \geq 2 \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\mathrm{clone}_2(s) := w_1 : w_2 : \ldots : w_{n-1} : w_n : w_n$$

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

$$\mathrm{collapse}(s) := \begin{cases} w_1 : w_2 : \ldots : w_r & \text{if } \mathrm{CLvl}(s) = 2, \mathrm{CLnk}(s) = r > 0 \\ \mathrm{pop}_1(s) & \text{if } \mathrm{CLvl}(s) = 1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

The *set of level* 2*-operations* is $\mathrm{OP} := \{\mathrm{push}_{\sigma,1}, \mathrm{push}_{\sigma,2}, \mathrm{clone}_2, \mathrm{pop}_1, \mathrm{pop}_2, \mathrm{collapse}\}$. The *set of level* 2 *stacks*, $\mathrm{Stck}(\Sigma)$, is the smallest set that contains $\bot_2$ and is closed under all operations from OP.

Note that collapse- and pop_k-operations are only allowed if the resulting stack is in $(\Sigma^+)^+$. This avoids the special treatment of empty words or stacks. Furthermore, a collapse on level 2 summarises a non-empty sequence of pop_2-operations. For example, starting from $\bot_2$, we can apply a clone_2, a $\mathrm{push}_{\sigma,2}$, a clone_2, and finally a collapse. This sequence first creates a level 2 stack that contains 3 words and then performs the collapse and ends in the initial stack again. This example shows that clone_2-operations are responsible for the fact that collapse-operations on level 2 may remove more than one word from the stack.

For $s, s' \in \mathrm{Stck}(\Sigma)$, we call s' a substack of s if there are $n_1, n_2 \in \mathbb{N}$ such that $s' = {\mathrm{pop}_1}^{n_1}({\mathrm{pop}_2}^{n_2}(s))$. We write $s' \leq s$ if s' is a substack of s.

3.2. Collapsible Pushdown Systems and Collapsible Pushdown Graphs

Now we introduce collapsible pushdown systems and graphs (of level 2) which are analogues of pushdown systems and pushdown graphs using collapsible pushdown stacks instead of ordinary stacks.

Definition 3.1. A *collapsible pushdown system* of level 2 (CPS) is a tuple $S = (\Sigma, Q, \Delta, q_0)$ where Σ is a finite stack alphabet with $\bot \in \Sigma$, Q a finite set of states, $q_0 \in Q$ the initial state, and $\Delta \subseteq Q \times \Sigma \times Q \times \mathrm{OP}$ the transition relation.

For $q \in Q$ and $s \in \mathrm{Stck}(\Sigma)$ the pair (q, s) is called a *configuration*. We define labelled transitions on pairs of configurations by setting $(q_1, s) \vdash^{(q_2, op)} (q_2, t)$ if there is a $(q_1, \sigma, q_2, op) \in \Delta$ such that $\mathrm{Sym}(s) = \sigma$ and $op(s) = t$. The union of the labelled transition relations is denoted as $\vdash := \bigcup_{l \in Q \times \mathrm{OP}} \vdash^l$. We set $C(S)$ to be the set of all configurations that are reachable from $(q_0, \bot_2)$ via $\vdash$-paths. We call $C(S)$ the set of *reachable* or *valid* configurations. The *collapsible pushdown graph (*CPG*) generated by* S is

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

Example 3.2. The following example of a collapsible pushdown graph of level 2 is taken from [6]. Let $Q := \{0, 1, 2\}, \Sigma := \{\bot, a\}$, and Δ 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 in Σ. In our picture (see Figure 1), the labels are abbreviated as follows: $\mathrm{cl} := (1, \mathrm{clone}_2)$, $a := (0, \mathrm{push}_{a,2})$,

$a' := (2, \mathrm{push}_{a,2})$, $p := (2, \mathrm{pop}_1)$, and $\mathrm{co} := (0, \mathrm{collapse})$.

