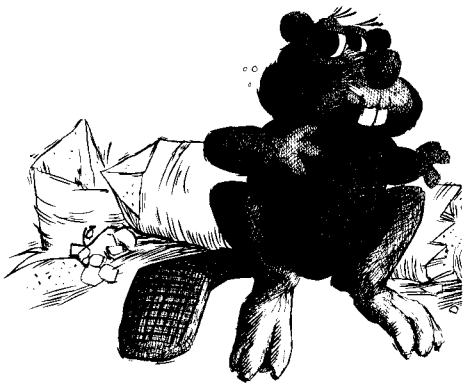
UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT



Synchronizable deterministic pushdown automata and the decidability of their equivalence

Karel Culik II
Juhani Karhumäki

Research Report CS-86-04

January 1986

UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT

Synchronizable deterministic pushdown automata and the decidability of their equivalence*

Karel Culik II

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1

Juhani Karhumäki†

Department of Mathematics
University of Turku
Turku, Finland

ABSTRACT

The notion of synchronized and synchronizable deterministic pushdown automata (DPDA's) is introduced. It is shown that the equivalence of two synchronized and even of synchronizable DPDA's can be tested. It is conjectured that every two equivalent DPDA's are synchronizable. It is also shown that the equivalence of two deterministic pushdown transducers whose underlying DPDA's are synchronized can be tested.

1. Introduction

Extensive efforts have been made in the past 20 years attempting to prove the decidability of the equivalence problem for deterministic pushdown automata (DPDA's). The problem still remains open; however, a number of partial results have been obtained. These results are surveyed in [20] by Tomita and classified into two "schools". The technique of the first school is called **branching algorithms** and was first used by Korenjak and Hopcroft [12] to show the decidability of the equivalence problem for simple deterministic grammars or equivalently simple DPDA's, i.e. single-state real-time DPDA's accepting by empty stack.

* This work was supported by the Natural Sciences and Engineering Research Council of Canada under Grant A-7403.

† This work has been done during the second author's visit at the University of Waterloo.

This technique has been mainly followed in [3,8,9,11,14,19,20,24].

The technique of the second school can be called **simultaneous simulation** and was first used by Rosenkrantz and Stearns to prove the decidability of the equivalence problem for $LL(k)$ grammars. When using this method we typically first construct a (nondeterministic) PDA M that simultaneously simulates the actions of two given DPDA's M_1 and M_2 so that M accepts a string iff M_1 and M_2 are nonequivalent. Then we use the decidability of the emptiness problem for context free languages. This second technique has been mainly followed in [2,6,13,15,16,18,21-23].

We suggest that the second school can be further subdivided into two groups. In the first group the composite PDA M directly simulates all computations of tested DPDA's M_1 and M_2, which is possible because M_1 and M_2 are essentially **synchronized** in the way they use the stack. Typical example of such synchronization is the alternate stacking technique of Valiant [21].

In the second group the composite PDA M modifies the computations of **almost synchronized** DPDA's M_1 and M_2 during the simulation whenever there is the danger of dissynchronization. An example of this technique is the parallel stacking technique of Valiant [22], here the replacement rules are the modifications preserving synchronization. This almost synchronization method has been the most powerful tool for testing equivalence of DPDA's to this date but it yet failed to solve the general case because it seems not possible to show that every two equivalent DPDA's are "almost synchronized" in the required sense.

Here we propose a new more powerful technique, again based on synchronization. First we define, in quite a general way, the notion of synchronization for DPDA's and show that two synchronized DPDA's can be tested for equivalence. Then the main idea is to attempt to prove that for any two equivalent DPDA's P and R (from certain class) there exists a chain of DPDA's $P = M_0, M_1, \ldots, M_n = R$, such that M_i and M_{i+1} are equivalent and synchronized for $i = 0, 1, \ldots, n-1$. If this is so, we call P and R **synchronizable**. We show that the equivalence of two synchronizable DPDA's can be tested. Thus in order to test their equivalence P and R do not need to be synchronized directly, but rather in several steps. Actually, $n = 2$, i.e. only one intermediate DPDA might be sufficient. We conjecture that every two equivalent DPDA's are synchronizable, however to prove that (even for some suitable normal form) is probably very difficult.

