Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE @ DIRECT®

Information Processing Letters 87 (2003) 187–193

Information Processing Letters

www.elsevier.com/locate/ipl

Deterministic finite automata with recursive calls and DPDAs [☆]

Jean H. Gallier [a], Salvatore La Torre [b,*], Supratik Mukhopadhyay [c]

[a] *University of Pennsylvania, Philadelphia, PA 19104, USA*
[b] *Università degli Studi di Salerno, Dipartimento di Informatica ed Applicazioni, Via S. Allende, Baronissi (SA) 84081, Italy*
[c] *West Virginia University, Morgantown, WV 26506, USA*

Received 4 December 2002; received in revised form 6 March 2003

Communicated by H. Ganzinger

Abstract

We study deterministic finite automata (DFA) with recursive calls, that is, finite sequences of component DFAs that can call each other recursively. DFAs with recursive calls are akin to recursive state machines and unrestricted hierarchic state machines. We show that they are language equivalent to deterministic pushdown automata (DPDA).
© 2003 Elsevier B.V. All rights reserved.

Keywords: Formal Languages

1. Introduction

Finite automata are a very intuitive and widely studied formalism. They are a natural framework for modeling and studying finite state systems. Descriptions using automata are useful to represent the control flow of a computer program, and in general the behavior of a digital system. This makes them suitable for formal analysis via well-founded technologies, such as model checking. In the simplest setting, a finite automaton consists of a labeled graph whose vertices correspond to system states and edges correspond to system transitions. To describe complex systems using finite automata, several extensions are useful.

In this paper, we study an extension of the concept of a DFA by allowing certain transitions to be "calls" to other DFAs. The automata we define are similar to restricted types of augmented transition networks (ATN) used in natural language processing [18], and to other hierarchic state machine models that have been proposed in different contexts where the need for a nesting capability is either natural or convenient. Such machines have been studied in the context of interprocedural dataflow analysis. They can model control-flow graphs of procedures in programming languages such as C. Our model is also related to the strict deterministic grammars of Harrison and Havel [11]. In [12] a class of nondeterministic finite automata with recursive calls is proved to correspond to the unambiguous context-free languages. In the design and verification

[☆] This research was partially supported by the supported by the MIUR in the framework of the project "Metodi Formali per la Sicurezza" (MEFISTO) and MIUR grant 60% 2002.
[*] Corresponding author.
E-mail addresses: jean@saul.cis.upenn.edu (J.H. Gallier), slatorre@unisa.it (S. La Torre), supratik@saul.cis.upenn.edu (S. Mukhopadhyay).

of large systems the use of hierarchic paradigms has been advocated by many authors both to have succinct descriptions and exploit modularity [9,4,1]. Recent results have been concerned with the verification of recursive finite state machines [3,2].

Here, we are interested in recursive (deterministic) finite state machines from a formal languages theoretic perspective. It is well known that (deterministic) finite automata define exactly the class of regular languages. Our aim is to extend this class of automata to capture the deterministic context-free languages. We define a DFA with recursive calls as a finite sequence of component DFAs that can call each other recursively. There are three kinds of transitions: ordinary transitions, calls and returns (from calls). A call corresponds to entering another component DFA, and a return to exiting it. The main limitation we place on the model is that if from a state q there is a call to a DFA then this is the only transition from q. This restriction is needed to preserve the determinism of the model. For any call we allow multiple return points, i.e., *multiple exits* in the component DFAs.

The main result of this paper is that the class of DFAs with recursive calls defines exactly the class of deterministic context-free languages. To prove that this class of automata accepts only deterministic context-free languages, we use the *atomic normal form* of deterministic pushdown automata given in [8]. The completeness result, that is, each deterministic context-free language has an equivalent DFA with recursive calls accepting it, is shown by translating an infinite tree of a recursion scheme to a DFA with recursive calls and using the fact that such trees can be used to characterize the class of deterministic context-free languages [8]. It is worth noticing that if in our model we allow nondeterminism on ordinary transitions (i.e., transitions that are neither calls nor returns), also nondeterministic context-free languages can be accepted. For example, consider the language $L = \{ww^R \mid w \in \Sigma\}$, where Σ is an alphabet. We can define an automaton with deterministic recursive calls that accepts L, as follows. The automaton consists of a single component that can call itself recursively. Calls are used to push symbols onto the stack and returns to pop symbols from the stack. The top of the stack can be stored in the state of the current activation of the component. Thus, nondeterminism is needed only to guess whether the current symbol is the last symbol of w or not, and only in the second case, a recursive call is made after this symbol is read.