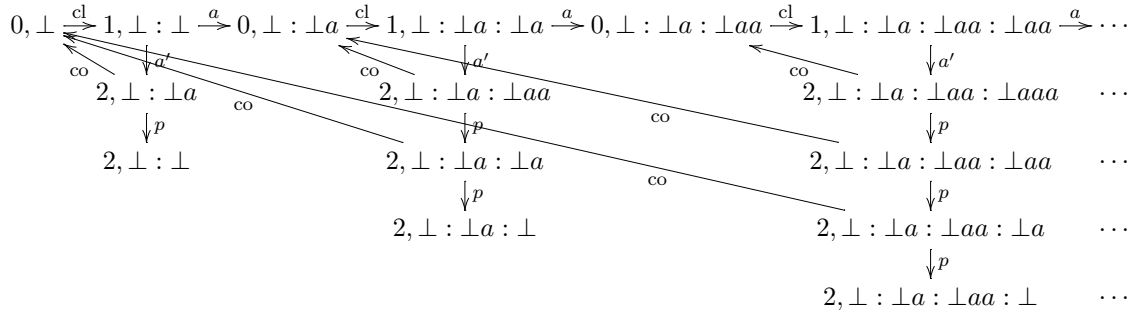

Figure 1: Example of a collapsible pushdown graph

Remark 3.3. Hague et al. [6] showed that modal μ-calculus model checking on level n CPG is n-EXPTIME complete. Note that there is an MSO interpretation which turns the graph of the previous example into a grid-like structure. Hence its MSO theory is undecidable.

The next definition introduces runs of collapsible pushdown systems.

Definition 3.4. Let S be a CPS. A run r of S of length n is a function

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

We write $\ln(r) := n$ and call r a run from $r(0)$ to $r(n)$. We say 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 $r \circ r'$ of r and r' in the obvious manner.

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

4. Encoding of Collapsible Pushdown Graphs in Trees

In this section we prove that 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 is a maximal list of words in the stack that share the same two first letters[2]. If we remove the first letter of every word of such a block, the resulting 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 corresponding block. This results in a tree in which every initial left-closed path 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 – via this encoding – each stack operation corresponds to a simple MSO-definable tree-operation. The main difficulty is to provide a tree-automaton that checks whether there is a run to the configuration represented by some tree. This problem is addressed in Section 5.

[2]see Figure 2 for an example of blocks and Definition 4.1 for their formal definition

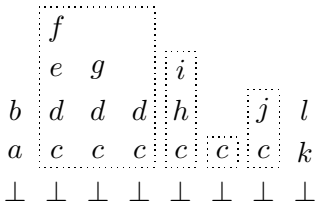

Figure 2: Example of blocks in a stack. These form a c-blockline.

As already mentioned, the encoding works by dividing stacks into blocks. The following definition makes 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 $s' = [ww_1] : [ww_2] : \ldots : [ww_n]$.

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 $\mathrm{Stck}(\Sigma)$ 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. 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 2-word.

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 default 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 ${}_hs_i := w_h : w_{h+1} : \ldots : w_i$. 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 $j' \leq j$, some fixed $(\sigma', l', k') \in \Sigma \times \{1, 2\} \times \mathbb{N}$, and appropriate $w_{j'}'' \in \Sigma^*$. For $\rho \in \big(\Sigma \times \{1, 2\}\big) \cup \{\varepsilon\}$, we define recursively the $\big(\Sigma \times \{1, 2\}\big) \cup \{\varepsilon\}$-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 $s \in \mathrm{Stck}(\Sigma)$.

Figure 3 shows a configuration and its encoding.

Remark 4.3. In this encoding, the first block of a (σ, l, k)-blockline 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 and that we always introduce a left-successor tree in order to encode letters that are higher in the stack.

The case $l = 2$ needs some closer inspection. Assume that some $d \in T := \mathrm{Enc}(s)$ is labelled $(\sigma, 2)$. Then it encodes a letter $(\sigma, 2, k)$ and this is not a cloned element.