In section 4 we consider deterministic pushdown transducers (DPDT's). We say that two DPDT's are synchronized if their underlying DPDA's are synchronized. We show that two synchronized DPDT's can be tested for equivalence, as well. We reduce this problem to the morphic equivalence problem for context free languages [5]. This gives an alternate proof and slightly extends the main result (Theorem 1) in [10]. It implies, for example, that the equivalence problem is decidable for DPDT's based on real time DPDA's accepting with empty stack.

We also show that we cannot extend our synchronizability conjecture to DPDT's; we give an example of two equivalent DPDT's that does not seem to be synchronizable. Finally, we show that, given DPDT T_1 and DPDA M_2 synchronized with the underlying DPDA M_2 of T_1, we can decide whether there exists a

DPDT T_2 based on M_2 and equivalent to T_1. Moreover if it does exist, then we can construct it. As a corollary of this result we show that, given a DPDT, it is decidable whether there exists an equivalent DPDT with the same underlying DPDA and having only "empty" outputs for ϵ-moves. Again, if this is the case such DPDT can be constructed.

2. Preliminaries

We assume that the reader is familiar with the basic notions of deterministic and nondeterministic pushdown automata as well as formal languages in general, cf. e.g. [7]. So the following lines are mainly to fix our notation.

A **deterministic pushdown automaton** (DPDA for short) is a tuple $M = (Q, \Sigma, \Gamma, (s, Z), F, P)$, where Q is a **set of states**, Σ and Γ are **input** and **stack alphabets**, respectively, (s, Z) where, $s \in Q$ and $Z \in \Gamma$ designates the bottom of the stack (which is never moved), is the **initial mode**, $F \subseteq Q \times \Gamma$ is the set of **accepting modes** and $P \subseteq Q \times \Gamma \times (\Sigma \cup \{\epsilon\}) \times Q \times \Gamma^*$ is the set of **transition rules** satisfying the well known "determinism restriction", cf. [7].

The transition (q, A, a, q', α), denoted also as $(q, A) \xrightarrow{a} (q', \alpha)$), is called **$\epsilon$-move** or **reading move** depending on whether a is the empty word ϵ or in Σ. Correspondingly, the mode (q, A) is called **ϵ-mode** or **reading mode**. The language **accepted** by M is denoted by $L(M)$, for the detailed definition cf. [7]. We use the abbreviation PDA for a (not necessarily deterministic) pushdown automaton.

Our model of DPDA accepts with final modes instead of final states; however, this model is clearly equivalent as regards to the language definition power. Moreover, we may assume without loss of generality that the set of accepting modes is a subset of reading modes.

A **deterministic pushdown transducer** (DPDT for short) is obtained from a DPDA by adding to its each transition an output word $w \in \Delta^*$. If T is a DPDT and we omit from it the outputs we obtain the **underlying** DPDA of T. It follows from the requirement that the set of accepting modes is a subset of the reading modes that our DPDT's are **single-valued**, i.e., for each input word there exists at most one output word. If ϵ-moves are allowed at the end of computations of underlying automata, then the corresponding transducers need not to be single-valued; therefore, we feel that our choice of definition is well motivated since, intuitively, determinism implies single-valuedness.

Finally, we say that two DPDA's are **equivalent** if they accept the same language, and that two DPDT's are **equivalent** if they define the same translation (input-output relation). Clearly, the equivalence of DPDT's requires that their domains are the same, i.e., that the underlying automata are equivalent.

3. Synchronized and synchronizable DPDA's

Let M_1 and M_2 be two DPDA's with common input alphabet Σ, $M_i = (Q_i, \Sigma, \Gamma_i, (s_i, Z_i), F_i, P_i)$ for $i = 1, 2$. Without loss of generality, we assume that $\Gamma_1 \cap \Gamma_2 = \varnothing$, and also that as M_i's are in a **normal form** namely they do not have any pushing ϵ-moves. We define the **crossproduct** $M_1 \times M_2$ of M_1 and M_2 as one stack machine accepting a subset of Σ^*, having

$(Q_1 \times \Gamma_1 \times Q_2 \times \Gamma_2)$ as the set of states and $\Gamma_1 \cup \Gamma_2$ as the stack alphabet, and operating in the following way: Let $M_1 \times M_2$ be in a state (q_1, A_1, q_2, A_2) and having $\overline{w}_n w_n \overline{w}_{n-1} w_{n-1} \cdots \overline{w}_0 w_0$, with $w_i \in \Gamma_1^+$ and $\overline{w}_i \in \Gamma_2^+$, for $i = 0, \ldots, n-1$, and $w_n \in \Gamma_1^+$, $\overline{w}_n \in \Gamma_2^*$, in its stack. Then