The equivalence problem for deterministic pushdown automata has been recently shown to be decidable [15]. Another proof of this result can be found in [17]. Besides the importance of such a result the DPDA equivalence problem has a number of important implications which have been discussed in [16]. By our results, we derive from the decidability of the DPDA equivalence problem also the decidability of the equivalence problem for DFAs with recursive calls. If we see the DFAs with recursive calls as recursive program schemes, we have also that a certain kind of strong equivalence is decidable (we might call it “flow equivalence”).

Unwinding the recursion in a DFA with recursive calls we obtain a deterministic automaton whose transition graph is an infinite graph (*infinite automaton*). It is possible to prove that this infinite automaton is the “initial fixpoint” (in the sense of Lehmann [13]) of a “substitution functor” induced by the set of DFAs with recursive calls. In fact, such automaton turns out to be the colimit of a sequence of finite approximating DFAs.

The rest of the paper is organized as follows. In Section 2 we introduce the model and recall the main definitions. In Section 3 we prove that DFAs with recursive calls characterize exactly the class of deterministic context-free languages. We conclude in Section 4 with further remarks.

2. The model

In this section we introduce the notion of deterministic finite automata with recursive calls.

Let Σ be a finite alphabet and let $\Phi = \{F_1, \dots, F_N\}$ be a set of function symbols, each having arity $r(F_i) \geqslant 0$. Let $M = \max\{r(F_i) \mid 1 \leqslant i \leqslant N\}$ and let $[1, M]$ denote the set $\{1, \dots, M\}$ (with $[1, 0] = \emptyset$). The symbols of Φ will be called nonterminals, while the symbol in Σ will be called terminals. The elements of Φ can be considered as procedure names.

A deterministic finite automaton with recursive calls is a set of component DFAs that can call each other recursively. Calls and returns are modeled as special transitions that do not read input symbols.

Moreover, if from a state q there exists a call transition then this is the only transition leaving from q. If p is the state reached after a call, it can have only an entering transition, and the transitions exiting p are exactly the returns from this call and each of them corresponds to an exit of the called component. There exists a one-to-one correspondence between the exits of a called component and the returns of the calling component. For any call we allow multiple return points, i.e., *multiple exits* in the component DFAs.

Formally, a *deterministic finite automaton with recursive calls* consists of a finite sequence $D = \langle F_1 \Leftarrow D_1, \ldots, F_N \Leftarrow D_N \rangle$ of DFA definitions, where D_i is a DFA which can issue calls $F_1, \ldots, F_N$ respectively to any of the component DFAs $D_1, \ldots, D_N$. The first definition $F_1 \Leftarrow D_1$ is considered to be the "main definition". Given an $F_i \in \Phi$, a DFA definition $F_i \Leftarrow D_i$ for F_i of arity m is a 6-tuple $\langle Q, \Sigma \cup \Phi \cup [1, M], \delta, in, OUT, FINAL \rangle$ where:

- Q is a finite set of states.
- $in \in Q$ is the entry state.
- $FINAL \subseteq Q$ is the set of final states.
- $OUT : Q \to [1, m]$ is a partial function whose domain is denoted as $dom(OUT)$; each state p for which $OUT(p) = j$ is called an exit state with index j.
- $\delta : Q \times \Sigma \cup \Phi \cup [1, M] \to Q$ is a partial function called the transition function and satisfies the following conditions:
  1. For every $p \in Q$ and $F_j \in \Phi$, if $q = \delta(p, F_j)$ is defined, then $\delta(p, \alpha)$ is not defined for any other $\alpha \in \Sigma \cup \Phi \cup [1, M]$, that is, $\delta(p, F_j)$ is the only transition defined from p.
  2. If $\delta(p, F_j) = q$ then $\delta(q, k)$ is defined only for k such that $1 \leqslant k \leqslant r(F_j)$, and $\delta(q, \alpha)$ is not defined for any $\alpha \in \Sigma \cup \Phi$.
  3. If defined, $\delta(p, F_j)$ is not a final state and has no other incoming edges.

