Math. Systems Theory **29**, 227–244 (1996)

Mathematical Systems Theory
© 1996 Springer-Verlag New York Inc.

Tree-Stack Automata

W. Golubski and W. M. Lippe

Fachbereich 15, Institut für Numerische und Instrumentelle Mathematik/Informatik, Wilhelms-Universität Münster, Einsteinstrasse 62, D-48149 Münster, Germany

Abstract. We introduce a new model of stack automata, the "tree-stack automata," extending the linear stack to a tree-stack. A main subject of our investigations is to explore the relationship between tree-stack automata and stack automata. The main result of this paper is that tree-stack have the same recognition power as stack-pushdown automata, another (well-known) extension of stack automata. Therefore we obtain that the class of languages accepted by the one-way (linear) stack automata is a proper subset of the class of languages accepted by the one-way tree-stack automata and that two-way tree-stack automata have the same recognition power as two-way (linear) stack automata. As a special case of tree-stack automata we consider tree-pushdown automata. As opposed to stack automata the tree-pushdown storage does not extend the recognition power of one-way (resp. two-way) pushdown automata.

1. Introduction

The theoretical model of stack automata and pushdown automata is traditionally motivated by compiler theory, and the implementation of recursive procedures with parameters. Stack languages and pushdown languages have been studied extensively in relation to complexity and generating devices [11]–[13], [15], [16].

Our work was motivated by formal investigations about decision problems of programs with procedures [17]–[19]. Such a program can be regarded as a tree manipulating system. The decision problems could be answered by an appropriate linearization of the tree structure which led to systems solving the decision problems. Trees generated by these programs are very special. We generalize the question of linearization to more powerful systems, i.e., we investigate the possibility of transforming systems working on trees into systems working on strings (linear trees) without loosing specific properties (e.g., halting problem).

It is shown in [11] how to extend regular canonical systems [4] to stack systems. For a generalization of these systems—the tree-stack systems are not more powerful than the regular canonical systems—see [20] and [21]. Tree-stack systems are a special type of stack automata—stack automata without an input tape.

In this paper we generalize tree-stack systems to "tree-stack automata." This new model of stack automata extends the linear stack to a tree-stack. In the writing-mode of the automaton a new tree can be pushed on the stack employing the subtrees of the old tree-stack, i.e., subtrees can be permuted, deleted, or copied. If the root of the tree-stack is popped, exactly one subtree is left in the store. In the reading-mode the automaton walks down to a son or walks up to the father of the current node of the tree-stack.

Pushdown automata can be considered as a special type of stack automata, to be more precise as stack automata without a reading-mode. Hence similarly we obtain tree-pushdown automata from tree-stack automata. In [14] Guessarian investigated pushdown tree automata where the input tape is organized as a tree and the storage is a tree-pushdown. Tree-stack automata can also be considered as an extension of tree-pushdown automata with an ordinary input tape.

The main subject of our investigation is to explore the relationship between tree-stack automata and stack automata (resp. the relationship between tree-stack languages and certain language-generating systems).

For the one-way version the recognition power of tree-stack automata exceeds the recognition power of (linear) stack automata. Our main result is a constructive proof for the equivalence of one-way tree-stack automata and one-way stack-pushdown automata [7]. Hence a grammatical characterization of our new stack automata can be obtained from the equivalence between one-way stack-pushdown automata and Extended Basic grammars. One-way tree-stack automata (resp. restricted versions (nonerasing, ...)) can thus be used for implementing languages of Extended Basic grammars (ETOL-systems, ...). For example, one-way checking tree-stack automata are nice and simple models of ETOL-systems. Closure properties for the class of accepted languages of one-way tree-stack automata and the relationship between various types of one-way tree stack automata also follow immediately from the equivalence given above.

The pushdown of the stack-pushdown is necessary only for simulating the reading-mode of tree-stack automata. Hence the technique used to prove the equivalence of the writing-mode can be applied to show that one-way tree-pushdown automata and one-way pushdown automata have the same recognition power.

Automata with tree-pushdown storage management are well known from [14]. There it is shown that pushdown tree automata can be simulated by restricted pushdown tree automata for which the storage is an ordinary pushdown. The proof is based on grammatical arguments unlike our automata-theoretical argumentation. Our construction presented in this paper can be directly applied to Guessarian's pushdown tree automata.

For the two-way automata we obtain that two-way tree-stack automata have the same recognition power as two-way stack-pushdown automata and hence as two-way nested stack automata and two-way stack automata. The result for two-way tree-pushdown automata is similar. Two-way tree-pushdown automata have the same recognition power as two-way pushdown automata.

The paper proceeds as follows. The text is divided into four sections. The second section contains some basic definitions. In Section 3 we introduce and investigate one-way tree-stack automata and in the last section we consider two-way tree-stack automata.

2. Definitions

First we recall some important definitions and notations for this paper.

Let N_0 be the set of nonnegative integers, let N be the set of positive integers, and let N^+ be the free monoid generated by N with the monoid-operation "$\cdot$" (concatenation) and unity 0. Let "$\prec$" be the relation on N^+ with $d \prec d' \Leftrightarrow \exists x \in N^+\colon d \cdot x = d'$.

A tree domain is a finite subset θ of N^+ such that, for every $d, d' \in N^+$ and $i, j \in N$:

(i) If $d \in \theta$ and $d' \prec d$, then $d' \in \theta$.
(ii) If $d \cdot j \in \theta$ and $i < j$, then $d \cdot i \in \theta$.

The elements of a tree domain θ are called nodes. The node 0 is called the root of θ and the set of all leaves of θ is defined by $F_\theta = \{d \in \theta \mid \text{for every } d' \in \theta\text{: if } d \prec d', \text{then } d' = d\}$.

A ranked alphabet is a pair $\langle A, \rho\rangle$, where A is a finite set of symbols and ρ is a mapping $\rho\colon A \to N_0$, called rank. For every $n \in N_0$ we denote $\rho^{-1}(n)$ by A^n. A tree t over $\langle A, \rho\rangle$ is a function $t\colon \theta \to A$, where θ is a tree domain and $\rho(t(d)) = \max(\{i \in N \mid d \cdot i \in \theta\})$ for every $d \in \theta$ where $\max(\emptyset) = 0$. The tree domain of a given tree t is denoted by $\theta(t)$. The set of all trees over $\langle A, \rho\rangle$ is denoted by $T_{\langle A,\rho\rangle}$. For $t \in T_{\langle A,\rho\rangle}$ and $i \in N$ we define the (direct) subtree s_t^i of t by $s_t^i = t'$ with $t'(0) = t(i)$ and $t'(d) = t(i \cdot d)$, for every $i \cdot d \in \theta(t)$.

Let $\langle A, \rho\rangle$ be a ranked alphabet and let X be a finite set of variables. A tree t over $\langle A, \rho\rangle$ and X is a tree over the ranked alphabet $\langle A \cup X, \rho'\rangle$ where $\rho'(a) = \rho(a)$ for every $a \in A$ and $\rho'(x) = 0$ for every $x \in X$. Then we define by $T_{\langle A,\rho\rangle}(X)$ the set of all trees over $\langle A, \rho\rangle$ and X. By this definition a variable can only appear at a leaf of a tree. Note that $T_{\langle A,\rho\rangle}(X) = T_{\langle A,\rho\rangle}$ if $X = \emptyset$.

