A Note on the P-completeness of Deterministic One-way Stack Language

Klaus-Jörn Lange

(WSI, University of Tübingen, Germany
lange@linux.de)

Abstract: The membership problems of both stack automata and nonerasing stack automata are shown to be complete for polynomial time.

Keywords: Automata, Grammars, Complexity classes, Completeness

Categories: F.1.3, F.4.3

1 Introduction

Stack Automata (abbreviated SA) are a well-known extension of push-down automata. As those they have a stack which can be manipulated by push and pop operations. In addition, stack automata are allowed to read inside of their stack without modifying its content (see [GGH67b, GGH67a, Gre69]). A stack automaton which has no pop transitions, i.e.: is only able of pushing symbols and of stack reading, is called nonerasing (abbreviated NeSA). A checking stack automaton (ChSA for short) is a nonerasing stack automaton which executes no more push move after doing its first stack reading move. In the following a prefix D (resp. N) denotes a deterministic (resp. nondeterministic) automaton. In addition, a prefix 1- or 2- denotes whether the input head operates one-way or two-way. The abbreviations denote the corresponding language classes.

The complexities of the two-way case are well-known and show a uniform behaviour ([Iba71]):
- Both 2-DSA and 2-NSA are DEXPOLYTIME-complete.
- Both 2-DNeSA and 2-NNeSA are PSPACE-complete.
- 2-DChSA is DSPCE(log n)-complete, 2-NChSA is PSPACE-complete.

In the one-way case, the following is known ([GGH67a, Iba71, SB74, Rou73]):
- 1-NSA, 1-NNeSA, and 1-NChSA are NP-complete.
- 1-DSA and 1-DNeSA are contained in P.
- 1-DChSA is contained in LOGSPACE.

By saying that a class $F$ of formal languages is complete for a complexity class $C$ we mean that both $F$ is contained in $C$ and that $F$ contains some language which is $C$-complete.
This leaves open the exact complexity (wrt completeness) of both 1-DSA and 1-DNeSA. In the following section this question is solved in the affirmative by showing both classes to contain P-complete sets. Finally, the last section addresses the relations to Formal languages in terms of grammars subject to LL[1] or LR[1] conditions.

2 The P-completeness of deterministic stack languages

Theorem 1. Both 1-DSA and 1-DNeSA are P-complete.

Proof: It is sufficient to show the P-hardness of 1-DNeSA, since 1-DNeSA ⊂ 1-DSA. As a first step, the P-complete Monotone Circuit Value Problem ([Gol77]) is converted into a form recognizable by nonerasing stack automata.

We code a monotone, levelled circuit $C$ consisting of $\lor$- and $\land$-gates of fan-in two by a word $\langle C \rangle \in \#\{0,1\}^*(\#\{\lor,\land\}a^*ba^*)^\ast$. The $\{0,1\}^\ast$-prefix codes the input bits, i.e. the bottom level of the circuit. Then each level of the circuit is coded by a word in $(\{\lor,\land\}a^*ba^*)^\ast$. These words are separated by $\#$-symbols. The $j$-th gate $g_{i,j}$ of level $i$ with inputs $g_{i-1,k_1}$ and $g_{i-1,k_2}$ from level $i-1$ is coded as a string $\lor a^{k_1}ba^{k_2}$ if $g_{i,j}$ is an $\lor$-gate, and as $\land a^{k_2}ba^{k_2}$, otherwise. This string is placed as the $j$-th subword in the word beginning after the $i$-th occurrence of a $\#$-symbol.

The set $L := \{ \langle C \rangle \mid C$ is a levelled, monotone circuit and the last gate of the last level evaluates to 1 $\}$ is P-complete. We now show that there is an deterministic one-way nonerasing stack automaton $A$ accepting $L$.

$A$ works in phases one for each level of the circuit. After a phase the stack of $A$ will contain a sequence of 0s and 1s on its top which are the values of the evaluated gates of the corresponding level. The boolean values on top of the stack correspond to the rightmost gates of that level while the values further down represent the gates more to the left.

On input $w \in \#\{0,1\}^*(\#\{\lor,\land\}a^*ba^*)^\ast$ $A$ first pushes the $\#\{0,1\}^\ast$-$\#$-prefix onto its stack and then iteratively evaluates each level. Reading a symbol $\tau \in \{\lor,\land\}$ on its input tape $A$ goes into stack reading mode and reads down to the second last occurrence of a $\#$-symbol. This symbol marks the beginning of the last completely evaluated level. Then for each symbol $a$ on its input tape $A$ reads upwards in the stack one symbol in $\{0,1\}$ until a symbol $b$ is found on the input tape. Then the 0 or 1 found on the current reading position of the stack represents the value of the first input of the gate to be evaluated. This value is remembered by the finite memory of $A$. Now $A$ again reads down to the next $\#$-symbol (which marks the beginning of the last completely evaluated level) and performs the same procedure with the next string of $a$-symbols on the input tape coding the second input of the gate to be evaluated. Now the value of the
actual gate can be determined according to $\tau$ and the two values collected on
the stack. (In case of too many $a$-symbols on the input, $A$ rejects.) Then $A$
goed up to the top of the stack, turns into stack writing mode and pushes the new
value on top of the stack. This is repeated until a $\#$-symbol is read on the input
tape. $A$ then pushes a $\#$ on the stack and starts with the evaluation of the next
level. When a $\$ is found $A$ accepts if the last value written on the stack has been
a 1 and rejects otherwise. ☐