  Furthermore, the entry state *in* has no incoming transitions, every state in $dom(OUT)$ has no outgoing transitions and every state lies on a path from *in* to some exit state.

As an example of a DFA with recursive calls, consider the automaton D given by the tuple of definitions $\langle F_1 \Leftarrow D_1, F_2 \Leftarrow D_2, F_3 \Leftarrow D_3 \rangle$, where $\{F_1, F_2, F_3\}$ is a set of nonterminals with $r(F_1) = 0$, $r(F_2) = 1$, and $r(F_3) = 2$. The details on the DFAs D_1, D_2 and D_3 are shown in Fig. 1. The alphabet of terminal symbols is $\Sigma = \{a, b, c, d\}$, the entry states are in_i for each F_i for $i = 1, 2, 3$, $OUT(out_2) = 1$, and $OUT(out_3^i) = i$ for $i = 1, 2$.

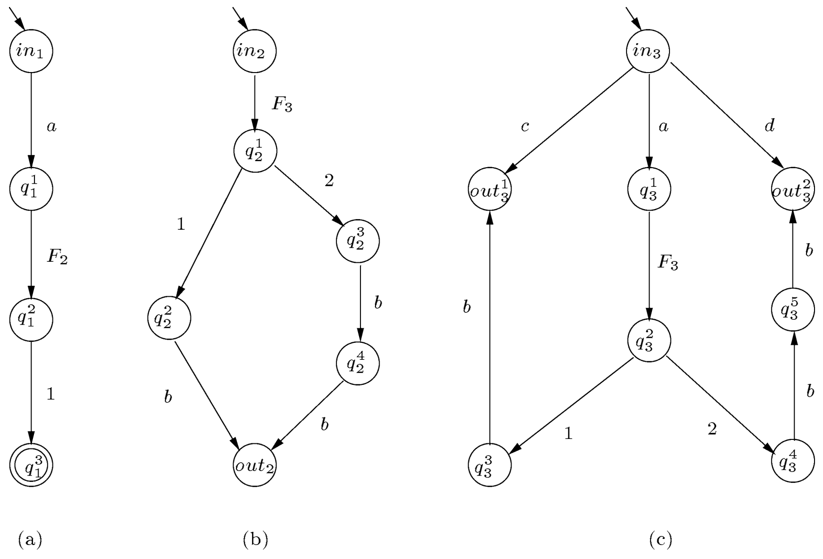

Fig. 1. Graphical representation of the DFAs D_1 (a), D_2 (b), and D_3 (c).

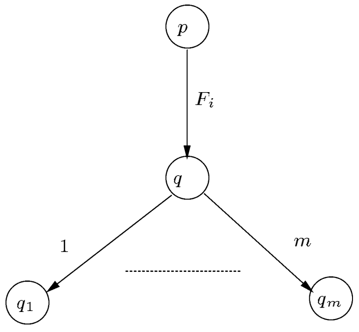

Fig. 2. Recursive call F_i.

For the sake of simplicity, in the following for any DFA with recursive calls $D = \langle F_1 \Leftarrow D_1, \ldots, F_N \Leftarrow D_N \rangle$, we assume that $Q_i \cap Q_j = \emptyset$ whenever $i \neq j$.

