Regular sets of higher-order pushdown stacks

Arnaud Carayol

Irisa – Campus de Beaulieu – 35042 Rennes Cedex – France
Arnaud.Carayol@irisa.fr

Abstract. It is a well-known result that the set of reachable stack contents in a pushdown automaton is a regular set of words. We consider the more general case of higher-order pushdown automata and investigate, with a particular stress on effectiveness and complexity, the natural notion of regularity for higher-order stacks: a set of level k stacks is regular if it is obtained by a regular sequence of level k operations. We prove that any regular set of level k stacks admits a normalized representation and we use it to show that the regular sets of a given level form an effective Boolean algebra. In fact, this notion of regularity coincides with the notion of monadic second order definability over the canonical structure associated to level k stacks. Finally, we consider the link between regular sets of stacks and families of infinite graphs defined by higher-order pushdown systems.

Introduction

Higher-order pushdown automata (hopdas for short) were introduced as a generalization of pushdown automata [Aho69,Gre70,Mas76]. Whereas a pushdown automaton works on a stack of symbols, a pushdown automaton of level 2 (or 2-hopda) works with a stack of level 1 stacks. In addition to the ability to push and to pop a symbol on the top-most level 1 stack, a 2-hopda can copy or remove the entire top-most level 1 stack. The k-hopdas are similarly defined for all level k and have been extensively studied as language recognizers [Dam82,Eng83].

Recently, the infinite structures defined by hopdas have received a lot of attention. First, in [KNU02,Cau02], the families of infinite terms defined by k-hopdas were shown to correspond to the solutions of safe higher-order recursive schemes. This study was later extended to the transition graphs of k-hopdas in [CW03]. Several characterizations of this hierarchy of families of infinite graphs were obtained. In particular, it was shown to coincide with a hierarchy defined using graph transformations by Caucal in [Cau96b] (see [Tho03] for a survey on this hierarchy).

The transition graphs of hopdas are defined in [CW03] by ε-closure of the reachability graphs. The vertices of the reachability graph of an hopda are the configurations reachable by the automaton from the initial configuration, and the edges represent the transition rules of the hopda. A major drawback of this definition is that it does not provide a direct description of the relations defining the edges of the transition graphs.

At level 1, the set of edges of the transition graph of a pushdown automaton can be given by prefix-recognizable relations [Cau96a,Cau03]. This characterization essentially uses the fact that the set of stack contents reachable by a pushdown automaton is regular. At level 2, due to the introduction of the copy operation, the set of words representing the stacks of stacks reachable by a 2-hopda is not a regular set of words. Hence, in order to obtain an internal representation of the transition graphs of k-hopdas, it is necessary to define a notion of regularity for sets of stacks of level k (k-stacks for short).

In this article, we study the notion of regularity for k-stacks induced by k-hopdas: a set of k-stacks is regular if it is obtained by a regular sequence of level k operations applied to the empty k-stack. In Section 2, we study the algebraic and algorithmic properties of this notion. We define a normal form for regular sets of k-stacks and use it to prove that they are closed under complementation. From the algorithmic point of view, the complexity of the normalization algorithm presented in Section 2.3 is a lower bound. We also show that the k-regular sets correspond to the sets definable by monadic second order logic over the canonical infinite structure associated with k-stacks. Finally, in Section 3, we use the notion of k-regularity to define an internal representation of the transition graphs of k-hopdas.

A complete version of this work including proofs can be found in [Car05].

1 Preliminary Definitions

1.1 Regular parts of a monoid

A *monoid* is given by a set M together with an associative internal product operation written $\cdot$ that admits a neutral element 1_M. The product operation is extended to subsets of M by $P \cdot Q = \{p \cdot q \mid p \in P \text{ and } q \in Q\}$. For any subset N of M, N^n is defined by $N^0 = \{1_M\}$ and $N^{n+1} = N \cdot N^n$. The iteration of N written N^* is equal to $\cup_{i\in\mathbb{N}} N^i$. Similarly, N^+ is defined as $\cup_{i>0} N^i$. The set of regular parts of a monoid M noted $\mathrm{Reg}(M)$ is the smallest set containing the finite subsets of M and closed under union, product and iteration.

A common example of monoid is the set of words over a finite alphabet Γ. A finite sequence of symbols (also called letters) in Γ is a word and the set of all words is written Γ^*. The empty word is noted ε.

1.2 Infinite graphs and transformations

Infinite graphs. Given a finite set Σ of edge labels and a countable set V, a *Σ-labeled graph* G is a subset of $V \times \Sigma \times V$. An element (s, a, t) of G is an *edge* of *source* s, *target* t and *label* a, and is written $s \xrightarrow[G]{a} t$ or simply $s \xrightarrow{a} t$ if G is understood. The set of all sources and targets of a graph is its *support* V_G. A sequence of edges $s_1 \xrightarrow{a_1} t_1, \ldots, s_k \xrightarrow{a_k} t_k$ with $\forall i \in [2, k],\ s_i = t_{i-1}$ is a *path* starting from s_1. We write $s_1 \xRightarrow{u} t_k$, where $u = a_1 \ldots a_k$ is the corresponding *path label*.