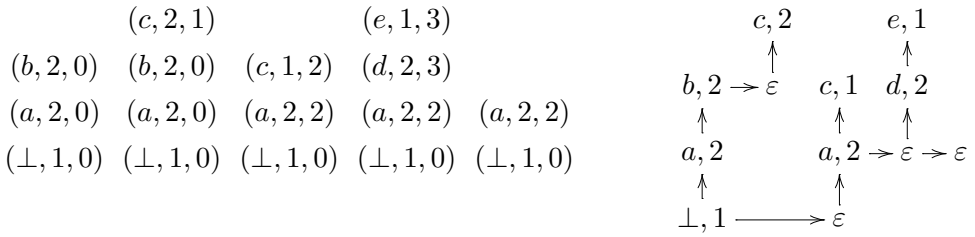

Figure 3: A stack s and its encoding $\mathrm{Enc}(s)$: right arrows lead to 1-successors (right successors), upward arrows lead to 0-successors (left successors).

Thus, k equals the numbers of words to the left of this letter $(\sigma, 2, k)$. We claim that $k = \left|\left\{e \in T \cap \{0,1\}^*1 : e \leq_{lex} d\right\}\right|$. The existence of a pair $e, e1 \in T$ corresponds to the fact that there is some blockline consisting of blocks $b_1 : b_2 : \ldots : b_n$ with $n \geq 2$ such that b_1 is encoded in $T_e \setminus T_{e1}$ and $b_2 : \ldots : b_n$ is encoded in T_{e1}. By induction, one easily sees that for each such pair $e, e1 \in T$ all the letters that are in words left of the letter encoded by $e1$ are encoded in lexicographically smaller elements. Furthermore, the size of $((0^*)1)^* \cap T$ corresponds to the number of words in s since the introduction of a 1-successor corresponds to the separation of the first block of some blockline from the other blocks. Each of these separation can also be seen as the separation of the last word of the first block from the first word of the second block of this blockline. Note that we separate two words that are next to each other in exactly one blockline. Putting these facts together our claim is proved.

Another view on this correspondence is the bijection $f : \{1, 2, \ldots, |s|\} \to R$ where $R := ((0^*)1)^* \cap \mathrm{dom}(T)$ and i is mapped to the i-th element of R in lexicographic order. $f(i)$ is exactly the position where the $(i-1)$-st word is separated from the i-th one for all $i \geq 2$. In order to state the properties of f, we need some more notation. We write π for the canonical projection $\pi : (\Sigma \times \{1,2\} \times \mathbb{N})^* \to (\Sigma \times \{1,2\})^*$ and w_i for the i-th word of s. Furthermore, let w_i' be a word such that, $w_i = (w_i \sqcap w_{i-1}) \circ w_i'$ (here we set $w_0 := \varepsilon$). Then the word along the path[3] from the root to $f(i)$ is exactly $\pi(w_i \sqcap w_{i-1})$ for all $2 \leq i \leq |s|$ and the path from $f(j)$ to $f(j) \circ 0^m$ for maximal $m \in \mathbb{N}$ is $\pi(w_j')$ for all $1 \leq j \leq |s|$.

In order to encode a configuration $c := (q, s)$, we add q as a new root of the tree and attach the encoding of s as the left subtree, i.e., $\mathrm{Enc}(c) := q(\mathrm{Enc}(s), \emptyset)$.

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 all trees T 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, $T(0) = (\bot, 1)$ and there are no other occurrences of $(\bot, 1)$ or $(\bot, 2)$.
(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 $t \in T$, if $T(t0) = (\sigma, 1)$ then $T(t10) \neq (\sigma, 1)$.

[3]By the word along a path from one node to another we mean the word consisting of the non ε-labels along this path.

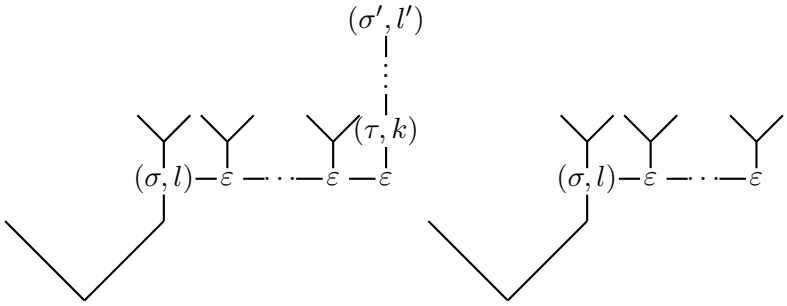

Figure 4: pop_2-operation

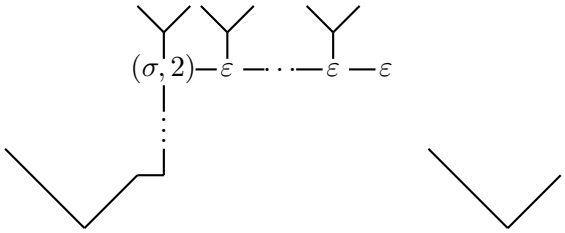

Figure 5: collapse-operation of level 2.