We define the language $L(D)$ accepted by a DFA with recursive calls D by defining the notion of a computation. For that, we view $D = \langle F_1 \Leftarrow D_1, \ldots, F_N \Leftarrow D_N \rangle$ as a context-free graph grammar with start symbol F_1. If D is a DFA with calls from the set $\{F_1, \ldots, F_N\}$ we say that R rewrites to S in one step using productions in D, denoted as $R \Rightarrow_D S$ (or $R \Rightarrow S$ when D is understood), if S is obtained from R by substituting D_i for some occurrence of a call F_i to D_i in R. More precisely, if the occurrence of F_i in R is as indicated in Fig. 2, the graph D_i is substituted in such a way that the entry of D_i is identified with p, and each exit state indexed with j ($1 \leqslant j \leqslant m$) is identified with q_j. Note that the node q always disappears together with all edges adjacent to it. The result of substituting D_i for F_i at q in R is denoted as $R[q \to D_i]$.

Let $\Rightarrow^+$ be the transitive closure of $\Rightarrow$ and $\Rightarrow^*$ be its reflexive and transitive closure. An *instantaneous description* (ID) is a triple $\langle R, p, u \rangle$, where R is a graph derivable from D_1 ($D_1 \Rightarrow^* R$), p is a state in R and u is the remaining input. A *computation* is a sequence of ID's such that given two consecutive IDs ID_1 and ID_2, where $ID_1 = \langle R, p, u \rangle$, we have that:

(1) If $u = av$, $a \in \Sigma$ and $q = \delta(p, a)$ is defined, then $ID_2 = \langle R, q, v \rangle$.
(2) If $q = \delta(p, F_i)$ is defined then $ID_2 = \langle R[q \leftarrow D_i], in_i, u \rangle$. This corresponds to a call F_i to D_i. In this move, D_i is substituted for F_i in R, and control is passed to the entry of D_i, no input being read.

Note that there is no need for computation steps $\delta(q, k)$ because substitution has been defined in such a way that these edges are deleted.

The language $L(D)$ accepted by a DFA with recursive calls D is the set of all strings u such that there exists a computation from $\langle D_1, in_1, u \rangle$ to an accepting configuration $\langle R, f, \varepsilon \rangle$, where f is a final state and with ε we denote as usual the empty string. It is easy to verify that the DFA with recursive calls D in Fig. 1 accepts the language $L = \{a^n c b^n \mid n \geqslant 1\} \cup \{a^n d b^{2n} \mid n \geqslant 1\}$.

3. Equivalence of DPDAs and DFAs with recursive calls

We assume that the reader is familiar with pushdown automata. A formal definition can be found in [10]. We recall a result from [8] that gives a characterization of deterministic pushdown automata in *atomic normal form*.

A DPDA M is in atomic normal form if it has the following structure:

(1) The set of states is partitioned into three disjoint subsets K_{read}, K_{push} and K_{pop} (states without outgoing transitions are considered in K_{read}).
(2) Transitions are of the form:
   (a) *read move*: being in a read state, M reads the next input regardless of the symbol on the top of the stack and changes its state without changing the stack. This is the only move advancing the input.
   (b) *push move*: being in a push state, M on an ε-move pushes the current state on top of the stack and changes its state (regardless of the top of the stack).
   (c) *pop move*: being in a pop state, M on an ε-move pops the top of stack and changes its state.
(3) A pop move never follows immediately a push move.
(4) Every accepting state is a read state.

The following theorem holds.