Another notation of trees is the representation by strings. A tree over $\langle A, \rho\rangle$ is either a symbol in A of rank 0 or a string of the form $a\langle t_1, \ldots, t_{\rho(a)}\rangle$, where $a \in A$, $\rho(a) \neq 0$, t_i is a tree over $\langle A, \rho\rangle$, for every $1 \leq i \leq \rho(a)$, and $\langle$, $\rangle$, and the comma are new symbols. As usual, *length*(w) denotes the length of the string w.

3. One-Way Tree-Stack Automata

In this section we consider only automata scanning their input tape in one direction. Since the pushdown automaton is a special case of the stack automaton, we define at first one-way tree-stack automata and then as a certain restricted form of this automaton model the tree-pushdown automata. Hereby we generalize the linear store of the well-known one-way stack automaton, see [11] and [12], to a tree-stack.

Informally a so-called one-way tree-stack automaton consists of an input tape, a tree-stack, and a control unit with two pointers: one to a square of the input tape (input pointer) and one to a node of the tree-stack (stack-pointer) (Figure 1). The one-way tree-stack automaton is in writing mode if it points to the root of the tree-stack and in

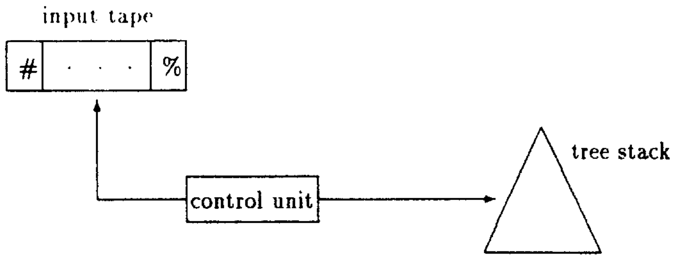

Fig. 1. A one-way tree-stack automaton.

reading-mode otherwise. In the writing-mode four operations (push, pop, down moves, and stay) are allowed. First, the automaton can push a new tree with variables on the top of its stack. If a tree t, where t is not a single variable, is pushed on the tree-stack $a\langle t_1, \ldots, t_{\rho(a)}\rangle$, then the tree-stack changes into t', where t' is the result of substituting t_i for x_i in t (where $t \in T_{\langle A,\rho\rangle}(\{x_1, \ldots, x_{\rho(a)}\})$). This means that some subtrees can be duplicated and others can be deleted in one step. A push move need not always really push something. It may also consist, e.g., of a change of the root symbol. Second, the automaton can pop the root of the tree-stack in such a way that exactly one subtree is left in the new tree-stack. If a pop to the ith subtree is made on the tree-stack $a\langle t_1, \ldots, t_{\rho(a)}\rangle$, then the tree-stack changes into t_i. In both cases the finite control moves to the root of t' (resp. t_i). Third, the automaton can walk down on the tree-stack to the ith son of the root. Hereby the root symbol cannot be changed and the automaton enters in reading-mode. In this case the finite control moves down to the ith son of the root of the tree-stack. Fourth, the automaton can stay on the tree-stack. A stay operation does not change the contents of the tree-stack and the finite control stays at the root of the tree-stack. In the reading-mode the automaton can walk on the tree-stack in both directions or can stay on the tree-stack. If a down move to the ith son or an up move to the father of the current node is made, then the finite control (the stack pointer) moves to this node. A stay operation does not change the contents of the tree-stack and the finite control stays at the current node of the tree-stack.

One-way tree-stack automata which are only in writing-mode, are called tree-pushdown automata.

The next definition formalizes the concept above.

Definition 3.1. A one-way tree-stack automaton (abbreviated 1tsa) is a 7-tuple $M = (\langle A, \rho\rangle, B, Q, F, q_0, a_0, \Pi)$ where $\langle A, \rho\rangle$ is a ranked alphabet, called the stack alphabet, B is a finite set of input symbols, Q is a finite set of states, $F \subseteq Q$ is a set of final states, q_0 is a designated element of Q, called the initial state, a_0 is a designated symbol in $\langle A, \rho\rangle$ of rank 0, called the initial stack symbol, and Π is a finite set of rules of the following forms:

1. $(b, q, a) \to (r, q', push(t))$,
2. $(b, q, a) \to (r, q', pop(i))$,
3. $(b, q, a) \to (r, q', down(i))$,

4. $(b, q, a) \to (r, q', up)$,
5. $(b, q, a) \to (r, q', stay)$,

where $q, q' \in Q, a \in A, b \in B \cup \{\#, \%\}, \#, \% \notin B$ (delimiter symbols of the input tape), $t \in T_{\langle A,\rho\rangle}(X)$, and $t \notin X$ where $X = \{x_1, \ldots, x_{\rho(a)}\}$, $1 \leq i \leq \rho(a)$, and $r \in \{0, 1\}$ except if $b = \%$, then $r = 0$.

Rules of form 1 mean that if the automaton is in writing-mode, is in state q, reads b on its input tape and a on its tree-stack, then it stays in writing-mode, it moves its input pointer r squares forward, goes into state q', and pushes the tree t.

Rules of form 2 mean that if the automaton is in writing-mode, is in state q, reads b on its input tape and a on its tree-stack, then it stays in writing-mode, it moves its input pointer r squares forward, goes into state q', and pops the top symbol of the tree-stack so that the tree-stack now consists of the ith subtree of a.

Rules of form 3 mean that if the automaton is in writing-mode (resp. reading-mode), is in state q, reads b on its input tape and a on its tree-stack, then it changes to reading-mode (resp. stays in reading-mode), it moves its input pointer r squares forward, goes into state q', and moves its stack pointer to the ith son of a.

Rules of form 4 mean that if the automaton is in reading-mode, is in state q, reads b on its input tape and a on its tree-stack, then it stays in reading-mode, it moves its input pointer r squares forward, goes into state q', and moves its stack pointer to the father of a. If now the top symbol (i.e., the root) of the tree-stack is reached, the automaton changes to writing-mode.

Rules of form 5 mean that if the automaton is in writing-mode (resp. reading-mode), is in state q, reads b on its input tape and a on its tree-stack, then it stays in writing-mode (resp. reading-mode), it moves its input pointer r squares forward, goes into state q', and leaves its stack pointer at the same node.

Definition 3.2. Let $M = (\langle A, \rho\rangle, B, Q, F, q_0, a_0, \Pi)$ be a 1tsa. A configuration of M is a sequence $(\#w\%, l, q, t, d)$ with $w \in B^*$ (w is the contents of the input tape), $0 \leq l \leq length(w) + 1$ (l shows the position of the input pointer on the input tape), $q \in Q, t \in T_{\langle A,\rho\rangle}$ (t is the contents of the tree-stack), and $d \in \theta(t)$ (d shows the position of the stack pointer on the tree-stack). The "move-relation" $\Rightarrow_M$ between configurations is defined as follows. Let $q, q' \in Q, b \in B \cup \{\#, \%\}, a \in A, r \in \{0, 1\}$ and let d be a node of t labeled with a. Let $C_1 = (\#w\%, l, q, t, d)$, let b be the lth symbol of $\#w\%$, and let $(b, q, a) \to (r, q', e)$ be an element of Π. Then $C_1 \Rightarrow_M C_2$ such that:

If $e = push(s)$ with $s \in T_{\langle A,\rho\rangle}(X)$ and $s \notin X$ where $X = \{x_1, \ldots, x_{\rho(a)}\}$, and $d = 0$ (i.e., d is the root of t), then $C_2 = (\#w\%, l + r, q', t', d')$ where $t' \in T_{\langle A,\rho\rangle}$ is the result of substituting s_t^i for x_i in s, $1 \leq i \leq \rho(a)$, and $d' = 0$ (i.e., d' is the root of t').