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 would contradict the maximality of the blocks in the encoding.

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

Our encoding turns the transitions of a CPG into regular tree-operations. The tree-operations corresponding to pop_2 and collapse can be seen in Figures 4 and 5. For the pop_2, note that if v_1 is the 0-successor of v_0 then v_0 and v_1 encode symbols in the same word of the encoded stack. As a pop_2 removes the rightmost word, we have to remove all the nodes encoding information about this word. As the rightmost leaf corresponds to the topmost symbol of the stack, we have to remove this leaf and all its 0-ancestors.

For the collapse (on level 2), we note that each ε represents a cloned element. The collapse induced by such an element produces the same stack as a pop_2 of its original version. The original symbol of the rightmost leaf is its first ancestor not labelled by ε.

Note that the operations corresponding to pop_2 and collapse are clearly MSO definable. All other transitions in CPG correspond to MSO definable tree-operations, too. Due to space restrictions we skip the details.

Lemma 4.7. *Let C be the set of encodings of configurations of a* CPS *S. Then there are automata $A_{(q,\mathrm{op})}$ for all $q \in Q$ and all* $\mathrm{op} \in \mathrm{OP}$ *such that for all $c_1, c_2 \in C$*

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

5. Recognising Reachable Configurations

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. It turns out that these are exactly those substacks of s that every run to s has to visit. Furthermore, the milestones of s are represented by the nodes of $\mathrm{Enc}(s)$: with every $d \in \mathrm{Enc}(s)$, we can associate a subtree of s which encodes a milestone. Furthermore, the substack relation on the milestones corresponds exactly to the lexicographical order $\leq_{lex}$ of the elements of $\mathrm{Enc}(s)$. For every $d \in \mathrm{Enc}(s)$ we can guess the state in which the corresponding milestone is visited for the last time by some run to s and we can check the correctness of this guess using MSO or, equivalently, tree-automata.

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

5.1. Milestones

Definition 5.1 (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 $0 \leq i < n$ and $w_i \sqcap w_{i+1} \leq w' \leq w_{i+1}$. We denote by $\mathrm{MS}(s)$ the set of milestones of s.

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

Lemma 5.2. *If s, t, m are stacks with $m \in \mathrm{MS}(t)$ but $m \not\leq s$, then every run from s to t visits m. Thus, for every run r from the initial configuration to s, the function*

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

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

In order to state the close correspondence between milestones of a stack s and the elements of $\mathrm{Enc}(s)$, we need the following definition.

Definition 5.3. Let $T \in \mathbb{T}_{\mathrm{Enc}}$ be a tree and $d \in T \setminus \{\varepsilon\}$. Then the *left and downward closed tree induced by d* is $LT(d, T) := T{\restriction_D}$ where $D := \{d' \in T : d' \leq_{lex} d\} \setminus \{\varepsilon\}$. Then we denote by $\mathrm{LStck}(d, T) := \mathrm{Dec}(LT(d, T))$ the *left stack induced by d*.

Remark 5.4. $\mathrm{LStck}(d, s)$ 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. In fact, $\mathrm{LStck}(d, s)$ is a milestone of s.

Lemma 5.5. *The map given by $g : d \mapsto \mathrm{LStck}(d, \mathrm{Enc}(s))$ is an order isomorphism between $(\mathit{dom}(\mathrm{Enc}(q, s)) \setminus \{\varepsilon\}, \leq_{lex})$ and $(\mathrm{MS}(s), \leq)$.*

Lemmas 5.5 and 5.2 imply that every run r decomposes as $r = r_1 \circ r_2 \circ \ldots \circ r_n$ where r_i is a run from the i-th milestone of $r(\mathrm{ln}(r))$ to the $(i+1)$-st milestone.