(i) if (q_i, A_i) is a reading mode for $i = 1, 2$ and $(q_i, A_i) \xrightarrow{a} (p_i, \alpha_i)$ is in p_i, $M_1 \times M_2$ when reading symbol a moves to the state $(p_1, B_1, p_2 B_2)$, where B_1 (resp. B_2) is the first symbol of $\alpha_1 w_n$ (resp. $\alpha_2 \overline{w}_n \overline{w}_{n-1}$) and changes the stack contents to $\overline{w}_{n+1} w_{n+1} \overline{w}'_n w'_n \overline{w}_{n-1} w_{n-1} \cdots \overline{w}_0 w_0$, where $w_{n+1} = \epsilon$ and $w'_n = B_1^{-1} w_n$ (resp. $\overline{w}_{n+1} = \epsilon$ and $\overline{w}'_n = B_2^{-1} \overline{w}_n$) if $\alpha_1 = \epsilon$ (resp. $\alpha_2 = \epsilon$), and if $\alpha_1 \neq \epsilon$ (resp. $\alpha_2 \neq \epsilon$) then $w_{n+1} = B_1^{-1} \alpha_1$ and $w'_n = w_n$ (resp. $\overline{w}_{n+1} = B_2^{-1} \alpha_2$ and $\overline{w}'_n = \overline{w}_n$).

(ii) if (q_1, A_1) (resp. (q_2, A_2)) is an ϵ-mode and $(q_1, A_1) \xrightarrow{\epsilon} (p_1, \epsilon)$ is in M_1 (resp. $(q_2, A_2) \xrightarrow{\epsilon} (p_2, \epsilon)$) is in M_2), then $M_1 \times M_2$ moves to a state (p_1, B_1, q_2, A_2) (resp. (q_1, A_1, p_2, B_2), where B_1 (resp. B_2) is the first symbol of w_n (resp. $\overline{w}_n \overline{w}_{n-1}$) and changes in its stack contents w_n to $B_1^{-1} w_n$ (resp. $\overline{w}_n$ to $B_2^{-1} \overline{w}_n$ if $\overline{w}_n \neq \epsilon$ and $\overline{w}_{n-1}$ to $B_2^{-1} \overline{w}_{n-1}$ otherwise).

Initially, $M_1 \times M_2$ is in the state (s_1, Z_1, s_2, Z_2), and it accepts an input word w if, after reading w, it is in a state (q_1, A_1, q_2, A_2), where (q_i, A_i), for $i = 1, 2$ is an accepting mode of M_i.

A few remarks concerning the definition of $M_1 \times M_2$ are in order. The whole construction resembles Valiant's "alternating stacking" construction, cf. [21]. However, we generalize it by allowing ϵ-moves as well as the acceptance with nonempty stack. The nondeterminism of $M_1 \times M_2$ (due to (ii)) could be

avoided by modifying the definition; however, this does not seem to yield any advantage.

It is a straightforward consequence of the construction that $M_1 \times M_2$ simulates both M_1 and M_2 and hence we have:

Lemma 1. $L(M_1 \times M_2) = L(M_1) \cap L(M_2)$.

From Lemma 1 it follows that the behaviour of $M_1 \times M_2$ cannot be described by either a DPDA or even by PDA (or other types of "simple" machines) in general. However, the main point in this construction is that the machine $M_1 \times M_2$ simulates M_1 and M_2 in a "synchronous" way, i.e., input symbols are read simultaneously, and so for some structurally similar DPDA's $M_1 \times M_2$ might be "simple".

We proceed by looking for conditions under which the simulating machine $M_1 \times M_2$ is simple enough to have the emptiness problem decidable. Let k be a natural number. We say that a computation in $M_1 \times M_2$ is **k-bounded** if, in any step of this computation, the new stack configuration depends only on the current state of the machine and the k topmost symbols of the stack. The DPDA's M_1 and M_2 are **k-synchronized** if all the computations for the words in $L(M_1) \cup L(M_2)$ are k-bounded, and the machines are **synchronized** if they are k-synchronized for some k.

As an easy consequence of the above we have:

Lemma 2. If M_1 and M_2 are synchronized, then $M_1 \times M_2$ can be simulated by a PDA.

We also have the following important lemma:

Lemma 3. Let k be a natural number. It is decidable whether two given DPDA's are k-synchronized.