The *unfolding* $\mathrm{Unf}(G, r)$ of a Σ-labeled graph G from a vertex $r \in V_G$ is the Σ-labeled tree T satisfying for any $a \in \Sigma$ that $\pi \xrightarrow{a} \pi' \in T$ if and only if π and π' are two paths in G starting from r and $\pi' = \pi\, s \xrightarrow{a} t$.

Inverse mappings. Let $\bar{\Sigma}$ be a set of symbols disjoint from but in bijection with Σ. For any $x \in \Sigma$, we write $\bar{x}$ the corresponding symbol in $\bar{\Sigma}$. We extend every Σ-labeled graph G to a $(\Sigma \cup \bar{\Sigma})$-labeled graph $\bar{G}$ by adding reverse edges (i.e. $\bar{G} = G \cup \{s \xrightarrow{\bar{x}} t \mid t \xrightarrow{x} s \in G\}$). Let Γ be a set of edge labels, a *rational mapping* is a mapping $h : \Gamma \to \mathrm{Reg}((\Sigma \cup \bar{\Sigma})^*)$ which associates to every symbol from Γ a regular subset of $(\Sigma \cup \bar{\Sigma})^*$. If $h(a)$ is finite for every $a \in \Gamma$, we also speak of a *finite mapping*. We apply a rational mapping h to a Σ-labeled graph G by the inverse to obtain a Γ-labeled graph $h^{-1}(G) = \{s \xrightarrow{a} t \mid s \overset{h(a)}{\underset{\bar{G}}{\Longrightarrow}} t\}$.

Monadic second order logic. We define the *monadic second-order logic* (MSO for short) over Σ-labeled graphs as usual (see e.g. [EF95]). For any monadic second order formula $\varphi(x_1, \ldots, x_n)$ whose free-variables are first-order variables in $\{x_1, \ldots, x_n\}$ and for any vertices $u_1, \ldots, u_n \in V_G$, we write $G \models \varphi(u_1, \ldots, u_n)$ the fact that the graph G satisfies the formula φ when x_i is interpreted as u_i for all $i \in [1, n]$.

1.3 Higher-order stacks and operations

Stacks. A stack over a finite alphabet Γ is a word over Γ. We write $\mathrm{Stacks}_1(\Gamma) = \Gamma^*$ for the set of all stacks of level 1 and note $[\,]_1$ the empty level 1 stack ε. For all $k > 1$, a level k stack over Γ (or simply a k-stack) is a non-empty sequence of $(k-1)$-stacks over Γ. We write $\mathrm{Stacks}_k(\Gamma) = (\mathrm{Stacks}_{k-1}(\Gamma))^+$ the set of all k-stacks or simply Stacks_k if Γ is understood. The empty stack of level k is the k-stack containing only the empty $(k-1)$-stack and is written $[\,]_k$. The stack $[\,[\,AB\,]\,[\,ABC\,]\,[\,BA\,]\,]_2$ designates a 2-stack whose top most 1-stack is $[\,BA\,]_1$. The set of all stacks over Γ is written $\mathrm{Stacks}(\Gamma) = \bigcup_{k \in \mathbb{N}} \mathrm{Stacks}_k(\Gamma)$.

Operations. An operation on higher-order stacks is a (partial) function from $\mathrm{Stacks}(\Gamma)$ to $\mathrm{Stacks}(\Gamma)$ which preserves the level of the stack (i.e. the image of a k-stack is a k-stack). The level $|\rho|$ of an operation ρ is the smallest k such that $\mathrm{Dom}(\rho) \cap \mathrm{Stacks}_k \neq \emptyset$. The only operation for which the level is not defined is the empty function $\emptyset$. Note that for any two functions f and g, $f \cdot g$ designates the mapping associating to x the value $g(f(x))$.

The operations, we consider, respect the access mode of higher-order stacks that is to say, in a level $k+1$ stack only the top most level k stack can be accessed. It implies that for any level k operation ρ and for all $k' > k$, we have: $\rho([w_1, \ldots, w_n]_{k'}) = [w_1 \ldots \rho(w_n)]_{k'}$. Hence, it is only necessary to define a level k operation on Stacks_k, its definition for level of stacks greater than k is implicit.

The operations of level 1 for stacks over Γ are the well known push_x and pop_x for all $x \in \Gamma$. The operations added at level $k+1$ are the copy of the top most k-stack written copy_k and the inverse operation which is usually the destruction of the top most k-stack written pop_k. We consider a more symmetric

operation $\overline{\text{copy}}_k$ that only destroys the top most k-stack if it is equal to previous one[1]. These operations are formally defined by:

$$\begin{array}{ll}
\text{push}_x([\, x_1 \dots x_n \,]_1) & = [\, x_1 \dots x_n x \,]_1 \\
\text{pop}_x([\, x_1 \dots x_n x \,]_1) & = [\, x_1 \dots x_n \,]_1 \\
\text{copy}_k([\, w_1 \dots w_n \,]_{k+1}) & = [\, w_1 \dots w_n w_n \,]_{k+1} \\
\text{pop}_k([\, w_1 \dots w_{n+1} \,]_{k+1}) & = [\, w_1 \dots w_n \,]_{k+1} \\
\overline{\text{copy}}_k([\, w_1 \dots w_n w_n \,]_{k+1}) & = [\, w_1 \dots w_n \,]_{k+1}
\end{array}$$

In addition for each level k, we consider an operation written E_k to test whether the top most k-stack is empty (i.e $E_k([\ ]_k) = [\ ]_k$ and is undefined otherwise). This operation is usually avoided by considering a bottom symbol in the definition of the stacks but we wish to remain as general as possible. Moreover, we write id_k the identity seen as a level k operation.