Theorem 1 [8]. *Every DPDA is equivalent to a DPDA in atomic normal form.*

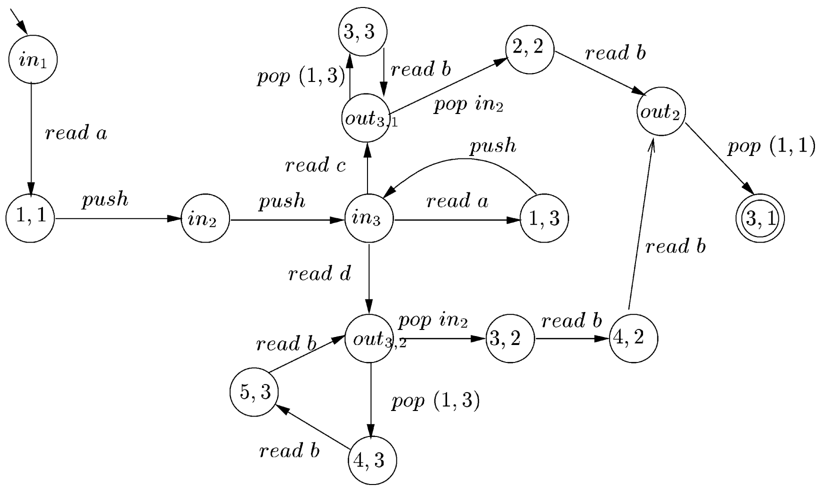

Fig. 3. A DPDA in atomic normal form equivalent to the DFA with recursive calls from Fig. 1.

In the rest of this section we prove that DFAs with recursive calls characterize exactly the deterministic context-free languages. We start by showing that any language accepted by a DFA with recursive calls can be accepted also by a deterministic pushdown automaton, then we prove the converse.

Lemma 1. *Given a DFA with recursive calls $D = \langle F_1 \Leftarrow D_1, \ldots, F_N \Leftarrow D_N \rangle$, the language $L(D)$ is accepted (by final state) by a pushdown automaton M, which can be constructed effectively from D.*

Proof. By Theorem 1 it is sufficient to construct a deterministic pushdown automaton M in atomic normal form that accepts $L(D)$. We define M as follows. The set of states of M is the union of the sets of states of each D_i for $i = 1, \ldots, N$. The tape alphabet, the initial state, and the accepting states of M are the same as for D. The stack symbols are the states from which a function symbol can be read in D. To complete the construction we need to give the read, push and pop moves. For any transition $\delta_i(p, \sigma) = q$, $i = 1, \ldots, N$, where $\sigma \in \Sigma$, we add a read move on σ from p to q in M. For any call to F_i from a state p, we add a push move from p to the entry state of D_i, and a pop move from any exit state to the corresponding state of D_i. It can be shown that the language accepted by M, by final state, is $L(D)$. $\square$

As an example of the above construction, we give in Fig. 3 the deterministic pushdown automaton in atomic normal form corresponding to the DFA with recursive calls shown in Fig. 1.

To prove the converse, that is, each deterministic context-free language is recognized by a DFA with recursive calls, we appeal to a result proved in [8]. Before stating this result we need to introduce some definitions.

A ranked alphabet is a set Δ together with a rank function $rank : \Delta \to \mathbb{N}$, where $\mathbb{N}$ is the set of natural numbers. Every symbol g of $rank(g) = n$ is said to have *arity* n. Let Δ be a ranked alphabet of function symbols, and $\Gamma = \{G_1, \ldots, G_N\}$ be a finite set of nonterminals. A *tree domain* DOM is a non empty set of strings over $\mathbb{N}_+$ (the set of positive integers) satisfying the following conditions:

- For each u in DOM, every prefix v of u is also in DOM.
- For each u in DOM, for every positive integer i, if ui is in DOM, then for every j, $1 \leqslant j \leqslant i$, uj is also in DOM.

Let $\{x_1, \ldots, x_m\}$ be a set of variables (considered as symbols of arity 0). A tree over $\Delta \cup \Gamma \cup \{x_1, \ldots, x_m\}$ is a mapping $t : \text{DOM} \to \Delta \cup \Gamma \cup \{x_1, \ldots, x_m\}$ such that the following conditions hold:

- DOM is a tree domain.
- For every u in DOM, if $n = |\{i \in \mathbb{N}_+ \mid ui \in \text{DOM}\}|$, then $rank(t(u)) = n$ (the arity of the symbol labeling u).

The set of all trees over $\Delta \cup \Gamma \cup \{x_1, \ldots, x_m\}$ is denoted by $T_{\Delta\cup\Gamma}(\{x_1, \ldots, x_m\})$. Given a tree t, its domain is denoted by $dom(t)$. The elements of the domain are called *nodes* or *tree addresses*, and a node u is a leaf if $|\{i \in \mathbb{N}_+ \mid ui \in \text{DOM}\}| = 0$. A tree is *finite*, if its domain is finite, and *infinite*, otherwise.