At this point readers familiar with Lindenmayer and Macro Languages might
wonder about the relations to the Deterministic Stack Push-Down Automata by
Engelfriet, Schmidt, and van Leeuwen in [ESvL80]. They looked at stack au-
tomata which do not read from an input tape but instead produce an output.
There is no difference between this generating and accepting power for nondeter-
mimistic machines, but it is crucial in the deterministic case. While deterministic
reading automata can branch their computations according to the input, deter-
mimistic writing automata have to make their decisions without getting informa-
tion by some input. How severe this restriction is can be seen by the fact, that
language families defined by these deterministic writing automata in [ESvL80]
are even not closed under inverse homomorphisms.

On the other hand, Engelfriet et al. extended the deterministic automaton
by the ability of pushing items nondeterministically on the stack, as long as
the automaton is not in stack reading mode\(^1\). It is this nondeterminism which
makes e.g. the membership problem of deterministic nonerasing stack push-down
languages $\text{NSPACE}(\log n)$-complete ([ESvL80, JS77]). If we would add this fea-
ture to deterministic reading automata, then deterministic nonerasing stack lan-
guages would no longer be contained in $P$ but could have an $NP$-complete
membership problem.

Since polynomially time bounded two-way deterministic (nonerasing) stack
automata which are augmented by a logarithmically space bounded working tape
(abbreviated as $\text{DAuxSA}_{pt}$ and $\text{DAuxNeSA}_{pt}$) obviously contain the $\text{LOG}$-
closure of 1-DSA (resp. 1-DNeSA), we get as a consequence:

\textbf{Corollary} [VC90] $P = \text{DAuxSA}_{pt} = \text{DAuxNeSA}_{pt}$.

3 Grammatical Determinism

The languages accepted by nondeterministic (one-way) automata are often repre-
sentable as those generated by grammars. Then the deterministic version of these
automata usually corresponds to grammars subject to an $LR$- or $LL$-condition.
For example, the deterministic contextfree languages are the those generated by

\(^1\) Alternatively one could regard writing automata with a nondeterministic stack con-
tent as transducers which on two-way inputs (in form of their stack) from a regular
set generate outputs from a Macro or Lindenmayer Language
contextfree grammars subject to an $LR[1]$ condition and the languages generated by contextfree grammars subject to an $LL[k]$ condition form a hierarchy w.r.t. $k$ strictly contained in the family of deterministic contextfree languages ([Har78]). But still their wordproblem is as hard as in the $LR$-case ([Sud78]).

Also in the case of linear contextfree languages $LR$-conditions are less restrictive than $LL$-conditions and characterize determinism in the corresponding machine model ([HL93, IJR88]).

When looking for grammatical representations of one-way stack languages one should consider the abundance of variations of this model ([ESvL80]). Of particular interest in this connection are the nested stack languages of Aho ([Aho69]). They show close relations to stack languages. For instance, Beeri was able to show the equivalence of two-way stack automata and two-way nested stack automata ([Bee75]):

**Proposition 2.** $2\text{-NSA} = 2\text{-NNestedSA}$.

Aho found a grammatical characterization of Nested Stack languages in terms of Indexed Languages$^2$ ([Aho68]).

Rounds showed that one-way nested stack automata have a polynomially bounded running time ([Rou73]). Hence the indexed languages are contained in $NP$ and any deterministic restriction of them in $P$. It is known that already some strict subfamilies of the indexed languages have an $NP$-complete membership problem ([vL75]). As a corollary of Theorem 1 we get that the deterministic nested stack languages are in $P$ and can have an $P$-complete membership problem.

This note is closed with some remarks concerning grammatical characterizations of deterministic nested stack languages. It is not obvious how to restrict indexed grammars by an $LL$ or $LR$ condition. Without giving the details here the main idea is to modify slightly the way an index production is used in a derivation. It is then possible to see that the language $L$ of Theorem 1 is an indexed $LR[1]$-language. We conjecture that the indexed $LR[1]$-languages coincide with $1\text{-DNestedSA}$. The relation seems to be different at first sight iff we consider $LL[1]$ (see [PDS80, PDS84]): but also in this case it is possible to show

**Theorem 3 (Reinhardt).** Indexed $LL(1)$-Languages are $P$-complete.

It should be remarked that in this construction the use of erasing productions is crucial.

**References**


$^2$ In fact Aho first found (at least published) the grammars and then the automata