ON THE INCLUSION PROBLEM FOR VERY SIMPLE DETERMINISTIC PUSHDOWN AUTOMATA

ERKKI MÄKINEN

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF TAMPERE

REPORT A-1999-8

UNIVERSITY OF TAMPERE
DEPARTMENT OF COMPUTER SCIENCE
SERIES OF PUBLICATIONS A
A-1999-8, JUNE 1999

ON THE INCLUSION PROBLEM FOR VERY SIMPLE DETERMINISTIC PUSHDOWN AUTOMATA

ERKKI MÄKINEN

University of Tampere
Department of Computer Science
P.O.Box 607
FIN-33101 Tampere, Finland

ISBN 951-44-4611-9
ISSN 0783-6910

On the inclusion problem for very simple deterministic pushdown automata

Erkki Mäkinen [1]

Department of Computer Science, University of Tampere, P.O. Box 607, FIN-33101 Tampere, Finland

Abstract

We present a new algorithm for checking the inclusion of very simple deterministic pushdown automata. The inclusion "$L(M_1) \subseteq L(M_2)$?" is checked by first constructing a finite characteristic set R of M_1, and then checking whether or not the inclusion $R \subseteq L(M_2)$ holds.

Keywords: decidability problems, deterministic pushdown automaton, left Szilard language, characteristic set

1 Introduction

The inclusion problem for simple deterministic pushdown automata is known to be decidable [2]. Wakatsuki and Tomita [6] have later studied the inclusion problem for very simple deterministic pushdown automata. They have shown that there exists a direct branching algorithm for solving this problem.

In this paper we show that there exists even a simpler algorithm for solving the inclusion problem for very simple deterministic pushdown automata. When solving the problem "$L(M_1) \subseteq L(M_2)$?" the algorithm first constructs a characteristic set of M_1. Characteristic sets of languages are used in grammatical inference algorithms [1]. Yokomori [7] has defined characteristic sets for very simple context-free grammars, i.e., for the grammars generating the class of languages accepted by very simple deterministic pushdown automata. Given a language $L(M_1)$, its characteristic set R has the property that $L(M_1)$ is the smallest language in the language class in question containing R. Hence, if also $L(M_2)$ contains R, then the smallest language $L(M_1)$ containing R must be included in $L(M_2)$. Otherwise ($L(M_2)$ does not contain R), it is clear that

[1] E-mail: em@cs.uta.fi. Work supported by the Academy of Finland (Project 35025).

$L(M_1) \subseteq L(M_2)$ cannot hold. Thus, the characteristic set R is used as a kind of "test set": it is sufficient to check whether $L(M_2)$ contains the elements of the characteristic set.

The Wakatsuki-Tomita algorithm [6] is polynomial on the maximal *thickness* of the stack symbols. The thickness of a stack symbol is the length of the shortest derivation that makes the symbol go to empty. Thickness can be exponential on the description length of an automaton [6].

We give our algorithm using grammar notation, and it turns out that similar "thickness" concepts are needed here as well. Hence, the time complexity of our algorithm contains an exponential factor as is also the case in the earlier algorithm by Wakatsuki and Tomita. However, our algorithm is conceptually much simpler and its time complexity depends only on one of the two input automata (or grammars).

Yokomori [8, Cor. 20] has earlier mentioned the possibility to use characteristic sets in solving inclusion problems.

2 Preliminaries

We mainly follow the definitions and notations given in [3]. The empty string is denoted by λ, the length of a string w by $lg(w)$, and the cardinality of a set S by $\mid S \mid$.

Consider a stateless deterministic pushdown automaton $M = (\Gamma, \Sigma, \delta, Z_0)$ where Γ, Σ, and δ are the finite sets of stack symbols, input symbols, and transition rules, respectively, and Z_0 is the initial stack symbol. We write $A \rightarrow^a \theta$, if reading an input symbol a with A on the top of the stack causes an action which replaces A with θ on the stack. We say that M is a *very simple deterministic pushdown automaton* (vspda, for short) if for each a in Σ there is exactly one transition rule $A \rightarrow^a \theta$, for some A in Γ and θ in Γ^*.