If $e = pop(i)$, $1 \leq i \leq \rho(a)$, and $d = 0$, then $C_2 = (\#w\%, l + r, q', s_t^i, d')$ where $d' = 0$ (i.e., d' is the root of s_t^i).

If $e = down(i)$, $1 \leq i \leq \rho(a)$, and $d \cdot i \in \theta(t)$, then $C_2 = (\#w\%, l + r, q', t, d')$ where $d' = d \cdot i$.

If $e = up$ and $d \neq 0$, then $C_2 = (\#w\%, l + r, q', t, d')$ where d' is the unique node of t such that $d = d' \cdot j$ for some $j \in N$.
If $e = stay$, then $C_2 = (\#w\%, l + r, q', t, d)$.

As usual, $\overset{*}{\Rightarrow}_M$ is used to denote the transitive closure of $\Rightarrow_M$ (i.e., sequences of $\Rightarrow_M$ moves). The language accepted by M, denoted by $L(M)$, is

$$L(M) = \{w \mid (\#w\%, 0, q_0, a_0, 0) \overset{*}{\Rightarrow}_M (\#w\%, k+1, q_f, t, d), \text{ for } w \in B^*, \\ length(w) = k, q_f \in F, t \in T_{\langle A,\rho\rangle}, d \in \theta(t)\}.$$

The class of languages accepted by one-way tree-stack automata is denoted by $\mathcal{L}(1TSA)$.

Note that M starts its computation in writing-mode, i.e., with the stack pointer at the root of the tree-stack. In a final configuration the stack pointer can be at any node of the tree-stack. Sometimes one way tree-stack automata are also called one-way nondeterministic tree-stack automata. Next we define several restricted types of these automata.

Definition 3.3. Let M be a 1tsa. M is a (linear, usual) one-way stack automaton (abbreviated 1sa) if, for all $a \in A$, $\rho(a) \in \{0, 1\}$; M is deterministic (abbreviated d) if for every left side of a rule in Π exactly one right side exists; M is nonerasing (abbreviated ne) if no "pop rule" in Π exists; M is checking (abbreviated c) if it is nonerasing and it is at first only in writing-mode and then exclusively in reading-mode; M is a tree-pushdown automaton (abbreviated 1tpda) if M has only push and pop rules.

Note that stack automata can still have push rules changing the whole tree-stack into a tree-stack consisting of only one symbol of the stack alphabet. This means that our definition of a linear stack automaton differs from the classical definition (e.g., see [11]) of a linear stack automaton. It is not difficult to see that our linear stack automaton model is equivalent to the classical model because such rules can be omitted.

In what follows we often use notation like $\mathcal{L}(x1TSA)$, $x \in \{nd, d, ne, c\}$, which has to be understood as an abbreviation of the class of languages accepted by one-way nondeterministic tree-stack automata, the class of languages accepted by one-way deterministic tree-stack automata, the class of languages accepted by one-way nonerasing tree-stack automata, and the class of languages accepted by one-way checking tree-stack automata. Similar notations are used for combinations of nd, d, ne, and c; e.g., $\mathcal{L}(dc1TSA)$ denotes the class of languages accepted by one-way deterministic checking tree-stack automata.

If no additional specification appears, then we deal with nondeterministic automata. Therefore $\mathcal{L}(1TSA) = \mathcal{L}(nd1TSA)$.

Example. Let $L = \{a^n b^{n^2} c^n \mid n \geq 0\}$. The following dc1tsa accepts L. Let $M = (\langle A, \rho\rangle, B, Q, F, q_0, a_0, \Pi)$ with $A = \{0, 1, \$, 2, 2'\}$ and $\rho(0) = 0$, $\rho(1) = \rho(\$) = 1$, $\rho(2) = \rho(2') = 2$, $B = \{a, b, c\}$, $Q = \{q_0, \ldots, q_{14}, f\}$, $F = \{f\}$, $a_0 = 0$, and Π is defined as follows:

1. $(\#, q_0, a_0) \rightarrow (1, q_1, push(a_0))$.

2. $(\%, q_1, a_0) \to (0, f, push(a_0))$.

Rule 2 is used for the case $n = 0$.

3. $(a, q_1, a_0) \to (1, q_2, push(a_0))$.
4. $(b, q_2, a_0) \to (1, q_3, push(a_0))$.
5. $(c, q_3, a_0) \to (1, q_4, push(a_0))$.
6. $(\%, q_4, a_0) \to (0, f, push(a_0))$.

Rules 4–6 are used for the case $n = 1$.

7. $(a, q_2, a_0) \to (1, q_5, push(2'\langle 0, 0\rangle))$.
8. $(a, q_5, 2') \to (1, q_5, push(2'\langle 2\langle x_1, x_2\rangle, 1\langle x_2\rangle\rangle))$.

By these rules M creates a tree-stack of depth n of the form shown in Figure 2 and accepts on its input exactly a^n for $n \geq 2$. The rule

9. $(b, q_5, 2') \to (0, q_6, push(\$\langle 2'\langle x_1, x_2\rangle\rangle))$

stops the tree-stack growing.

10. $(b, q_6, \$) \to (0, q_7, down(1))$.
11. $(b, q_7, 2') \to (0, q_7, down(1))$.
12. $(b, q_7, 2) \to (0, q_7, down(1))$.

Now M walks down along the left edges to the leftmost leaf by rules 11 and 12.

13. $(b, q_7, 0) \to (1, q_8, up)$.
14. $(b, q_8, 2) \to (1, q_8, down(2))$.
15. $(b, q_8, 1) \to (1, q_8, down(1))$.
16. $(b, q_8, 0) \to (1, q_9, up)$.
17. $(b, q_9, 1) \to (0, q_9, up)$.
18. $(b, q_9, 2) \to (0, q_8, up)$.
19. $(b, q_8, 2') \to (1, q_{10}, down(2))$.
20. $(b, q_{10}, 1) \to (1, q_{10}, down(1))$.
21. $(b, q_{10}, 0) \to (1, q_{11}, up)$.
22. $(b, q_{11}, 1) \to (0, q_{11}, up)$.

By these rules M counts each node one time. In each counting step one input symbol b is accepted. So M has accepted at this moment $a^n b^m$ with $m = \sum_{i=1}^{n} i$.

The next rules repeat this procedure but the right edge will not be counted.

23. $(b, q_{11}, 2') \to (0, q_{11}, down(1))$.
24. $(b, q_{11}, 2) \to (0, q_{11}, down(1))$.
25. $(b, q_{11}, 0) \to (1, q_{12}, up)$.
26. $(b, q_{12}, 2) \to (1, q_{12}, down(2))$.
27. $(b, q_{12}, 1) \to (1, q_{12}, down(1))$.
28. $(b, q_{12}, 0) \to (1, q_{13}, up)$.
29. $(b, q_{13}, 1) \to (0, q_{13}, up)$.
30. $(b, q_{13}, 2) \to (0, q_{12}, up)$.