We define $\text{Ops}_1 = \{\text{push}_x, \text{pop}_x \mid x \in \Gamma\} \cup \{E_1\}$. and $\text{Ops}_{k+1} = \text{Ops}_k \cup \{\text{copy}_k, \overline{\text{copy}}_k, E_{k+1}\}$. The set $\text{Ops}_k^* = \{\rho \mid |\rho| = k, \rho = \rho_1 \cdots \rho_n \text{ for } \rho_1, \dots, \rho_n \in \text{Ops}_k\} \cup \{\emptyset\}$ is a monoid for composition of functions with neutral element id_k.

Instructions. In order to work in a symbolic manner, we associate to each operation in Ops_k a symbol in an alphabet Γ_k called an *instruction*. Let $\overline{\Gamma}$ be a finite alphabet disjoint from but in bijection with Γ, we write $\bar{x}$ the letter of $\bar{\Gamma}$ corresponding to $x \in \Gamma$. The set of instructions of level k written Γ_k is defined by: $\Gamma_1 = \Gamma \cup \bar{\Gamma} \cup \{\bot_1\}$ and $\Gamma_{k+1} = \Gamma_k \cup \{\bot_{k+1}, k, \bar{k}\}$. We write $\Gamma_k^t = \{\bot_1, \dots, \bot_k\}$ and $\Gamma_k^o = \Gamma_k - \Gamma_k^t$. For all sequence $w \in \Gamma_k^*$, we designate by $\text{Last}(w)$ (resp. $\text{First}(w)$) the last (resp. first) element of Γ_k^o appearing in w.

We define a morphism[2] of monoid O from Γ_k^* to Ops_k^* associating to any sequence of instruction $w \in \Gamma_k^*$ the corresponding operation $O(w) \in \text{Ops}_k^*$ as follows: $O(\varepsilon) = \text{id}_k$,$O(x) = \text{push}_x$ and $O(\bar{x}) = \text{pop}_x$ for all $x \in \Gamma$, $O(i) = \text{copy}_i$, $O(\bar{i}) = \overline{\text{copy}}_i$, and $O(\bot_i) = E_i$ for all $i \in [1, k]$. The morphism O is extended to Γ_k^* in the canonical way. For example, the sequence of instructions $m = ab\bar{b}a1\bar{a}$ is evaluated to $O(m) = \text{push}_a\text{push}_b\text{pop}_b\text{push}_a\text{copy}_1\text{pop}_a = \text{push}_a\text{push}_a\text{copy}_1\text{pop}_a$. For any subset R of Γ_k^*, we write $O(R)$ the corresponding set of operations in Ops_k^* and $S(R) = O(R)([\ ]_k)$ the corresponding set of stacks in $\text{Stacks}_k(\Gamma)$.

For each k-stack s, there exists a minimal sequence of instructions $w \in \Gamma_k^*$ such that $S(w) = s$. It is easy to see that if $k = 1$, w belongs to Γ^* and if $k > 1$, w does not contain $\bar{k}$. In fact, a sequence of instructions $w \in (\Gamma_k^o \cup \{k\})^*$ (such that $S(w) \neq \emptyset$) is the minimal sequence of some level $k+1$ stack if and only if it does not contain $x\bar{x}$, $l\bar{l}$ or $\bar{l}l$ for any $x \in \Gamma \cup \overline{\Gamma}$ or any $l < k$. A sequence of instructions that does not contain such sub-sequences will be called *loop-free*. A k-stack s is a prefix of a k-stack s' (written $s \sqsubseteq s'$) if the minimal sequence of s is a prefix of the minimal sequence of s'.

[1] It is already known from [CW03] that hopdas defined using $\overline{\text{copy}}_k$ recognize the same languages as the ones defined using pop_k (see. Proposition 3.1).

[2] The definition is such that we always obtain a level k operation. So strictly speaking, there should be one evaluation mapping for each level.