A *configuration* is $\alpha \in \Gamma^*$, where the leftmost symbol of α is the top of the stack. Z_0 is the *initial configuration.* A vspda makes a *move* from configuration $A\omega$ to configuration $\theta\omega$, denoted by $A\omega \rightarrow^a \theta\omega$, if there is a transition rule $A \rightarrow^a \theta$. A *derivation* $\alpha_1 \Rightarrow^{a_1} \alpha_2 \Rightarrow^{a_2} \ldots \Rightarrow^{a_m} \alpha_m$ is a sequence of moves from configuration α_1 to configuration α_m reading the input string $a_1 \ldots a_m$. The language accepted by $M = (\Gamma, \Sigma, \delta, Z_0)$ is $L(M) = \{w \in \Sigma^* \mid Z_0 \Rightarrow^w \lambda\}$.

We denote context-free grammars as $G = (N, \Sigma, P, S)$, where N, Σ, and P, are the sets of nonterminals, terminals, and productions, respectively, and S is the start symbol. A *very simple context-free grammar* (vscfg, for short)

$G = (N, \Sigma, P, S)$ has productions of the form $A \to a\alpha$ where α is in N^*, and the terminal symbols appearing in the right hand sides are unique. A language L is a *very simple language* if there exists a vscfg generating L. Very simple languages are often called *left Szilard languages* [4].

There is an obvious bijective correspondence between vspda's and vscfg's [6,7], and between languages accepted by vspda's and left Szilard languages [4].

Given a vspda $M = (\Gamma, \Sigma, \delta, Z_0)$, we can construct the corresponding vscfg $G = (N, \Sigma, P, S)$ by setting $N = \Gamma$, $S = Z_0$, and taking, for each transition rule $A \to^a \theta$, a production $A \to a\theta$ to P. A similar straightforward construction is naturally possible to the opposite direction as well. Hence, in what follows we use grammar and automaton notations related to vspda's and vscfg's mixed without any danger of confusion.

We consider *reduced* grammars only, i.e., each terminal and nonterminal appears at least in one derivation from the start symbol to a terminal string. The corresponding assumption naturally holds for vspda's.

We need a grammar concept corresponding to the thickness of a stack symbol. The *thickness* of a nonterminal A in a vscfg $G = (N, \Sigma, P, S)$ is the smallest k such that there exists a derivation of the form $A \Rightarrow \alpha_1 \Rightarrow \ldots \Rightarrow \alpha_k = w, w \in \Sigma^+$. The thickness of G, denoted by $k(G)$, is the maximum thickness over G's nonterminals. Moreover, we define the *upper thickness* of a nonterminal A to be the greatest K such that there exists a non-recursive derivation of the form $A \Rightarrow \alpha_1 \Rightarrow \ldots \Rightarrow \alpha_K = w, w \in \Sigma^+$. The upper thickness of G, denoted by $K(G)$, is defined to be the upper thickness of the start symbol S.

For a vscfg $G = (N, \Sigma, P, S)$, we denote the number of nonterminals by n, $n = \mid N \mid$. A production with a nonterminal A in the left hand side is said to be an *A-production*. By m we denote the maximum number of A-productions over the nonterminals in G. As noted in [6], $k(G)$ can be exponential on n and m. The same naturally holds for $K(G)$.

3 Characteristic sets

A characteristic set of a vscfg G is a set R of words such that $L(G)$ is the smallest very simple language containing R. Yokomori [7] has presented an algorithm to infer G in the limit from R in polynomial time. R can be found with the help of the *characteristic graph* of G. The characteristic graph of a vscfg is a digraph whose vertices are labelled with certain nonterminal strings of G and whose edges are labelled with the terminals of G.

In order to construct characteristic graphs, we need the concept of *shortest-cut production* [7]. A shortest-cut production of a nonterminal A in G is the production which starts the shortest derivation from A to a terminal string. A nonterminal may have several shortest-cut productions.