At this moment M has accepted $a^n b^m b^{m-n}$ with $m = \sum_{i=1}^{n} i$. Since $2(\sum_{i=1}^{n} i) - n = n^2$

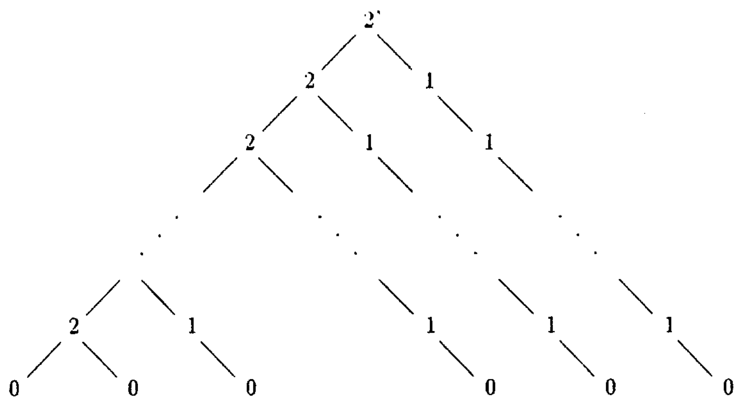

Fig. 2. A tree-stack.

we obtain the acceptance of $a^n b^{n^2}$. Finally rules are necessary for the acceptance of c^n. This is done by walking down the left edges.

31. $(c, q_{12}, 2') \rightarrow (1, q_{14}, down(1))$.
32. $(c, q_{14}, 2) \rightarrow (1, q_{14}, down(1))$.
33. $(c, q_{14}, 0) \rightarrow (1, f, up)$.

Thus the construction is complete.

The main question of our investigations is to relate the recognition power of tree-stack automata and their linear version. In other words, can a 1tsa be simulated by a 1sa and can a 1tpda be simulated by a 1pda? Immediately we can answer this question for one-way tree-stack automata. Ogden has proved in [22] that $L = \{a^n b^{n^2} c^n \mid n \geq 0\} \notin \mathcal{L}(1SA)$. Hence we obtain the following lemma.

Lemma 3.4. *$\mathcal{L}(x1SA) \subset \mathcal{L}(x1TSA)$ for $x \in \{nd, d, ndne, dne, ndc, dc\}$.*

By this result the question of how to characterize the class of languages accepted by a 1tsa arises. We have found another automaton model to simulate a 1tsa. An appropriate model turns out to be the one-way stack-pushdown automaton (1spda) investigated in [7]. We give a short definition. This automaton is obtained from an ordinary 1sa by adding a pushdown store during the reading-mode. The pushdown is upside down (Figure 3), the bottom of the pushdown is at the same level as the stack square directly below the top, and the top of the pushdown is at the same level as the stack pointer. In writing-mode the pushdown is empty.

During the writing-mode the stack-pushdown automaton works like an usual 1sa. With the beginning of the reading-mode, the auxiliary pushdown is initialized. The automaton pushes an auxiliary symbol on the pushdown, when it moves down one step in the stack. During an up move in the stack, the symbol at the top of the pushdown must be popped. In writing-mode and reading-mode the automaton can stay on its storage and does not change the contents of the storage.

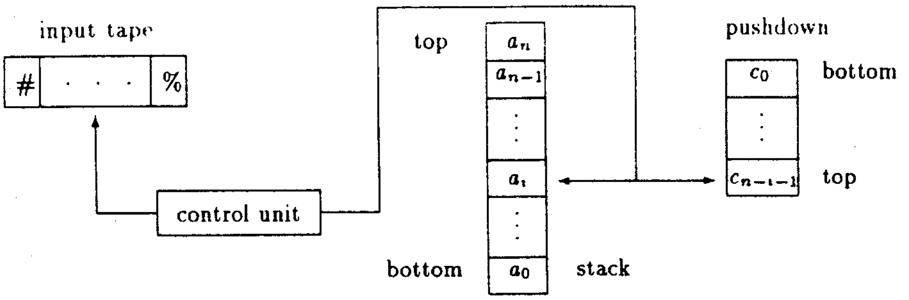

Fig. 3. A one-way stack-pushdown automaton (1spda).

Definition 3.5. A 1spda is an 8-tuple $N = (A, C, B, Q, F, q_0, a_0, \Pi)$ where A is a finite set of stack symbols, C is a finite set of pushdown symbols, B is a finite set of input symbols, Q is a finite set of states, $F \subseteq Q$ is a set of final states, q_0 is a designated symbol in Q, called the initial state, a_0 is a designated symbol in A, called the initial stack symbol, and Π is a finite set of rules of the form:

1. $(b, q, a) \to (r, q', push(v))$,
2. $(b, q, a) \to (r, q', pop)$,
3. $(b, q, a) \to (r, q', down, c')$
4. $(b, q, a, c) \to (r, q', down, c')$,
5. $(b, q, a, c) \to (r, q', up)$,
6. $(b, q, a) \to (r, q', stay)$,
7. $(b, q, a, c) \to (r, q', stay)$,

where $q, q' \in Q, a \in A, c, c' \in C, b \in B \cup \{\#, \%\}, \#, \% \notin B$ (delimiter symbols of the input tape), $v \in A^+$, and $r \in \{0, 1\}$ except if $b = \%$, then $r = 0$.

Rules of form 1 mean that if the automaton is in writing-mode, is in state q, reads b on its input tape and a on its stack, then it stays in writing-mode, it moves its input pointer r squares forward, goes into state q', and replaces the top symbol a of the stack by the string v.

Rules of form 2 mean that if the automaton is in writing-mode, is in state q, reads b on its input tape and a on its stack, then it stays in writing-mode, it moves its input pointer r squares forward, goes into state q', and pops the top symbol off the stack.

Rules of form 3 mean that if the automaton is in writing-mode, is in state q, reads b on its input tape and a on its stack, then it changes to reading-mode, it moves its input pointer r squares forward, goes into state q', moves its stack pointer one square down, and initializes the auxiliary pushdown by c'.

Rules of form 4 mean that if the automaton is in reading-mode, is in state q, reads b on its input tape, a on its stack and c on its pushdown, then it stays in reading-mode, it moves its input pointer r squares forward, goes into state q', moves its stack pointer one square down, and pushes one symbol c' on the auxiliary pushdown.

Rules of form 5 mean that if the automaton is in reading-mode, is in state q, reads b on its input tape, a on its stack and c on its pushdown, then it stays in reading-mode, it moves its input pointer r squares forward, goes into state q', moves its stack pointer one square up, and pops the top symbol off the auxiliary pushdown. If now the top symbol of the stack is reached, the automaton changes to writing-mode.

Rules of form 6 (resp. 7) mean that if the automaton is in writing-mode (resp. reading-mode), is in state q, reads b on its input tape, a on its stack and in reading-mode c on its pushdown, then it stays in writing-mode (resp. reading-mode), it moves its input pointer r squares forward, goes into state q', and leaves its stack pointer at the same stack-pushdown square. The storage of the automaton remains unchanged.