In order to describe the structure of the r_i, we have to introduce the notion of a loop. Informally speaking, a loop is a run r that starts and ends with the same stack s and which does not look too much into s.

Definition 5.6. Let r be a run of length n with $r(i) = (q_i, s_i)$ for all $0 \leq i \leq n$.

- r is called a *simple high loop* if $s_0 = s_n$ and if $s_0 < s_i$ for all $0 < i < n$.
- 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 $0 \leq i \leq n$, and $r{\restriction_{[2,n-1]}}$ is the composition of simple low loops and simple high loops of $\mathrm{pop}_1(s)$.
- r is called *loop* if it is a finite composition of low loops and high loops.

Lemma 5.7. *Let s be some stack, m_1, m_2 milestones of s, and r a run from m_1 to m_2 that never visits any other milestone of s. Then either $r = l_1 \circ p \circ l_2$ or $r = l_0 \circ c \circ l_1 \circ p_1 \circ l_2 \circ p_2 \circ l_3 \circ \ldots \circ p_n \circ l_{n+1}$ where each l_i is a loop, and all p_i, p, and c are runs of length 1, p performs one $\mathrm{push}_{\sigma,k}$, c performs one clone_2, and the p_i perform one pop_1 each.*

This lemma motivates why we only define low loops for stacks s with $\mathrm{CLvl}(s) = 1$. Whenever the topmost symbol of a milestone m is not a cloned element, then $\mathrm{pop}_1(m)$ is another milestone. Hence, the l_i can only contain low loops if they start at a stack with cloned topmost symbol. But any stack s with cloned topmost symbol and $\mathrm{CLvl}(s) = 2$ cannot be restored from $\mathrm{pop}_1(s)$ without passing $\mathrm{pop}_2(s)$ since a $\mathrm{push}_{\sigma,2}$-operation would create the wrong link-level.

From Lemma 5.7 we can derive that deciding whether there is a run from one milestone to the next is possible if we know the pairs of initial and final states of loops of certain stacks s. Hence we are interested in the sets $\mathrm{Loops}(s) \subseteq Q \times Q$ with $(q_1, q_2) \in \mathrm{Loops}(s)$ if and only if there is a loop from (q_1, s) to (q_2, s). The crucial observation is that $\mathrm{Loops}(s)$ may be calculated by a finite automaton reading $\mathrm{top}_2(s)$.

Lemma 5.8. *For every* CPS *there exists a finite automaton A that calculates*[4] *on input $w \in (\Sigma \times \{1,2\})^*$ the set $\mathrm{Loops}(s)$ for all stacks s such that $w = \pi(\mathrm{top}_2(s))$. Here, $\pi : (\Sigma \times \{1,2\} \times \mathbb{N})^* \to (\Sigma \times \{1,2\})^*$ is the projection onto the symbols and collapse-levels.*

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 state in which a run r to (q, s) visits each milestone as a certificate for the reachability of (q, s). To be precise, *a certificate for the reachability of* (q, s) is a map $f : \mathrm{dom}\big(\mathrm{Enc}(q, s)\big) \setminus \{\varepsilon\} \to Q$ such that there is some run r from $\bot_2$ to (q, s) and $f(d) = q$ if and only if $r(i) = \big(q, \mathrm{LStck}(d)\big)$ for i the maximal position in r where $\mathrm{LStck}(d)$ is visited.

Lemma 5.9. *For every* CPG *G, there is a tree-automaton that checks for each map*

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

whether f is a certificate of the reachability of (q, s), i.e., whether f is induced by some run r from the initial configuration to (q, s).

The proof of the lemma uses Lemma 5.8 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 certificate of reachability.

[4]We consider the final state reached by A on input w as the value it calculates for w.

As a tree-automaton may non-deterministically guess a certificate of the reachability of a configuration, the encodings of reachable configurations form a regular set.