Given a vscfg $G = (N, \Sigma, P, S)$ its characteristic graph $C(G)$ is defined as follows [7]:

- **A** For each production $S \to a\alpha$ in P take the edge $([S], [\alpha])$ labelled with a to the digraph. If $\alpha = \lambda$ then take the edge $([S], F)$ where F is a specific final vertex.
- **B** In steps $i = 2, 3, \ldots$ until the algorithm is not applicable to any vertex do the following:
  - **B1** Let $[A\omega]$, $A \in N$, $\omega \in N^*$, be a vertex inserted in step $i-1$. If A does not appear as the leftmost nonterminal in any of the vertices created in the previous steps then, for each production $A \to b\beta$ in P, take the edge $([A\omega], [\beta\omega])$ labelled with b to the digraph.
  - **B2** Otherwise (there is an earlier inserted vertex with A as the leftmost nonterminal) take, for one of the shortest-cut productions $A \to b\beta$ of A, the edge $([A\omega], [\beta\omega])$ labelled with b to the digraph. In both cases, if $\beta\omega = \lambda$, take the edge $([A], F)$ labelled with b to the digarph.

("Taking an edge $([\alpha], [\beta])$ to the digraph" means that either there already exist vertices $[\alpha]$ and $[\beta]$ in the digraph or that we now insert them.)

The characteristic set R is now obtained by concatenating the labels appearing in the paths from $[S]$ to the final vertex F as described in Algorithm 1.

Algorithm 1 (Characteristic set)
Input: *A characteristic graph $C(G)$.*
Output: *The corresponding characteristic set $R(C(G))$.*
begin

1. *Take to $R(C(G))$ the words related to paths from $[S]$ to F with no directed cycles.*
2. *For each directed cycle with the subword w related to it, find the shortest word v_1 from $[S]$ to a vertex $[\alpha]$ in the cycle. Moreover, let v_2 be the shortest from $[\alpha]$ F. Then take $v_1 w v_2$ to $R(C(G))$.*

end *{Algorithm}*

The characteristic set obtained is unique up to the choice of shortest-cut productions. As noted in [5], the cardinality of the output set of Algorithm 1 could be decreased by including more than one directed cycle in a single word. However, this would make the algorithm more complicated without having any real effect to the time complexity.

Proposition 1 *[7] Very simple languages have characteristic sets, and they can be effectively constructed by Algorithm 1.*

Knowing the size of the resulting characteristic set is essential for our purposes. We have the following rough upper bounds for the time complexity of Algorithm 1 and for the cardinality of characteristic sets obtained. Recall that we denote the thickness of G by $k(G)$, the upper thickness by $K(G)$, the number of nonterminals by n, and the maximum number of A-productions in G by m. (Notice that n^m is about the same size as $K(G)$.)

Theorem 2 *Algorithm 1 runs in time $O(k(G)mn)$. The cardinality of $R(C(G))$ is $O(n^m + k(G)mn)$.*

Proof. For each nonterminal A in G, $C(G)$ has, because of Step B1 above, at most $m+1$ vertices labelled with a string with A as the leftmost nonterminal. Moreover, there can be vertices labelled with a string having A as the leftmost nonterminal in the non-recursive derivations containing shortest-cut productions (Step B2 above). The total length of such non-recursive derivations is $O(k(G)mn)$. Hence, $C(G)$ has $O(k(G)mn)$ vertices.

When a nonterminal appears first time as the leftmost symbol in a label of a vertex, we take at most m new edges to $C(G)$. The shortest-cut derivations have at least one additional edge per vertex. Hence, there are $O(k(G)mn)$ edges in $C(G)$.

The time complexity is dominated by the construction of the digraph which takes time $O(k(G)mn)$.

When applying Step 1 of Algorithm 1, we can visit at most n vertices added in Step B1. Each such vertex allows at most m branching possibilities. In other vertices there is only one possibility to continue. Hence, Step 1 of Algorithm 1 adds at most $O(n^m)$ words to $R(C(G))$. Since there certainly are at most as many directed cycles in $C(G)$ as there are edges, Step 2 of Algorithm 1 adds at most $O(k(G)mn)$ words to $R(C(G))$. Hence, the cardinality of $R(C(G))$ is $O(n^m + k(G)mn)$.