The straightforward task of giving the analogues of Definitions 3.2 and 3.3 for a 1spda is left to the reader. Let $N = (A, C, B, Q, F, q_0, a_0, \Pi)$ be a 1spda. A configuration of N is a sequence $(\#w\%, l, q, v, d)$ with $w \in B^*$ (w is the contents of the input tape), $0 \leq l \leq \mathit{length}(w) + 1$ (l shows the position of the input pointer on the input tape), $q \in Q$, $v \in A(A \times C)^*A^*$ (v is the contents of the stack-pushdown where if N is in reading-mode the stack squares with their associated pushdown squares are stated as a tuple sequence in $(A \times C)^+$), $1 \leq d \leq \mathit{len}(v)$ (d shows the position of the stack pointer on the stack-pushdown), and len is defined by $\mathit{len}(\varepsilon) = 0$, $\mathit{len}(v_1 v') = 1 + \mathit{len}(v')$, for $v_1 \in A \cup (A \times C)$ and $v' \in (A \times C)^*A^*$. For example, let $C_N = (\#w\%, l, q', a_1\{a_2, c_1\}\{a_3, c_2\}a_4, 3)$ be a configuration of a 1spda N. Then the stack pointer refers to the stack square with contents a_3 and the pushdown square with contents c_2. A stack-pushdown tuple is denoted by set braces. The language accepted by N is given by

$$L(N) = \{w \mid (\#w\%, 0, q_0, a_0, 1) \overset{*}{\Rightarrow}_N (\#w\%, k+1, q_f, v, d), \text{ for } w \in B^*, \mathit{length}(w) = k, q_f \in F, v \in A(A \times C)^*A^*, 1 \leq d \leq \mathit{len}(v)\}.$$

The class of languages accepted by one-way stack-pushdown automata is denoted by $\mathcal{L}(1SPDA)$.

Now we want to prove that one-way tree-stack automata and one-way stack-pushdown automata have the same power. First we show that every 1tsa can be simulated by a 1spda. Here the tree-stack has to be simulated by a linear stack (with a pushdown). Now we describe an approach of linearizing the tree-stack.

The tree-stack will be linearized as a sequence of nodes, with pointers to implement the tree structure in the usual way. The pointers are important for two reasons. The first reason is that they allow the 1spda to walk on the encoded tree (in reading-mode). The second reason is that they allow the sharing of certain subtrees, which is necessary because the 1tsa may copy a subtree of the tree-stack whereas the 1spda cannot of course copy parts of its stack, and instead simulates copying by generating pointers to the root of the encoded subtree.

We discuss how a push rule and a down rule are simulated, because this gives more insight into our following construction. Assume that the rule $(b, q, a) \to (r, q', \mathit{push}(t))$ is applied to a tree-stack $s = a\langle s_1, \ldots, s_n\rangle$, where $n = \rho(a)$ and so $t \in T_{\langle A,\rho\rangle}(\{x_1, \ldots, x_n\})$; the linear stack now contains an encoding of s, such that the top square contains an encoding of its root. This is simulated by a push rule of the 1spda that replaces that top square by a string that encodes t. This string is obtained by ordering the nodes of t in lexicographic order, i.e., in the usual preorder. Each node of t that is not labeled by a variable x_i is encoded by a sequence $[a', p_1, \ldots, p_k]$ where a' is its label and p_i is a pointer to (the encoding of) its ith son. Each node of t that is labeled by x_i is encoded by a pointer p to (the encoding of) the root of s_i; note that p can be found in the top square of the linear stack. A pointer is implemented as a natural number that indicates the distance to the square pointed at; since the size of t is bounded, the set of these natural numbers is finite and consequently these pointers can be incorporated in the stack alphabet of the 1spda.

A down rule $(b, q, a) \to (r, q', down(i))$ can be simulated in the following way. The 1spda goes down into its stack following the pointer chain that leads to the ith son of the current node. For every pointer in this chain on the way to the stack square to which the pointer refers, the 1spda pushes a "dummy" symbol on the pushdown and when the required square has been reached, the used pointer is pushed on the pushdown. The pushed pointer allows the 1spda to follow the pointer chain backward on the pushdown whereby an up move of the 1tsa can be simulated.

We introduce three new types of rules for a 1spda to simplify the following construction.

1. $(b, q, a) \to (r, q', pop_i)$ means that the first i elements from the top of the stack are erased.
2. $(b, q, a) \to (r, q', D_i)$ and $(b, q, a, c) \to (r, q', D_i)$ means that the 1spda moves down its stackpointer i squares and pushes in each step a dummy symbol $+$ in its auxiliary pushdown except the last step, where the symbol i is pushed.
3. $(b, q, a, c) \to (r, q', U_i)$ means that the 1spda moves up its stackpointer i squares and pops the i top symbols of the auxiliary pushdown.

These new types of rules can easily be simulated by ordinary rules (preserving the properties d, ne, and c).

Construction 1. For a given 1tsa $M = (\langle A, \rho\rangle, B, Q, F, q_0, a_0, \Pi)$ we construct a 1spda $N = (A', C, B, Q', F, q_0, [a_0], \Pi')$ that simulates M and hence accepts the same language.

We define A', C, Q' by

$$A' = \{[a, p_1, \ldots, p_{\rho(a)}] | a \in A, 1 = p_1 < \cdots < p_{\rho(a)} \le r_{\max}\} \cup \{i | 1 \le i \le N_{\max}\}.$$

$$C = \{i | 1 \le i \le N_{\max}\} \cup \{+\},$$

$$Q' = Q \cup \{q_d | q \in Q\} \cup \{q_u | q \in Q\}.$$

Here $r_{\max}$ denotes the maximal size of all trees t in all push rules of M (where the size of a tree is the number of its nodes), and $N_{\max} = 2 * r_{\max}$. In $[a, 1, p_2, \ldots, p_{\rho(a)}]$ the 1 is superfluous, but is kept to improve readability: the 1 always points to the next square, which encodes the root of the first subtree.

Construction of Π'

A. For every rule in Π of the form $(b, q, a) \to (r, q', push(t))$, there are rules in Π' constructed as follows:

(1) The tree t is arranged by lexicographic ordering of the nodes of the tree domain of t where $t(0) \in A$. (Note that t is not a variable.) In this way we obtain a string $z_0 \cdots z_n$ where $z_0 = t(0)$ and $z_i \in A \cup \{x_1, \dots, x_{\rho(a)}\}$, $1 \leq i \leq n$.

(2) We obtain the set of rules in Π' of the form $(b, q, [a, p_1, \dots, p_{\rho(a)}]) \to (r, q', push(z'_0 \dots z'_n))$ for all $1 = p_1 < p_2 < \cdots < p_{\rho(a)} \leq r_{\max}$. The string $z'_0 \cdots z'_n$ is obtained from $z_0 \cdots z_n$ as follows.

(3) $z_0 = t(0)$ is extended to $z'_0 = [t(0), r_1, \dots, r_k]$ where $k = \rho(t(0))$ and z_{r_j} corresponds to the root of the subtree s_t^j, for all $1 \leq j \leq k$. Note that $r_1 = 1$ (if $k \geq 1$). Note also that $r_j \leq r_{\max}$.

(4) $z_i = c$, $1 \leq i \leq n$, $c \in A$. Let d be the unique node in tree t corresponding to z_i, and let t' be the subtree of t at d, i.e., $t'(0) = t(d)$ and $t'(d') = t(d \cdot d')$ for every $d \cdot d' \in \theta(t)$. Then z_i is extended to $z'_i = [c, r_1, \dots, r_k]$ where $k = \rho(c)$ and $r_j = s_j - i$ with z_{s_j} corresponding to the root of the subtree $s_{t'}^j$. Note again that $r_1 = 1$ (if $k \geq 1$), and that $r_j \leq r_{\max}$.

(5) $z_i = x_j$, $1 \leq i \leq n$, $j \in \{1, \dots, \rho(a)\}$. Then z_i is replaced by the pointer $z'_i = n - i + p_j$. Note that $z'_i \leq N_{\max}$ because $n - i + p_j \leq n + p_j \leq r_{\max} + r_{\max}$. A(2) and A(5) cause the presence of pointers in the right-hand side of the rules. In the following example we have the pointers 1, 2, 4, 6.