Proof. We define a machine M' which operates as follows:

(i) it simulates all k-bounded computations of $M_1 \times M_2$;

(ii) in each step of a simulation it detects whether k-boundedness property is violated and if so it nondeterministically chooses the number 1 or 2 for i and from that point on simulates the machine M_i. M' accepts an input word if and only if it is, after reading the whole word and making the above nondeterministic choice, in an accepting mode of M_1 or M_2. Clearly, M' can be realized by a PDA, and moreover, M_1 and M_2 are k-synchronized if and only if the language of M' is empty. Hence, the lemma follows. □

Although we do not know how to decide whether M_1 and M_2 are synchronized, it follows straightforwardly from the proof of Lemma 3 that if we know that they are synchronized we can effectively find a k such that they are k-synchronized and therefore also a simulating (push-down) machine for $M_1 \times M_2$. Having this simulating PDA M' we check the equivalence of M_1 and M_2 by modifying M' as follows: (i) if a computation in M' is blocked the new machine simulates the nonblocking (if such exists) M_i and accepts if M_i accepts; (ii) if the whole input word is read by M', then the new machine does the same and accepts if exactly one of the simulated machines is in an accepting mode. Clearly, the testing of the emptiness for this new machine provides a test for the equivalence of M_1 and M_2. So we have proved:

Theorem 1. Given two synchronized DPDA's, it is decidable whether they are equivalent.

After having Theorem 1 an interesting question arises: When two DPDA's are synchronized?

Example 1. Any DPDA is synchronized with any deterministic finite automaton. This is immediate since the stack of the simulating machine contains only symbols from the stack alphabet of the DPDA.

Example 2. Any **equivalent** pair of realtime DPDA's accepting with the empty stacks is synchronized. This is proved in [16] as a generalization of Valiant's earlier proof for nonsingular DPDA's, cf. [21].

Example 3. Consider the language $a^*b^*c^*$ and two DPDA's M_1 and M_2 accepting it: M_1 when reading a's and b's pushes them into the stack, while reading c's pops one symbol in each step as long as there are symbols in the stack and after that do not change the stack at all. M_2, in turn, pushes only a's into the stack and starts popping already when reading b's and continue that (as long as the stack is nonempty) when reading c's. Clearly M_1 and M_2 are not synchronized. However, the use of the stack is now inessential since $a^*b^*c^*$ is regular, so that it can be accepted by a deterministic finite automaton M. Moreover, by Example 1, both M_1 and M_2 is synchronized with M. In particular, this shows that the property of being synchronized is not transitive.

Example 3 naturally suggests the following generalized definition. Two DPDA's $M_1(0)$ and $M_2(0)$ are **synchronizable** if there exists a natural number t and DPDA's $M_1(j)$ and $M_2(j)$, for $j = 1, \ldots, t$, such that $M_i(j)$ and

$M_i(j+1)$ are equivalent and synchronized for $i = 1, 2$ and $j = 0, \ldots, t-1$, and in addition, $M_1(t)$ and $M_2(t)$ are synchronized.

Theorem 1 can be strengthened to

Theorem 2. Given two synchronizable DPDA's, it is decidable whether they are equivalent.

Proof. Let $M_1(0)$ and $M_2(0)$ be two synchronizable DPDA's. We guess a natural number t and DPDA's $M_i(j)$, for $i = 1, 2$ and $j = 1, \ldots, t$, satisfying the properties required in the definition of the synchronizability. By Lemma 3, we can check whether our guess is correct and therefore, by Theorem 1, we can decide the equivalence of $M_1(0)$ and $M_2(0)$. □

Although it is probably very difficult to show that two DPDA's are synchronizable, we feel that this framework might be fruitful in attacking the equivalence problem for DPDA's. In particular, an important question is: Are all equivalent DPDA's synchronizable? We conjecture that this might be the case, and hence, our Theorem 2 might yield an affirmative answer to the equivalence problem for DPDA's.

As a related open problem we ask whether or not, for each two equivalent DPDA's M_1 and M_2, there exists a third DPDA M which is synchronized with both M_1 and M_2. In other words, can two equivalent DPDA's always be synchronized via a third one?

4. Synchronizing transducers

In this section we discuss in what regard, if at all, our notions of synchronized and synchronizable can be extended to DPDT's. We start with an example indicating difficulties.