Theorem 3 *The total length of the words in $R(C(G))$ is $O((n^m+k(G)mn)K(G))$.*

Proof. If a word in $R(C(G))$ does not contain a directed cycle, its length is at most $K(G)$. Each directed cycle in $C(G)$ has length at most $K(G)$. The length of words containing a directed cycle is at most $2K(G)$. By Theorem 2 the total length is $O((n^m + k(G)mn)K(G))$.

4 The algorithm

In the previous chapter we defined characteristic sets for vscfg's. Because of the one-to-one correspondence between vspda's and vscfg's, we could also talk about "the characteristic set of a vspda". We give our algorithm using grammar formalism for notational convenience.

Algorithm 2 (Inclusion)
Input: *Very simple context-free grammars G_1 and G_2.*
Output: *"Yes", if $L(G_1) \subseteq L(G_2)$, otherwise "No".*
begin
  construct $R(C(G_1))$ by Algorithm 1;
  if $R(C(G_1)) \subseteq L(G_2)$
    then *output("Yes")*
    else *output("No");*
end $\{$*Algorithm*$\}$

By the definition of characteristic sets and by Proposition 1 we have the following theorem.

Theorem 4 *Algorithm 2 correctly solves the inclusion problem for very simple context-free languages.*

Further, by the results of the previous chapter, we have

Theorem 5 *Algorithm 2 runs in time $O((n^m + k(G_1)mn)K(G_1))$.*

Proof. The inclusion $R(C(G_1)) \subseteq L(G_2)$ is easily checked by simulating derivations according to G_2. Hence, the time complexity of Algorithm 2 depends on the total length of the words in the characteristic set $R(C(G_1))$. The result now follows by Theorem 3.

By the one-to-one correspondence between vscfg's and vspda's, Theorems 4 and 5 imply the existence of an efficient algorithm for the inclusion problem for vspda's. Wakatsuki and Tomita [6] proved that their algorithm runs in polynomial time on the thickness of the stack symbols. Their exact time bound is given by a complex formula including f.ex. the factors $(pnt)^5$ and $k(G_2)^2$, where, expressed in grammar notation, and p is the number of nonterminals in the longest right hand side of G_1's productions, n is the number of nonterminals in G_1, t is the number of terminals in G_1, and $k(G_2)$ is the thickness of G_2.

The main difference in the time bounds is that the time bound of our algorithm depends on G_1 only, while the time complexity of the Wakatsuki-Tomita algorithm depends also on G_2.

5 Discussion

The use of characteristic sets is one way of avoiding over-generalization in language inference algorithm using positive data only [1]. As shown above, characteristic sets allows also efficient inclusion algorithms for the language classes in question. In the literature, characteristic sets are defined e.g. for k-reversible languages [1] and strictly deterministic languages [9]. Efficient inclusion algorithms for these subclasses of regular languages can be constructed following the above method.

Acknowledgement

The author wishes to thank Takashi Yokomori for valuable comments.

References

[1] Angluin, D., Inference of reversible languages. *J. ACM* **29** (1982), 741–765.

[2] Friedman, E.P., The inclusion problem for simple languages. *Theoret. Comput. Sci.* **1** (1976), 297–316.

[3] Harrison, M.A., *Introduction to Formal Language Theory* (Addison-Wesley, 1978).

[4] Mäkinen, E., On context-free derivations. *Acta Universitatis Tamperensis* **198**, 1985.

[5] Mäkinen, E., On grammatical inference and derivational complexity of context-free grammars. *Fundam. Inform.* **17** (1992), 363–368.

[6] Wakatsuki, M., and Tomita, E., A fast algorithm for checking the inclusion for very simple deterministic pushdown automata. *IEICE Trans. on Information and Systems* **E76-E** 10 (1993), 1224–1233.

[7] Yokomori, T., Polynomial-time learning of very simple grammars from positive data. *Proc. 4th Workshop on Computational Learning Theory*, pp. 213–227, 1991.

[8] Yokomori, T., Polynomial-time learning of very simple grammars from positive data. Report **CSIM 90-15**. University of Electro-Communications, Dept. of Computer Science and Information Mathematics, December 1990, Revised September 1993.

[9] Yokomori, T., On polynomial-time learnability in the limit of strictly deterministic automata. *Machine Learning* **19** (1995), 153–179.