A *recursion scheme* $\alpha = \langle \alpha_1, \ldots, \alpha_N \rangle$ is a sequence of definitions $G_i \Leftarrow \alpha_i$, where α_i is a tree in the free algebra $T_{\Delta\cup\Gamma}(\{x_1, \ldots, x_{m_i}\})$, m_i being the arity of G_i. It is well known that a recursion scheme α can be unfolded into a N-tuple α^∇ of infinite trees in $T_\Delta(\{x_1, \ldots, x_m\})$ [6,8,14].

In [8] it has been shown that every deterministic context-free language is the set of tree-addresses in an infinite tree obtained by unfolding a (tree) recursion scheme. We recall this result in the following theorem.

Theorem 2. *Given a DPDA M with alphabet* $[1, N]$, *one can construct a recursion scheme α over a ranked alphabet $\Delta = \{\sharp, c\}$ where $rank(\sharp) = rank(c) = N$, such that the language $L(M)$ accepted by M is equal to the set of all tree addresses labeled with $\sharp$ in the infinite tree α_1^∇, the first component of the unfoldment of α.*

We can now conclude that every deterministic context-free language is accepted by a DFA with recursive calls. There is a simple way of translating a recursion scheme $\alpha = \langle \alpha_1, \ldots, \alpha_N \rangle$ into a DFA with calls. All we have to do is to disregard the labels $\sharp$ and c and modify the trees α_i in the following way:

(1) For every part of the tree of the form shown in Fig. 4(a), where G_i labels a node, change it to the graph shown in Fig. 4(b), where F_i labels an edge.
(2) Change every variable x to an exit state of index i.
(3) For every node, the ith outgoing edge is labeled with input a_i.

Hence we prove the following result.

Theorem 3. *Every deterministic context-free language is accepted by a DFA with recursive calls.*

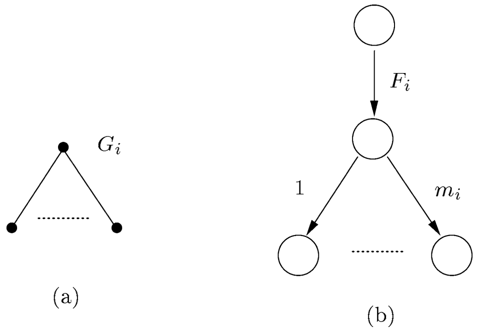

Fig. 4. Translation of a recursion scheme to a DFA with recursive calls.

4. Discussion

In this paper we have shown that DFAs with recursive calls accept exactly the deterministic context-free languages. Together with the celebrated result by Sénizergues [15] on the DPDA equivalence problem, this gives a decidability result also for the equivalence problem for DFAs with recursive calls.

The unfolding of the recursion in a DFA with recursive calls yields an infinite graph. It is possible to show that this infinite graph is the colimit of a sequence of finite approximating DFAs. We can also prove that any infinite graph obtained from a DFA with recursive calls is representable by a simple equational graph [7]. There is an ongoing research aimed at characterizing the Chomsky hierarchy by classes of infinite graphs (see, for example, [5]).

An interesting feature of our formalism is the intimate connection with DFAs. For several problems, it is possible to obtain algorithms by adapting algorithms given for the corresponding problems on DFAs. This is true, for example, for the emptiness problem, where an efficient algorithm can be obtained by combining the algorithm for checking the emptiness on DFAs with a depth-first search of the DFA with recursive calls.

If we focus on DFAs with recursive calls sharing the same "recursion structure" this connection is also more evident. Given a DFA with recursive calls D and denoting by $D_1, \ldots, D_n$ the sequence of its component DFAs, we define the *call graph* of D as a di-graph where each vertex v_i corresponds to D_i, each v_i is labeled with the number of exits of D_i and there exists an edge from v_i to v_j if and only if there exists a call to D_j from D_i. By applying the

DFA minimization algorithm locally to each DFA D_i, we can construct a minimal DFA with recursive calls which has the same call graph as D and such that the language accepted by D is preserved.