Example 4. Consider DPDT's T_1 and T_2 which translate the word $a^n b^m \#$ into the word $a^n b^n$ and operate as follows: T_1 when reading a's produces them as outputs and pushes for each a another one into the stack. When seeing first b (or endmarker $\#$) T_1 starts to pop a's from the stack by reading the empty word and producing output b in each step. When the stack is empty it continues to read b's (and $\#$) without producing any output. T_2 behaves like T_1 when reading a's but when seeing b's it reads them, simultaneously outputs b's, and pops one a from the stack as long as there are a's there. From the point on when there are no a's in the stack T_2 continues reading b's without producing outputs, and if there are still some a's in the stack when all the b's are read, then T_2 pops them by ϵ-moves and outputs b in each move.

Clearly, the underlying automata of T_1 and T_2 are not synchronized; therefore T_1 and T_2 are not synchronized either. On the other hand, the underlying automata are synchronizable (since the domains are regular), but this does not synchronize the transducers, i.e., does not provide a chain of equivalent DPDT's such that, for each consecutive two, there exist a single PDT which simulates simultaneously both these transducers. In fact, T_1 and T_2 seems to be nonsynchronizable in this sense.

The above example indicates that the equivalence problem for DPDT's is essentially more difficult than for DPDA's. In particular, we cannot make the similar conjecture as we made for DPDA's in the previous section. For this reason we define formally only the notion of synchronized transducer by saying that two DPDT's are **synchronized** if their underlying DPDA's are synchronized. In this case Theorem 1 can be strengthened for DPDT's, too. In order to be allowed to use results of the previous section we note that the normal form of Section 3 (no pushing ϵ-moves) can be assumed here, too.

Theorem 3. Given two synchronized DPDT's, it is decidable whether they are equivalent.

Proof. Let T_1 and T_2 be synchronized DPDT's and let M_1 and M_2 be the underlying DPDA's, respectively. We first test, by Theorem 1, whether M_1 and M_2 are equivalent. If not we are done; so assume that they are equivalent. Then, as we described in the previous section, we can find a PDA M simulating simultaneously M_1 and M_2. Moreover, this simulation is "faithful" in the sense that all the successful computations of M_1 and M_2 can be recovered from those of M, or formally, for $i = 1, 2$, there exists a weak coding c_i (a morphism mapping letters into letters or the empty word) mapping the set of all successful computations of M onto those of M_i. Therefore the testing of the equivalence of T_1 and T_2 can be reduced to the testing of whether two morphisms are equivalent on the language corresponding to all successful computations of a PDA. Such a language is context-free, as is easy to see, and so Theorem 3 follows since the

morphic equivalence for context-free languages is decidable, cf. [5] or [1]. □

Theorem 3 provides an alternate proof and a slight generalization of Theorem 1 in [10]. For example, it shows that the equivalence problem is decidable for DPDT's based on realtime DPDA's accepting with empty stack, cf. Example 2. It also shows that the equivalence of a DPDT and a deterministic finite transducer can be decided, cf. Example 1.

We conclude this section with a couple of results which, besides being interesting on their own, might be useful in proving the decidability of the equivalence problem for some classes of DPDT's.

Theorem 4. Let T_1 be a DPDT and M_2 a DPDA such that the underlying automaton M_1 of T_1 and M_2 are synchronized. Then it is decidable whether there exists a DPDT T_2, with M_2 as the underlying automaton, such that T_1 and T_2 are equivalent, and if such a T_2 exists it can be found effectively.

Proof. Again we first test, by Theorem 1, whether M_1 and M_2 are equivalent, and if not we are done. So assume that they are equivalent. We attach to each transition of M_2 an output which we consider as an unknown. Let the DPDT thus obtained be $T_2(X)$ where X denotes the set of the unknowns. Now, the fact that T_1 and $T_2(X)$, for a fixed assignment of the unknowns, are equivalent on a word $w \in L(M_1)$ means that the assignment is a solution of the equation

$$T_1(w) = T_2(X)(w)\ , \tag{1}$$

where the left hand side is a word over the output alphabet of T_1 and the right hand side is a word over the set X of unknowns. More generally, all the solutions of the (infinite) system

$$T_1(w) = T_2(X)(w),\ \ w \in L(M_1) \tag{2}$$

of equations characterizes all those transducers obtained from $T_2(X)$ by fixing the variables which are equivalent to T_1.