Example: Let $(b, q, a) \to (r, q', push(a'\langle b_1\langle x_2, x_1\rangle, b_2\langle b_3, x_2\rangle\rangle))$ be a rule in Π and $\rho(a) = 2$. Then we obtain in Π' the rules $(b, q, [a, 1, p]) \to (r, q', push(z'_0 \cdots z'_6))$ for all $p \in \{2, \dots, r_{\max}\}$ with $z'_0 = [a', 1, 4]$; $z'_1 = [b_1, 1, 2]$; $z'_2 = 4+p$; $z'_3 = 4$; $z'_4 = [b_2, 1, 2]$; $z'_5 = [b_3]$; $z'_6 = p$. Let $C_M = (\#w\%, l, q, a\langle c_1, c_2\rangle, 0)$ be a configuration of a 1tsa, b is the lth symbol of the input w, and let a corresponding configuration of a simulating 1spda be $C_N = (\#w\%, l, q, [a, 1, 2][c_1][c_2], 1)$. Then we obtain the following configurations by applications of the respective rules:

$$C'_M = (\#w\%, l + r, q', a'\langle b_1\langle c_2, c_1\rangle, b_2\langle b_3, c_2\rangle\rangle, 0)$$

and

$$C'_N = (\#w\%, l + r, q', [a', 1, 4][b_1, 1, 2]64[b_2, 1, 2][b_3]2[c_1][c_2], 1).$$

B. (1) For every rule in Π of the form $(b, q, a) \to (r, q', pop(1))$, there are rules in Π' of the form $(b, q, [a, p_1, \dots, p_{\rho(a)}]) \to (r, q', pop)$ for all $1 = p_1 < p_2 < \cdots < p_{\rho(a)} \leq r_{\max}$.

(2) For every rule in Π of the form $(b, q, a) \to (r, q', pop(i))$, where $2 \leq i \leq \rho(a)$, there are rules in Π' of the form $(b, q, [a, p_1, \dots, p_{\rho(a)}]) \to (r, q', pop_{p_i})$ for all $1 = p_1 < p_2 < \cdots < p_{\rho(a)} \leq r_{\max}$.

C. We need additional rules in Π' for popping along the pointer chains. These are all the rules of the form $(b, q, i) \to (0, q, pop_i)$, for all $b \in B \cup \{\#, \%\}$, for all $q \in Q$, and for all $i \in \{1, \dots, N_{\max}\}$.

D. For every rule in Π of the form $(b, q, a) \to (r, q', down(i))$, there are rules in Π' that enable the 1spda N to go down into its stack following the pointer chain that leads to the ith son of the current node. For every pointer in this pointer chain, N pushes, on the way to the stack square to which the pointer refers, the dummy symbol $+$ on its pushdown and when N reaches the required square the used pointer will be pushed. Formally we obtain:

(1) For every rule in Π of the form $(b, q, a) \to (r, q', down(1))$, there are rules in Π' of the form $(b, q, [a, p_1, \ldots, p_{\rho(a)}]) \to (r, q'_d, down, 1)$ and $(b, q, [a, p_1, \ldots, p_{\rho(a)}], c) \to (r, q'_d, down, 1)$ for all $1 = p_1 < p_2 < \cdots < p_{\rho(a)} \leq r_{\max}$ and for all $c \in C$.
(2) For every rule in Π of the form $(b, q, a) \to (r, q', down(i))$, where $2 \leq i \leq \rho(a)$, there are rules in Π' of the form $(b, q, [a, p_1, \ldots, p_{\rho(a)}]) \to (r, q'_d, D_{p_i})$ and $(b, q, [a, p_1, \ldots, p_{\rho(a)}], c) \to (r, q'_d, D_{p_i})$ for all $1 = p_1 < p_2 < \cdots < p_{\rho(a)} \leq r_{\max}$ and for all $c \in C$.
(3) Additionally we need the rules $(b, q_d, [a, p_1, \ldots, p_{\rho(a)}], c) \to (0, q, stay)$ and $(b, q_d, i, c) \to (0, q_d, D_i)$, for all $b \in B \cup \{\#, \%\}$, for all $a \in A$, for all $q \in Q$, for all $1 = p_1 < p_2 < \cdots < p_{\rho(a)} \leq r_{\max}$, and for all $c, i \in \{1, \ldots, N_{\max}\}$.

Example: The following configurations are given. Let

$$C_M = (\#w\%, l, q, a'\langle b_1\langle c_1, c_2\rangle, b_2\langle b_3, c_2\rangle\rangle, 1)$$

and let a corresponding configuration be

$$C_N = (\#w\%, l, q, [a', 1, 4]\{[b_1, 1, 2], 1\}55[b_2, 1, 2][b_3]2[c_1][c_2], 2).$$

If a down rule $(b, q, b_1) \to (r, q', down(2))$, where b is the lth symbol of w, is applied, then we obtain

$$C'_M = (\#w\%, l + r, q', a'\langle b_1\langle c_1, c_2\rangle, b_2\langle b_3, c_2\rangle\rangle, 1 \cdot 2).$$

The 1spda uses rules D(2) and D(3) to obtain a corresponding configuration to C'_M. First a rule $(b, q, [b_1, 1, 2], 1) \to (r, q'_d, D_2)$ is applied and we obtain

$$C^1_N = (\#w\%, l + r, q'_d, [a', 1, 4]\{[b_1, 1, 2], 1\}\{5, +\}\{5, 2\}$$
$$[b_2, 1, 2][b_3]2[c_1][c_2], 4).$$

Next a rule $(b', q'_d, 5, 2) \to (0, q'_d, D_5)$, where b' is the $(l + r)$th symbol of w, is applied and then we obtain

$$C^2_N = (\#w\%, l + r, q'_d, [a', 1, 4]\{[b_1, 1, 2], 1\}\{5, +\}\{5, 2\}$$
$$\{[b_2, 1, 2], +\}\{[b_3], +\}\{2, +\}\{[c_1], +\}\{[c_2], 5\}, 9).$$

Finally, a rule $(b', q'_d, [c_2], 5) \to (0, q', stay)$ is applied and so the resulting configuration

$$C'_N = (\#w\%, l + r, q', [a', 1, 4]\{[b_1, 1, 2], 1\}\{5, +\}\{5, 2\}$$
$$\{[b_2, 1, 2], +\}\{[b_3], +\}\{2, +\}\{[c_1], +\}\{[c_2], 5\}, 9)$$

corresponds to C'_M.

E. For every rule in Π of the form $(b, q, a) \to (r, q', up)$, there are rules in Π' that enable the 1spda N to go up as many squares as the pointers on the pushdown indicate. Remember that these pointers refer to a square which corresponds to the father of the current node of the tree-stack. For example, if we have a configuration C'_N as in the last example and a rule $(b', q', c_2) \to (r', q'', up)$, where b' is the $(l + r)$th symbol of the input w, then N goes up seven squares. Formally we obtain:

   (1) For every rule in Π of the form $(b, q, a) \to (r, q', up)$, there are rules in Π' of the form $(b, q, [a, p_1, \dots, p_{\rho(a)}], c) \to (r, q'_u, U_c)$ for all $1 = p_1 < p_2 < \cdots < p_{\rho(a)} \leq r_{\max}$, and for all $c \in \{1, \dots, N_{\max}\}$.
   (2) Additionally, we need the rules $(b, q_u, [a, p_1, \dots, p_{\rho(a)}]) \to (0, q, stay)$ and $(b, q_u, [a, p_1, \dots, p_{\rho(a)}], c) \to (0, q, stay)$ and $(b, q_u, i, c) \to (0, q_u, U_c)$ for all $b \in B \cup \{\#, \%\}$, for all $a \in A$, for all $q \in Q$, for all $1 = p_1 < p_2 < \cdots < p_{\rho(a)} \leq r_{\max}$, and for all $c, i \in \{1, \dots, N_{\max}\}$.