5.3. Extension to Regular Reachability

By now, we have already established the tree-automaticity of each CPG G since we have seen that our encoding yields a regular image of the vertices of G and the transition relations are turned into regular relations of the tree encoding. Using similar techniques, we can improve this result:

Theorem 5.10. *If G is the ε-closure of some* CPG G' *then* (G, Reach) *is tree-automatic where* Reach *is the binary predicate that is true on a pair (c_1, c_2) of configurations if there is a path from c_1 to c_2 in G.*

Remark 5.11. Each graph in the second level of the Caucal-hierarchy can be obtained as the ε-contraction of some level 2 CPG (see [3]) whence all these graphs are tree-automatic.

For a CPS S let $R \subseteq \Delta^*$ be a regular language over the transitions of S. As collapsible pushdown graphs are closed under products with finite automata even the reachability predicate Reach_R with restriction to R is tree-automatic. Here, $\mathrm{Reach}_R xy$ holds if there is a path from x to y in $\mathrm{CPG}(S)$ that uses a sequence of transitions in R. If A is the automaton recognising R, we obtain that $\mathrm{Reach}_R(q,s)(q',s')$ holds in $\mathrm{CPG}(S)$ iff $\mathrm{Reach}\big((q,q_i),s\big)\big((q',q_f),s'\big)$ holds in $\mathrm{CPG}(S \times A)$ where q_i is the initial and q_f the unique final state of A. Using this idea one can define a CPG G' which is basically $\mathrm{CPG}(S \cup (S \times A))$ extended by transitions from (q,s) to $((q,q_i),s)$ and to $((q,q_f),s)$. $\mathrm{CPG}(S)$ as well as Reach_R w.r.t. $\mathrm{CPG}(S)$ are FO[Reach]-interpretable in G'. Hence we obtain:

Theorem 5.12. *Given a collapsible pushdown graph of level* 2*, its* FO[Reach_R] *theory is decidable for each regular $R \subseteq \Delta^*$.*

5.4. Computation of concrete tree-automatic representations of CPG

Up to now, we have only seen that there is a tree-automatic representation for each CPG. For computing a concrete representation, we rely on the following lemma.

Lemma 5.13. *Given some* CPS $S = (\Gamma, Q, \Delta, q_0)$*, some $q \in Q$, and some stack s, it is decidable whether (q,s) is a vertex of* $\mathrm{CPG}(S)$*.*

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. Hence we can construct an extension S' of S and a modal formula $\varphi_{q,s}$ such that there is some element $v \in \mathrm{CPG}(S')$ satisfying $\mathrm{CPG}(S'), v \models \varphi_{q,s}$ iff $(q,s) \in \mathrm{CPG}(S)$. S' basically contains new states for every substack of s and connects the different states via the appropriate pop_i-operations which are only applied if the topmost symbol of the stack agrees with the symbol we would expect when starting the pop_i-sequence in configuration (q,s).

From this lemma we can derive the computability of the automata in Lemma 5.8. Having obtained these automata, the construction of a tree-automatic representation of some CPG is directly derived from the proofs yielding the following theorem.

Theorem 5.14. *There is an algorithm that, given a level* 2 CPG G *and regular sets $R_1, \ldots, R_n \subseteq \Delta^*$, computes a tree-automatic representation of $(G, \mathrm{Reach}_{R_1}, \ldots, \mathrm{Reach}_{R_n})$.*

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 regular reachability predicates. This implies that the second level of the Caucal-hierarchy is tree-automatic. But our result can only be seen as a starting point for further investigations of the CPG hierarchy: are level 3 collapsible pushdown graphs tree-automatic? We know an example of a level 5 CPG which is not tree-automatic. But even when tree-automaticity of all CPG cannot be expected, the question remains whether all CPG have decidable FO theories. In order to solve this problem one has to come up with new techniques.

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 the decidability of extensions of first-order logic on tree-automatic structures would give us the corresponding decidability results for collapsible pushdown graphs of level 2.

References

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

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

[3] A. Carayol and S. 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] D. Caucal. On infinite terms having a decidable monadic theory. In *MFCS'02*, pages 165–176, 2002.

[5] J. 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. Springer, 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] D. Kuske. Theories of automatic structures and their complexity. In *CAI'09, Third International Conference on Algebraic Informatics*, volume 5725 of *LNCS*, pages 81–98. Springer, 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] J. W. Thatcher and J. 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.