Now, we use the fact that M_1 and M_2 are synchronized. It means that the system (2) is algebraic, cf. [4], and therefore there effectively exists a finite equivalent subsystem of (2), cf. [1]. Hence, Theorem 4 follows, since there remains only a finite number of possibilities which has to be checked. □

As a special case of the above proof we obtain:

Corollary. Given a DPDT, it is decidable whether there exists an equivalent DPDT with the same underlying automaton and having only empty outputs for ϵ-moves. If this is the case such a DPDT can be effectively found.

Finally, we note that using the ideas of the proof of Theorem 4 we can find effectively, for a given deterministic finite transducer, an equivalent minimal (with respect to the number of states, for example) DPDT. A proof of Theorem 4 based directly on Theorem 3 can be obtained, too.

References

1. Albert, J., Culik II, K., Karhumäki, J., Tests sets for context-free languages and algebraic systems of equations, Inform. and Control 52, 172-186 (1982).

2. Beeri, C., An improvement on Valiant's decision procedure for equivalence of deterministic finite turn pushdown machines, Theoret. Comput. Sci. 3, 305-320 (1976).

3. Courcelle, B., An axiomatic approach to the Korenjak-Hopcroft algorithms, in Proc. 8th ICALP, Lecture Notes in Computer Science No. 115, 393-407, Springer-Verlag, Berlin/New York 1981.

4. Culik II, K., Karhumäki, J., Systems of equations over a free monoid and Ehrenfeucht's conjecture, Discrete Math. 43, 139-153 (1983).

5. Culik, II, K., Salomaa, A., On the decidability of morphic equivalence for languages, J. Comput. System Sci. 17, 163-175 (1978).

6. Friedman, E.P., Greibach, S.A., Superdeterministic DPDA's: the method of accepting does affect decision problems, J. Comput. System Sci. 19, 79-117 (1979).

7. Harrison, M.A., Introduction to formal language theory, Addison-Wesley, Reading, Mass. (1978).

8. Harrison, M.A., Havel, I.M., Real-time strict deterministic languages, SIAM J. Comput. 1, 333-349 (1972).

9. Harrison, M.A., Havel, I.M., Yehudai, A., On equivalence of grammars through transformation trees, Theoret. Comput. Sci. 9, 173-205 (1979).

10. Ibarra, O., Rosier, L., On the decidability of equivalence for deterministic pushdown transducers, Inform. Proc. Letters 13, 89-93 (1981).

11. Katayama, T., Tsuchiya, N., Enomoto, H., On the decidability of equivalence for deterministic pushdown transducers, Trans. Inst. Electron. Commun. Engrs. Japan 58-D, 760-767 (1975).

12. Korenjak, A.J., Hopcroft, J.E., Simple deterministic languages, in Proc. IEEE 7th Annual Symposium on Switching and Automata Theory, 36-46, Berkeley, CA. 1966.

13. Linna, M., Two decidability results for deterministic pushdown automata, J. Comput. System Sci. 18, 92-107 (1979).

14. Olshansky, T., Pnueli, A., A direct algorithm for checking equivalence of $LL(k)$ grammars, Theoret. Comput. Sci. 4, 321-349 (1977).

15. Oyamaguchi, M., Honda, N., The decidability of equivalence for deterministic stateless pushdown automata, Inform. Contr. 38, 367-376 (1978).

16. Oyamaguchi, M., Honda, N., Inagaki, Y., The equivalence problem for real-time strict deterministic languages, Inform. Control 45, 90-115 (1980).

17. Rosenkrantz, D.J., Stearns, R.E., Properties of deterministic top-down grammars, Inform. Control 17, 226-256 (1970).

18. Taniguchi, K., Kasami, T., A result on the equivalence problem for deterministic pushdown automata, J. Comput. System Sci. 13, 38-50 (1976).

19. Tomita, E., A direct branching algorithm for checking equivalence of strict deterministics vs. $LL(k)$ grammars, Theoret. Comput. Sci. 23 (1983).

20. Tomita, E., A direct branching algorithm for checking equivalence of some classes of deterministic pushdown automata, Inform. and Control 52, 187-238 (1982).

21. Valiant, L.G., Decision procedures for families of deterministic pushdown automata, Ph.D. Thesis, Department of Computer Science, University of Warwick, Coventry, England 1973.

22. Valiant, L.G., The equivalence problem for deterministic finite-turn pushdown automata, Inform. Control 25, 123-133 (1974).

23. Valiant, L.G., Paterson, M.S., Deterministic one-counter automata, J. Comput. System Sci. 10, 340-350 (1975).

24. Wood, D., Some remarks on the KH algorithm for s-grammars, BIT 13, 476-489 (1973).