Higher-order pushdown automata. An higher-order pushdown automaton P over $\text{Stacks}_k(\Gamma)$ (k-hopda for short) with Σ as an input alphabet is a tuple (Q, i, F, δ) where Q is a finite set of states, i is the initial state, F is the set of final states and $\delta \subset Q \times \Sigma \cup \{\varepsilon\} \times \Gamma_k^* \times Q$. The set of configurations of P noted C_P is $Q \times \text{Stacks}_k(\Gamma)$. For each $x \in \Sigma \cup \{\varepsilon\}$, P induces a transition relation $\xrightarrow{x} \subset C_P \times C_P$ defined by $(p, w) \xrightarrow{x} (q, w')$ if (p, x, ρ, q) belongs to δ and $w' = O(\rho)(w)$. For any word $u \in \Sigma^*$, we write $c \overset{u}{\Longrightarrow} c'$ if there exists a sequence $c \xrightarrow{x_1} c_1 \dots c_{n-1} \xrightarrow{x_n} c'$ and $u = x_1 \dots x_n$. A word $u \in \Sigma^*$ is accepted by P if $(i, [\,]_k) \overset{u}{\Longrightarrow} (f, w)$ for some $f \in F$.

2 Regular set of higher-order stacks

We consider the notion of regularity for sets of higher-order stacks that naturally extends what is known at level 1. A set of k-stack is *k-regular* if it is the set of stacks appearing in the reachable final configurations of a k-hopda. In other terms, a k-regular set is obtained by applying a regular set of operations in Ops_k^* to the empty stack of level k. The set of all k-regular subsets of $\text{Stacks}_k(\Gamma)$ is written $\text{Reg}_k(\Gamma) = \text{Reg}(\text{Ops}_k^*(\Gamma))([\,]_k) = S(\text{Reg}(\Gamma_k^*))$.

A normal form for k-regular sets is presented in Section 2.1 and it is proved in Section 2.2 that every k-regular set admits a normalized representation. Complexity related issues are dealt with in Section 2.3. Finally, Section 2.4 establishes that k-regular sets correspond to MSO-definable sets in the canonical infinite structure associated to k-stacks.

2.1 Normal forms

A regular set of instructions is not *per se* a useful representation of a set of stacks. We therefore define a normal form that gives a *forward* representation of the set of stacks in the sense that the set of instructions produced are loop-free.

At level 1, such a normal form is easily achieved: the set of minimal sequences of a 1-regular set is also regular [Büc64]. Hence, any 1-regular set admits a normalized representation in $\text{Norm}_1 = \text{Reg}(\Gamma^*)$. At level 2, a loop-free set of instruction does not contains $\bar{1}$. As illustrated by the following example, it is not possible to describe all sets in $\text{Reg}(\Gamma_2^*)$ without $\bar{1}$.

Example 2.1. The regular set of instructions $R = \{a, b\}^* 1 \{\bar{a}, \bar{b}\}^* \bar{b} 1 (\bar{a}\bar{a})^+ \bar{b} b a^* \bar{1}$ represents the set of stacks $S(R) = \{[\,[\,w b a^{2n} b w'\,][\,w b a^{2n}\,]\,]_2 \mid w \in \Gamma^*, w' \in \Gamma^* \text{ and } n \geq 0\}$. It can be proved that R is not equivalent to any set in $\text{Reg}(\,(\Gamma_1 \cup \{1\})^*\,)$. The problem is that the set $1(\bar{a}\bar{a})^+ \bar{b} b a^* \bar{1}$ correspond to the operation $\text{id}_2|_A$ where $A = \{[\,w_1 \dots w_n\,]_2 \mid w_n \in \Gamma^* b (aa)^+\}$ which tests, in a non-destructive manner, that the top-most 1-stack belongs to A.

Hence, in order to give a forward presentation of sets in $\text{Reg}(\Gamma_k^*)$, we need to introduce k-regular tests as a new operation. Theorem 2.1 will prove that we do not need additional operations. For any set Q of k-stacks, $\text{id}_k|_Q$ designates

the identity function restricted to the set of k'-stack whose top-most k-stack is in Q for all $k' > k$.

Definition 2.1 (Regular tests of level k). *Let T_k be an countable set of symbols with one symbol written T_R for each $R \in \mathrm{Reg}(\Gamma_k)$. We extend the evaluation mapping to $(\Gamma_k \cup T_k)^*$ by defining $O(T_R) = \mathrm{id}_k|_{S(R)}$.*

A subset R of $\mathrm{Reg}((\Gamma_k \cup T_k)^*)$ is loop-free is the set of obtained by removing the tests from R is.

In order to normalize a 2-regular set of stacks, it is necessary to give a normal form for sets of operations in $O(\mathrm{Reg}((\Gamma_k \cup T_k)^*))$. At level 1, a normal form was obtained in [Cau96a,Cau03]. It is proved that any $R \in \mathrm{Reg}((\Gamma_1 \cup T_1)^*)$, there exists a finite union $R' = \cup_{i\in I}\, U_i \cdot T_{W_i} \cdot \overline{V_i}$ where $U_i \in \mathrm{Reg}(\overline{\Gamma}^*)$, $V_i, W_i \in \mathrm{Reg}(\Gamma^*)$ for some finite set I with $\mathrm{Last}(U_i) \cap \overline{\mathrm{First}(V_i)} = \emptyset$[3] such that $O(R) = O(R')$. We write Rew_1 the set all $R \in \mathrm{Reg}((\Gamma_1 \cup T_1)^*)$ than can be expressed as such a finite union.