Finally, we remark that our model is similar to a paradigm that has been recently introduced in the context of verification [2,3]. Analogies with these papers end at this point, in fact we depart from them on both the results we obtain and the techniques we use. In [2], the authors solve the reachability and cycle detection problems on recursive finite state machines by reducing them to evaluating datalog programs. Benedikt et al. [3] consider model checking with respect to temporal logic specifications, and also show that, for every recursive finite state machine M, one can construct in linear time a pushdown automaton A such that the Kripke structures generated by M and A are bisimilar. From this result, we can only infer that the languages accepted by recursive finite state machines are context free. In both papers, the focus is on verification problems and thus determinism is not addressed. Here we have characterized the class of deterministic context-free languages, and for this purpose we have placed suitable restrictions on the model.

References

[1] R. Alur, M. Yannakakis, Model checking of hierarchical state machines, in: Proc. of the 6th ACM Symposium on the Foundations of Software Engineering, 1998, pp. 175–188.

[2] R. Alur, K. Etessami, M. Yannakakis, Analysis of recursive state machines, in: Proc. of the 13th International Conference on Computer-Aided Verification, CAV'01, in: Lecture Notes in Comput. Sci., Vol. 2102, Springer, Berlin, 2001, pp. 207–220.

[3] M. Benedikt, P. Godefroid, T.W. Reps, Model checking of unrestricted hierarchical state machines, in: Proc. of the 28th International Colloquium Automata, Languages and Programming, ICALP'01, in: Lecture Notes in Comput. Sci., Vol. 2076, Springer, Berlin, 2001, pp. 652–666.

[4] G. Booch, I. Jacobson, J. Rumbaugh, Unified Modeling Language User Guide, Addison-Wesley, Reading, MA, 1997.

[5] D. Caucal, On the transition graphs of Turing machines, in: Proc. of the 3rd International Conference on Machines, Computations and Universality, MCU'01, in: Lecture Notes in Comput. Sci., Vol. 2055, Springer, Berlin, 2001, pp. 177–189.

[6] B. Courcelle, M. Nivat, Algebraic families of interpretations, in: Proc. of the 17th IEEE Symposium on Foundations of Computer Science, FOCS'76, 1976, pp. 137–146.

[7] B. Courcelle, The monadic second-order logic of graphs: II. Infinite graphs of bounded width, Math. System Theory 21 (1989) 187–221.

[8] J.H. Gallier, Dpda's in "atomic normal form" and applications to equivalence problems, Theoret. Comput. Sci. 14 (2) (1981) 155–186.

[9] D. Harel, Statecharts: a visual formalism for complex systems, Sci. Comput. Programm. 8 (1987) 231–274.

[10] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading, MA, 1979.

[11] M.A. Harrison, I.M. Havel, Strict deterministic grammars, J. Comput. System Sci. 7 (1973) 237–277.

[12] W. Kuich, Systems of pushdown acceptors and context-free grammars, EIK 6 (1970) 95–114.

[13] D. Lehmann, Categories for fixpoint semantic, in: Proc. of the 17th IEEE Symposium on Foundations of Computer Science, FOCS'76, 1976, pp. 122–126.

[14] M. Nivat, On the interpretation of recursive polyadic program schemes, Sympos. Math. 15 (1975) 255–281.

[15] G. Sénizergues, The equivalence problem for deterministic pushdown automata is decidable, in: Proc. of the 24th International Colloquium Automata, Languages and Programming, ICALP'97, in: Lecture Notes in Comput. Sci., Vol. 1256, Springer, Berlin, 1997, pp. 671–681.

[16] G. Sénizergues, Some applications of the decidability of DPDA's equivalence, in: Proc. of the 3rd International Conference on Machines, Computations and Universality, MCU'01, in: Lecture Notes in Comput. Sci., Vol. 2055, Springer, Berlin, 2001, pp. 114–132.

[17] C. Stirling, Decidability of DPDA equivalence, Theoret. Comput. Sci. 255 (1–2) (2001) 1–31.

[18] W.A. Woods, Transition network grammars for natural language analysis, Comm. ACM 13 (10) (1970) 591–606.