F. For every rule in Π of the form $(b, q, a) \to (r, q', stay)$, there are rules in Π' of the form $(b, q, [a, p_1, \dots, p_{\rho(a)}]) \to (r, q', stay)$ and $(b, q, [a, p_1, \dots, p_{\rho(a)}], c) \to (r, q', stay)$ for all $1 = p_1 < p_2 < \cdots < p_{\rho(a)} \leq r_{\max}$ and for all $c \in C$.

Since the simulation of a 1tsa by a 1spda as described in Construction 1 should be intuitively clear, we do not give a formal correctness proof of the construction. Thus we obtain the following result.

Lemma 3.6. *$\mathcal{L}(x1TSA) \subseteq \mathcal{L}(x1SPDA)$ for $x \in \{nd, d, ndne, dne, ndc, dc\}$.*

If we consider Construction 1, then we can mention that an ordinary 1sa can simulate the push, pop, and down moves of the 1tsa. The critical situation, why a 1sa cannot simulate a 1tsa, arises during the reading-mode. The linear automaton cannot distinguish between two occurrences of the same label (the label c_2 in the example of Construction 1 part D). There are two pointer chains which refer to c_2 and it cannot remember which of these chains it has used. The problem is solved by pushing this information in an auxiliary store. Therefore we really need a 1spda.

If we extend Construction 1 in the form that additionally, for every node, a pointer to its "father" is encoded in the stack symbols, we get the following statement which expresses that a restricted form of copying in the tree-stack does not increase the recognition power of a 1tsa in relation to the recognition power of a 1sa. The detailed proof is left to the reader.

Theorem 3.7. *$\mathcal{L}(x1TSA) = \mathcal{L}(x1SA)$ for $x \in \{nd, d, ndne, dne, ndc, dc\}$ with the restriction that on the right-hand sides of the push rules of the 1tsa the nodes labeled by x_j have the same father.*

Example. $(b, q, a) \to (r, q', push(a'\langle b_1\langle x_1, x_1\rangle, x_2\rangle))$ where $\rho(a) = 2$ is a rule of this form.

The next goal is to show the reverse statement of the last lemma, i.e., $\mathcal{L}(1SPDA) \subseteq \mathcal{L}(1TSA)$. We can prove this statement by another construction, where we arrange the stack-pushdown in the form of a tree-stack. The construction which follows is a variation of a well-known idea in tree language theory that was introduced in the proof of Theorem 7 of [23]. It was also used on p. 175 of [5], in Lemma 5.4 of [9], and in Lemma 5.10 of [10].

Construction 2. For a given 1spda $N = (A, C, B, Q, F, q_0, a_0, \Pi)$ we construct a 1tsa $M = (\langle A', \rho\rangle, B, Q', F, q_0', a_0', \Pi')$ that simulates N and hence accepts the same language.

Assume $C = \{c_1, \ldots, c_k\}$. We define $\langle A', \rho\rangle$, Q' by $A' = A \cup \{[a, c] \in A \times C\} \cup \{a_0'\}$, $\rho(a) = \rho([a, c]) = k$ and $\rho(a_0') = 0$, and $Q' = Q \cup \{q_d | q \in Q\} \cup \{q_0'\}$. First we need a rule in Π' so that M can go in a configuration corresponding to N's initial configuration. So, Π' obtains a rule $(\#, q_0', a_0') \to (0, q_0, push(a_0\langle a_0', \ldots, a_0'\rangle))$.

Construction of Π'

A. For every rule in Π of the form $(b, q, a) \to (r, q', push(v))$, where $v = a_1 \cdots a_n$ and $n \geq 1$, there is a rule in Π' of the form $(b, q, a) \to (r, q', push(t))$ with

$$t = a_1\langle [a_2, c_1]\langle \cdots \langle t_1, \ldots, t_k\rangle \cdots \rangle, \ldots, [a_2, c_k]\langle \cdots \langle t_1, \ldots, t_k\rangle \cdots \rangle\rangle,$$

where $t_i = [a_n, c_i]\langle x_1, \ldots, x_k\rangle$ for all $i \in \{1, \ldots, k\}$.

That means that the simulating 1tsa pushes in tree form all later possible stack-pushdown forms, where the nodes of the tree-stack below the root have coded the stack symbol and the pushdown symbol.

B. For every rule in Π of the form $(b, q, a) \to (r, q', pop)$, there are rules in Π' of the form $(b, q, a) \to (r, q_d', pop(1))$ and $(b', q_d', [a', c_1]) \to (0, q', push(a'\langle x_1, \ldots, x_k\rangle))$ for all $b' \in B \cup \{\#, \%\}$, and for all $a' \in A$.

If the 1spda pops one symbol, the 1tsa also pops the root and changes the new root, which actually consists of a pair [*stacksymbol, pushdownsymbol*], so that only the stack symbol appears.

C. For every rule in Π of the form $(b, q, a) \to (r, q', down, c_i)$ (resp. $(b, q, a, c) \to (r, q', down, c_i)$), $1 \leq i \leq k$, there is a rule in Π' of the form $(b, q, a) \to (r, q', down(i))$ (resp. $(b, q, [a, c]) \to (r, q', down(i))$).

Now we regard the reading-mode. The 1spda moves down and pushes a c_i on its pushdown. Then the 1tsa goes down to the ith son, which has the encoded information [*stacksymbol*, c_i].

D. For every rule in Π of the form $(b, q, a, c) \to (r, q', up)$, there is a rule in Π' of the form $(b, q, [a, c]) \to (r, q', up)$.

If the 1spda moves up and pops the top of the pushdown, then the 1tsa moves up.

E. For every rule in Π of the form $(b, q, a) \to (r, q', stay)$ (resp. $(b, q, a, c) \to (r, q', stay)$), there is a rule in Π' of the form $(b, q, a) \to (r, q', stay)$ (resp. $(b, q, [a, c]) \to (r, q', stay)$).

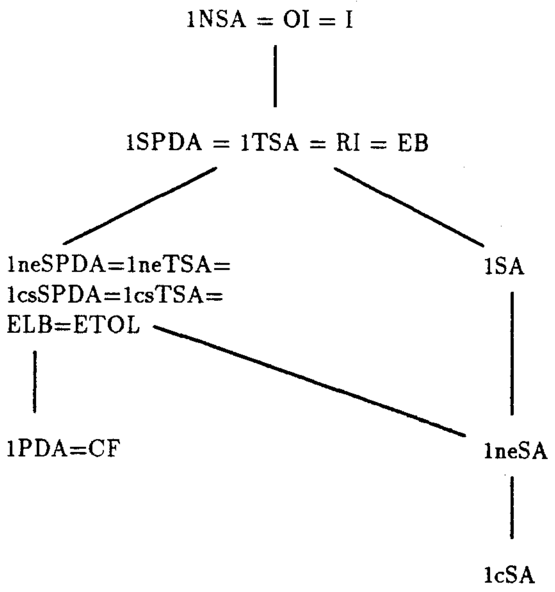

Fig. 4. The relation between automata models and well-known language classes. NSA: Nested stack automata; OI: OI-macro grammars; I: Indexed grammars; RI: Restricted indexed grammars; EB: Extended basic grammars; ELB: Extended linear basic grammars; ETOL: ETOL-systems; CF: Contextfree grammars. For more details see [7].