We now define Norm_k and Rew_k for level $k > 1$ as a straightforward extension of what is known at level 1:

- Norm_{k+1} is the set of all finite union of elements in $\mathrm{Norm}_k \cdot \mathrm{Reg}((k\,\mathrm{Rew}_k)^*)$,
- Pop_{k+1} and Push_{k+1} designate respectively the sets $\mathrm{Rew}_k \cdot \mathrm{Reg}((\bar{k}\,\mathrm{Rew}_k)^*)$ and $\mathrm{Rew}_k \cdot \mathrm{Reg}((k\,\mathrm{Rew}_k)^*)$,
- Rew_{k+1} is the set of all finite unions of sets of the form $U \cdot T_W \cdot V$ where $W \in \mathrm{Norm}_{k+1}$, $U \in \mathrm{Pop}_{k+1}$ and $V \in \mathrm{Push}_{k+1}$ with $\mathrm{Last}(U) \cap \overline{\mathrm{First}(V)} = \emptyset$ and $\overline{\mathrm{Last}(W)} \cap (\overline{\mathrm{Last}(U)} \cup \mathrm{First}(V)) = \emptyset$.

An equivalent characterization of the sets in Norm_{k+1} is through finite automata $A = (Q, i, F, \delta)$ labeled by a finite subset $\mathrm{Norm}_k \cup k \cdot \mathrm{Rew}_k$ such the only edges labeled by an element of Norm_k are starting from the initial state i and such that no transition comes back to i. We will call such an automaton a $(k+1)$-automaton. It is obvious that $L(A)$ belongs to Norm_{k+1} and that conversely, all $R \in \mathrm{Norm}_k$ is accepted by a k-automaton. By a slight abuse of language, we will say that a set R of k-stacks is accepted by A if $R = S(L(A))$.

The interest of this notion is that a deterministic version can be defined : a $(k+1)$-automaton labeled by $\{N_1, \ldots, N_n\} \subset \mathrm{Norm}_k$ and $\{R_1, \ldots, R_m\} \subset \mathrm{Rew}_k$ is deterministic if $S(N_i) \cap S(N_j) = \emptyset$ for $i \neq j$, $O(R_i) \cap O(R_j) = \emptyset$ for $i \neq j$ and A is deterministic. In a deterministic k-automaton, if two k-stacks are produced by two different executions (not necessarily successful) then they are different.

Proposition 2.1. *For all level k, any set in $S(\mathrm{Norm}_k)$ can be accepted by a deterministic k-automaton. Moreover, $S(\mathrm{Norm}_k)$ and $O(\mathrm{Rew}_k)$ are effective Boolean algebras*[4]*.*

[3] This corresponds to the right-irreducible prefix-recognizable relations in [Cau03].

[4] This result was already obtained in [Cau96a] for $O(\mathrm{Rew}_1)$.

2.2 Normalization

In this part, we prove by induction on the level k that for any set in $R \in \mathrm{Reg}(\Gamma_k^*)$, there exists a set $N \in \mathrm{Norm}_k$ such that $S(R) = S(N)$.

Characterization of loop languages Let $B = (Q, i, F, \delta)$ be an automaton labeled by Γ_{k+1}. For any two states p and $q \in Q$, the automaton B loops on a $(k+1)$-stack w starting in p and ending in q if there exists a sequence $(p, w) \xrightarrow[B]{x_1} (p_1, w_1) \dots \xrightarrow[B]{x_n} (p_n, w_n)$ such that $q = p_n$, $w_n = w$ and for all $i \in [1, n]$, $w \sqsubseteq w_i$.

It follows from the definition that "looping" behavior of an automaton only depends on the top-most k-stack. Hence, we define the *loop language* $L_{p,q}$ to be the set of k-stacks such that $w \in L_{p,q}$ if and only if for any $(k+1)$-stack w' with top-most k-stack w, B loops on w starting in p and ending in q. The loop languages allow us to define a loop-free equivalent of $L(A)$.

Proposition 2.2. *For any automaton A labeled by Γ_k, $L(A)$ is equivalent to a loop-free set in* $\mathrm{Reg}((\Gamma_k \cup \{T_{L_{p,q}} \mid p, q \in Q\})^*)$.

The loop languages are regular In order to simulate the copy_{k+1} and $\overline{\mathrm{copy}}_{k+1}$ operations on a level k stack, we use *alternation* to simultaneously perform the computation taking place on different copies of the stack.

Definition 2.2. *An alternating automaton A over Γ_k is a tuple (Q, i, Δ) where Q is a finite set of states, i is the initial state and $\Delta \subset Q \times \Gamma_k^t \cup \{\varepsilon\} \times 2^{Q \times \Gamma_k^o}$ is the set of transitions.*

A transition $(p, t, \{(q_1, a_1), \dots, (q_n, a_n)\}) \in \Delta$ is written $p, t \longrightarrow (q_1, a_1) \wedge \dots \wedge (q_n, a_n)$. An execution of A is a finite tree T with vertices V_T whose edges are labeled by Γ_k and whose vertices are labeled by $Q \times \mathrm{Stacks}_k(\Gamma)$. We write c the labeling mapping from V_T to $Q \times \mathrm{Stacks}_k(\Gamma)$. An execution satisfies the following conditions:

- $x \xrightarrow{a} y \in T$ implies $c(x) = (q, w)$ and $c(y) = (p, w')$ and $w' = O(a)(w)$.
- for all $x \in V_T$ with children $y_1, \dots, y_n$ (i.e $x \xrightarrow{a_1} y_1, \dots, x \xrightarrow{a_n} y_n$), if $c(y_i) = (q_i, w_i)$ then there exists $q, t \longrightarrow (q_1, a_1) \wedge \dots \wedge (q_n, a_n) \in \Delta$ such that $O(t)(w)$ is defined[5].

An execution T of A is accepting a stack w if the root of T is labeled by (i, w). We write $S(A) \subset \mathrm{Stacks}_k(\Gamma)$ the set of stacks accepted by A.

The following lemma states that the loop languages defined in the previous part are accepted by alternating automata labeled by Γ_k.

Lemma 2.1. *For any automaton B labeled by Γ_{k+1}, there exists an alternating automaton labeled by Γ_k accepting $L_{p,q}$.*

[5] The set of final states is implicitly given by transitions of the form $q, t \longrightarrow \emptyset$

In order to prove that the language accepted by an alternating automaton over Γ_k is k-regular, we define a normal form for alternating automata in which at most one execution can be sent for a given instruction in Γ_k^o and such that no execution contains two vertices labeled by the same stack. More formally, a *normalized* alternating automaton over Γ_k is an alternating automaton with transitions of the form $q, t \longrightarrow (q_1, b_1) \wedge \ldots (q_n, b_n)$ with $b_i \neq b_j$ for $i \neq j$ for which no execution tree T contains $x \xrightarrow[T]{a\bar{a}} y$ for $a \in \Gamma_k^o$. The following proposition establishes that any alternating automaton can be transformed into an equivalent normalized alternating automaton.

Proposition 2.3. *For any alternating automaton A labeled by Γ_k, an equivalent normalized alternating automaton B labeled by Γ_k can be constructed in $\mathcal{O}(2^{p(|B|)})$ for some polynomial p.*

We can now establish that the languages accepted by alternating automata labeled by Γ_k are k-regular languages.

Proposition 2.4. *The sets of k-stacks accepted by alternating automaton labeled by Γ_k are k-regular sets.*

Proof (Sketch). First, we establish that the languages accepted by normalized alternating automaton over Γ_{k+1} are loop-free languages in $\mathrm{Reg}((\Gamma_k^o \cup \{k\} \cup T_k^A)^*)$ where T_k^A designates the tests by languages accepted by alternating automaton over Γ_k. The result follows by induction on the level k combining the above property and Proposition 2.3.

□

Normalization result. We proceed by induction on the level k of stacks. It follows from the Proposition 2.2, Lemma 2.1 and Proposition 2.4, that any set of instructions in $R \in \mathrm{Reg}(\Gamma_k^*)$ is equivalent to a loop-free subset of $\mathrm{Reg}((\Gamma_k \cup T_k)^*)$.

Proposition 2.5. *For all loop-free set R in $\mathrm{Reg}((\Gamma_k \cup T_k)^*)$ with tests languages in $S(\mathrm{Norm}_k)$, there exists a $R' \in \mathrm{Norm}_k$ such that $S(R) = S(R')$.*

Note that to achieve this normalization we need to determinize the languages appearing in the tests. However, if the languages appearing in the tests are already determinize the transformation is polynomial. The normalization result is obtained by a straightforward induction.

Theorem 2.1 (Normalization). *Every k-regular set can be accepted by a k-automaton. Hence, $\mathrm{Reg}_k = S(\mathrm{Reg}(\Gamma_k^*)) = S(\mathrm{Reg}(\mathrm{Norm}_k))$ is an effective Boolean algebra.*

2.3 Complexity and lower bounds

In order to evaluate the complexity of the normalization algorithm, we need a notation for towers of exponentials. We define $2\uparrow^0(n) = n$ and $2 \uparrow^{k+1} (n) = 2^{2\uparrow^k(n)}$.

The complexity of the algorithm obtained in the previous section when applied to a k-regular set of stacks is $\mathcal{O}(2\uparrow^{2k+1}(p(n)))$ where p is a polynomial. This complexity can be improved by transforming directly an alternating automaton over Γ_k into a deterministic k-automaton.

Theorem 2.2 (Lower bound).

1. *For any alternating automaton A labeled by Γ_k, there exists a deterministic k-automaton accepting $S(A)$ which can be computed in time $\mathcal{O}(2\uparrow^k(p(n)))$ for some polynomial p.*
2. *For any automaton A labeled by Γ_k, there exists a k-automaton accepting $S(A)$ which can be computed in time $\mathcal{O}((2\uparrow^{k-1}(p(n))))$*

It is easy to see that normalization can be used to test the emptiness of a regular set of k-stacks. In fact, for any set $R \in \mathrm{Reg}(\Gamma_k^*)$, the normalized representation of $R \cdot \Gamma_k^* \cdot \bot_k$ contains $[\ ]_k$ if and only if $S(R)$ is not empty. Therefore, the normalization can be used to test the emptiness of the language accepted by a k-hopda (i.e it is equivalent to the emptiness of the set of reachable final configurations).