A formal correctness proof of Construction 2 is left to the reader.

Lemma 3.8. $\mathcal{L}(x1SPDA) \subseteq \mathcal{L}(x1TSA)$ *for* $x \in \{nd, d, ndne, dne, ndc, dc\}$.

The main result of our paper is stated in the next theorem.

Theorem 3.9. $\mathcal{L}(x1TSA) = \mathcal{L}(x1SPDA)$ *for* $x \in \{nd, d, ndne, dne, ndc, dc\}$.

All results for a 1spda can be taken over for a 1tsa and also the relationship between various types of one-way stack-pushdown automata can be taken over for one-way tree-stack automata.

We state some results about closure properties for the class of languages accepted by one-way tree-stack automata. The results stated are taken from [7].

Remark 1. $\mathcal{L}(1TSA)$ is a full AFL, i.e., closed under union, concatenation, Kleene star, homomorphism, inverse homomorphism, and intersection with regular sets.

Remark 2. The relation between automata models and well-known language classes (stated by grammar names) are given in Figure 4 where solid lines indicate proper inclusion.

Remark 3. It would not be difficult to define the notion of a tree-stack-pushdown automaton. Construction 2 could be easily generalized to show that tree-stack-pushdown automata have the same power as tree-stack automata.

As stated above one-way stack automata are an extended model of pushdown automata. If we consider the first construction and omit the part which is used for the reading-mode, then we obtain that by the pointer chains a linearization of a tree-pushdown is possible. So, an easy modification of Construction 1 leads to the following statement.

Theorem 3.10. $\mathcal{L}(x1TPDA) = \mathcal{L}(x1PDA)$ *for* $x \in \{nd, d\}$.

4. Two-Way Tree-Stack Automata

The input tape in Section 3 is restricted to the one-way case. Here we permit a two-way input tape. The question of our investigations is again the relationship between two-way tree-stack automata and two-way stack automata. The constructions of Section 3 can be taken over, because they are independent of the type of input tape. Therefore we immediately obtain the following result for two-way pushdown automata.

Theorem 4.1. $\mathcal{L}(x2TPDA) = \mathcal{L}(x2PDA)$ *for* $x \in \{nd, d\}$.

After reading the preceding section, we get immediately the following statement.

Lemma 4.2. $\mathcal{L}(x2TSA) = \mathcal{L}(x2SPDA)$ *for* $x \in \{nd, d, ndne, dne, ndc, dc\}$.

2nsa denotes a two-way nested stack automaton, introduced in [1]. It is well known, that two-way stack-pushdown automata are a special case of two-way nested stack automata, see [6].

Theorem 4.3. $\mathcal{L}(x2TSA) = \mathcal{L}(x2SA)$ *for* $x \in \{nd, d, ndne, dne\}$.

Proof. Beeri has shown in [2] that $\mathcal{L}(x2NSA) = \mathcal{L}(x2SA)$ for $x \in \{nd, d\}$ and together with Lemma 4.2 we get the statement. For the nonerasing case, in [21] Lippe has constructed for a ndne2tsa (resp. dne2tsa) a simulating space-bounded turing machine $space(s^2) - TM$ (resp. $space(s * \log_2 s) - TM$), which has the same recognition power as a ndne2sa (resp. dne2sa), that is shown in [16]. □

This result means that, in opposition to Section 3, the tree automata and the linear automata have the same recognition power.

Remark 4. The equivalence between tree-stack automata and stack-pushdown automata also holds for many other types of automata, such as multihead automata and transducers.

Acknowledgments

The authors would like to thank the anonymous referee for his many helpful comments and detailed suggestions for improving this paper.

References

[1] Aho, A. V. (1969), Nested stack automata, *J. Assoc. Comput. Mach.* **16**, 383–406.
[2] Beeri, C. (1975), Two-way nested stack automata are equivalent to two-way stack automata, *J. Comput. System Sci.* **10**, 317–339.
[3] Brainerd, W. S. (1969), Tree generating regular systems, *Inform. and Control* **14**, 217–231.
[4] Büchi, J. R. (1960), Regular canonical systems, *Archiv Math. Logik Grundlag.* **6**, 91–111.
[5] Damm, W. (1982), The IO and OI hierarchies, *Theoret. Comput. Sci.* **20**, 95–207.
[6] Engelfriet, J. (1991), Iterated stack automata and complexity classes, *Inform. and Comput.* **95**, 21–75.
[7] Engelfriet, J., Meineche Schmidt, E., and van Leeuwen, J. (1980), Stack machines and classes of nonnested macro languages, *J. Assoc. Comput. Mach.* **27**, 96–117.
[8] Engelfriet, J., Rozenberg, G., and Slutzki, G. (1980), Tree transducers, L systems, and two-way machines, *J. Comput. System Sci.* **20**, 150–202.
[9] Engelfriet, J., and Vogler, H. (1986), Pushdown machines for the macro tree transducer, *Theoret. Comput. Sci.* **42**, 251–368.
[10] Engelfriet, J., and Vogler, H. (1988), High-level tree transducers and iterated pushdown tree transducer, *Acta Inform.* **26**, 131–192.
[11] Ginsburg, S., Greibach, S. A., and Harrison, M. A. (1967), Stack automata and compiling, *J. Assoc. Comput. Mach.* **14**, 172–201.
[12] Ginsburg, S., Greibach, S. A., and Harrison, M. A. (1967), One-way stack automata, *J. Assoc. Comput. Mach.* **14**, 389–418.
[13] Greibach, S. A. (1969), Checking automata and one-way stack languages, *J. Comput. System Sci.* **3**, 196–217.
[14] Guessarian, I. (1983), Pushdown tree automata, *Math. Systems Theory* **16**, 237–263.
[15] Harrison, M. A., and Schkolnick, M. (1971), A grammatical characterization of one-way nondeterministic stack languages, *J. Assoc. Comput. Mach.* **18**, 148–172.
[16] Hopcroft, J. E., and Ullman, J. D. (1979), *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Reading, MA.
[17] Langmaack, H. (1973), On correct procedure parameter transmission in higher programming languages, *Acta Inform.* **2**, 110–142.
[18] Langmaack, H. (1973, 1974), On procedures as open subroutines, I, II, *Acta Inform.* **2**, 331–333, **3**, 227–241.
[19] Lippe, W.-M. (1976), Über die Entscheidbarkeit der formalen Erreichbarkeit von Prozeduren bei monadischen Programmen, 4. *Fachtagung über Programmiersprachen*, *Informatik-Fachberichte* 1, Springer-Verlag, Berlin, pp. 124–134.
[20] Lippe, W.-M., (1980), Erweiterungen von Dendrogrammatiken, *J. Inform. Process. Cybernet.* **16**, 21–39.
[21] Lippe, W.-M., (1983), Top-Down-Baumerzeugende Systeme und Automaten, Habilitation Thesis, University Kiel.
[22] Ogden, W. F., (1969), Intercalation theorems for stack languages, *Proc. ACM Symp. on Theory of Computing*, Marina del Rey, CA, pp. 31–42.
[23] Rounds, W. C. (1970), Mappings and grammars on trees, *Math. Systems Theory* **4**, 257–287.

Received February 16, 1993, *and in revised form November* 2, 1993, *and March* 18, 1994, *and in final form April* 28, 1994.