In [Eng83], the author proves that $\mathcal{O}(2\uparrow^{k-1}(p(n)))$ is a lower bound for the emptiness problem of k-hopda. It follows the complexity of the normalization algorithm obtained in Theorem 2.2 is a lower bound.

2.4 MSO-definability over Δ_2^n

In this part, we fix $\Gamma = \{a, b\}$. The canonical infinite structure associated to words in Γ^* is the infinite binary tree Δ_2. A set $X \subset V_G$ (resp. $Y \subset V_G \times V_G$) is MSO-definable in G if there exists a formula $\varphi(x)$ (resp. $\varphi(x, y)$) such that X (resp. Y) is the set of $u \in V_G$ (resp. $(u, v) \in V_G \times V_G$) such that $G \models \varphi(u)$ (resp. $G \models \varphi(u, v)$). It is well known that the MSO-definable sets in Δ_2 are the regular sets of words. Moreover, Blumensath [Blu01] proved that the relations MSO-definable in Δ_2 are the prefix-recognizable relations (i.e $O(\mathrm{Rew}_1)$).

In order to investigate the notion of MSO-definable set of k-stacks, we consider the canonical structure Δ_2^k associated to k-stacks (see Figure 1). The graph Δ_2^k is labeled by $\Sigma_k = \{a, b, 1, \ldots, k\}$, its set of vertices is $\mathrm{Stacks}_k(\Gamma)$ and it edges are defined by: $w \xrightarrow{i} w'$ if $w' = O(i)(w)$ for $i \in \Sigma_k$. For instance, the set of k-stacks whose top-most 1-stack is empty is defined by the formula $\varphi(x) = \neg(\exists y.y \xrightarrow{a} x) \wedge \neg(\exists y.y \xrightarrow{b} x)$.

Proposition 2.6. *The set of k-stacks MSO-definable in Δ_2^k are the k-regular sets and the sets of relations MSO-definable in Δ_2^k are the relation in $O(\mathrm{Rew}_k)$.*

3 Higher-order pushdown graphs

In this section, we consider infinite graphs associated to hopdas and we show how the notion of k-regularity can be used to study their structure.

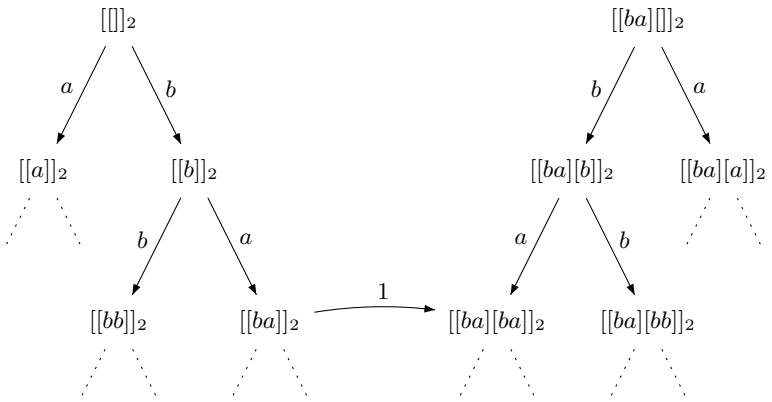

Fig. 1. The canonical structure associated to level 2 stacks.

3.1 Reachability graphs

The most natural way to associate an infinite graph to a k-hopda is to consider its *reachability graph*: the set of vertices is the set of configurations reachable from the initial configuration and the edges are given by the k-hopda. Due to the restriction by reachability, it is possible to encode in the stack a finite information corresponding to rational tests. The idea behind this encoding is taken from [Cau02] and was used in [CW03] to simulate the $\overline{\text{copy}}_k$ operation using pop_k.

Proposition 3.1. *The reachability graphs of k-hopdas enriched with k-regular tests and the reachability graphs of the k-hopda are definable only with the* pop_k *operation instead of* $\overline{\text{copy}}_k$ *up to isomorphism.*

From a structural point of view however, this approach is limited as the graphs obtained are necessarily directly connected. For instance, Δ_2^k is not the reachability graph of any hopda.

3.2 Configuration and transition graphs

The configuration graph of a k-hopda P is obtained by restricting the transition relation induced by P to a k-regular set of configurations R:

$$\{w \xrightarrow{x} w' \mid x \in \Sigma \cup \{\varepsilon\}, w \xrightarrow[P]{x} w' \text{ and } w, w' \in R\}$$

As the set of k-stacks reachable from the initial configuration is a k-regular set, the reachability graph is a particular case of configuration graph. The following property gives a structural characterization of configurations graphs.

Proposition 3.2. *The configuration graphs of k-hopdas are the graphs obtained by a k-fold iteration of unfolding and finite inverse mapping starting from a finite graph.*

The transition graphs are defined as the ε-closure of the configuration graphs:

$$\{w \xrightarrow{x} w' \mid x \in \Sigma, w \xRightarrow[P]{x} w' \text{ and } w, w' \in R\}$$

for some k-regular set of configurations R. The following proposition summarizes various equivalent characterization of the transition graphs of k-hopda.

Proposition 3.3 ([CW03]). *The family of transition graphs of k-hopdas is equal up-to isomorphism to the families of:*

- *graphs whose edges are defined by relations in $O(\mathrm{Rew}_k)$,*
- *graphs obtained by a k-fold iteration of unfolding and inverse rational mapping starting from a finite graph,*
- *graphs MSO-definable in Δ_2^k.*

Conclusion

We define a natural notion of regularity for higher-order pushdown stacks shares some of the most important properties of regular sets of words. In fact, we proved that they can be accepted by a deterministic machine and form an effective Boolean algebra. Furthermore, in the same way as regular set of words corresponds to the MSO-definable sets of words over the infinite binary tree, regular sets of k-stacks correspond to the MSO-definable sets over the canonical infinite structure associated to k-stacks. From the algorithmic point of view, we provided a normalization algorithm whose complexity is in fact a lower bound. To demonstrate the usefulness of this notion to work with hopdas, we used it to give a simple characterization of transitions graphs of hopdas similar to prefix-recognizable graphs for level 1.

From the model checking point of view, this notion could be used to extend the work done on pushdown automaton [BEM97] or on sub-families of hopdas [BM04]. From the structural point of view, it would be interesting to obtain an internal characterization of the transition graphs of k-hopdas of bounded degree. At level 1, the prefix recognizable graphs of bounded degree correspond to the configuration graphs of pushdown automata. It would be interesting to know how this property extends to higher-order.

Acknowledgment

The author would like to thank Didier Caucal and Antoine Meyer for their support, Jacques Sakarovitch for his suggestions on this work and the anonymous referees for their detailed and thorough reports.

References

[Aho69] Alfred V. Aho. Nested stack automata. *J. ACM*, 16(3):383–406, 1969.

[BEM97] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model checking. In *Proc. of CONCUR '97*, volume 1243 of *LNCS*, pages 135–150. Springer Verlag, 1997.

[Blu01] A. Blumensath. Prefix-recognisable graphs and monadic second-order logic. Technical Report AIB-2001-06, RWTH Aachen, 2001.

[BM04] A. Bouajjani and A. Meyer. Symbolic reachability analysis of higher-order context-free processes. In *Proc. of FSTTCS '04*, volume 3328 of *LNCS*, pages 135–147. Springer Verlag, 2004.

[Büc64] R. Büchi. Regular canonical systems. In *Archiv fur Math. Logik und Grundlagenforschung 6*, pages 91–111, 1964.

[Car05] A. Carayol. Regular sets of higher-order pushdown stacks. Extended version. A preliminary version of this article is available at the address `http://www.irisa.fr/galion/acarayol/`, 2005.

[Cau96a] D. Caucal. On infinite transition graphs having a decidable monadic theory. In *Proc. of ICALP '96*, volume 1099 of *LNCS*, pages 194–205. Springer Verlag, 1996.

[Cau96b] D. Caucal. *Sur des graphes infinis réguliers*. Habilitation thesis, Université de Rennes 1, 1996.

[Cau02] D. Caucal. On infinite terms having a decidable monadic theory. In *Proc. of MFCS '02*, volume 2420 of *LNCS*, pages 165–176. Springer Verlag, 2002.

[Cau03] D. Caucal. On infinite transition graphs having a decidable monadic theory. *Theor. Comput. Sci.*, 290:79–115, 2003.

[CW03] A. Carayol and S. Wöhrle. The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *Proc. of FSTTCS '03*, volume 2914 of *LNCS*, pages 112–123. Springer Verlag, 2003.

[Dam82] W. Damm. The OI- and IO-hierarchies. *Theor. Comput. Sci.*, 20(2):95–207, 1982.

[EF95] H.D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer-Verlag, 1995.

[Eng83] J. Engelfriet. Iterated pushdown automata and complexity classes. In *Proc. of STOC '83*, pages 365–373. ACM Press, 1983.

[Gre70] S. Greibach. Full AFL's and nested iterated substitution. *Inf. Control*, 16(1):7–35, 1970.

[KNU02] T. Knapik, D. Niwiński, and P. Urzyczyn. Higher-order pusdown trees are easy. In *Proc. of FoSSaCS '02*, volume 2303 of *LNCS*, pages 205–222. Springer Verlag, 2002.

[Mas76] A.N. Maslov. Multilevel stack automata. *Problemy Peredachi Informatsii*, 12:55–62, 1976.

[Tho03] W. Thomas. Constructing infinite graphs with a decidable MSO-theory. In *Proc. of MFCS '03*, volume 2747 of *LNCS*, pages 113–124. Springer Verlag, 2003.