KYBERNETIKA — VOLUME 17 (1981), NUMBER 5

VALIDITY TEST FOR FLOYD'S OPERATOR-PRECEDENCE PARSING ALGORITHMS IS POLYNOMIAL IN TIME

PETER RUŽIČKA

A polynomial time algorithm is developed to decide the equivalence of an operator-precedence grammar with the underlying Floyd's operator-precedence parsing algorithm, a result of possible practical significance. As a consequence the necessary and sufficient condition for an operator-precedence grammar to be the valid grammatical characterization of the underlying Floyd's operator-precedence parsing algorithm is obtained.

1. INTRODUCTION

Recent work on the improvement of both the running time and the size of bottom-up parsing algorithms has been oriented in several directions. Much effort has been devoted to the development of optimizing transformations to reduce the size and/or the time of parsing algorithms. In addition modified techniques have been proposed to produce parsing algorithms of practical size. The research on the validity of shift-reduce parsing algorithms is also a contribution in this direction.

The shift-reduce operator-precedence parsing algorithm is conceptually very simple and hereby a very effective technique for syntactical analysis. It is sufficiently general to handle the parsing of a variety of the programming language constructs. However, the main impediment for using this technique is that Floyd's operator-precedence parsing algorithm might accept strings not in the language of the underlying operator-precedence grammar. This is a consequence of using different means in the definitions of both concepts. While operator-precedence grammars use non-terminal symbols as well as unique operator-precedence relations to derive valid strings, Floyd's parsing algorithms use only operator-precedence relations to recognize valid input strings. On the other hand, the great advantage of this method is the time and size efficiency because only those reductions are made that do not involve single productions and in practical cases a copy of the whole grammar

usually need not to be kept in order to know which reduction is to be made by Floyd's operator-precedence parsing algorithm.

Other parsing techniques are known in which it is necessary to overcome the same problem of invalid strings being recognized. Well known are canonical precedence parsing algorithms of Gray [5] or skeletal LR parsing algorithms of ElDjabri [3] and Demers [2], or Knuth's [7] top-down parsing algorithms with partial back-up used in McClure's TMG translator writing system. Knuth has proved that in the case of top-down parsing algorithms with partial back-up it is not algorithmically decidable whether a grammar generates the same language as it is accepted by the intended parsing algorithm. Demers presented an algorithm to decide whether a skeletal LR parsing algorithm accepts exactly the language of the underlying grammar. Demers further claimed that his validity test applies to skeletal canonical precedence parsing algorithms as well. We consider a similar problem for operator-precedence parsing algorithms and we solve it by presenting a decision test for the equivalence of an operator-precedence grammar and the underlying Floyd's operator-precedence parsing algorithm.

2. BACKGROUND

We present some basic definitions and terminology in the area of context-free languages and parsing theory (for more detailed information see i.e. [1]).

A context-free grammar G is a quadruple $\langle N, T, P, S\rangle$, where N is the finite set of nonterminal symbols, T is the finite set of terminal symbols, S is the start symbol from N, and $P \subseteq N \times V^*$ is a set of productions. Instead of (A, α) we write $A \to \alpha$. A single production is one of the form $A \to B$, where A and B are both nonterminal symbols. A grammar is backwards-deterministic if no two productions have the same right-hand side.

Unless specified we use Roman capitals $(A, B, C, \ldots, X, Y, Z)$ to denote nonterminal symbols, lower-case Roman letters at the beginning of the alphabet $(a, b, c, \ldots)$ to denote terminal symbols, lower-case Roman letters near the end of the alphabet $(x, y, z, w, \ldots)$ to denote terminal strings, lower-case Greek letters $(\alpha, \beta, \gamma, \ldots)$ to denote strings of terminal and nonterminal symbols. We use e for the empty string.

We write $\alpha \Rightarrow \beta$ in G if $\alpha = \alpha_1 A \alpha_3$, $\beta = \alpha_1 \alpha_2 \alpha_3$, and $A \to \alpha_2$ is a production in P. The relation $\Rightarrow^*$ is the reflexive and the transitive closure of $\Rightarrow$, and $\Rightarrow^+$ is the transitive closure of $\Rightarrow$. The context-free language for the grammar G is $L(G) = \{w \in T^* \mid S \Rightarrow^* w\}$. A derivation $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_k$ in G is called rightmost derivation (denoted as $\Rightarrow_R$) if at the i-th step $(1 \leqq i < k)$ of the derivation the rightmost nonterminal symbol of α_i is replaced according to some production of G to yield α_{i+1}.

Let $\mathcal{G} = \langle V, E\rangle$ be a graph with a set of vertices V and a set of edges E, and $f = (v_1, v_2)$ be an edge from E. We define a graph $\mathcal{G}' = \langle V \cup \{v\}, E'\rangle$ such that $E' = (E - \{f\}) \cup \{(v_1, v), (v, v_2)\}$, $v \notin V$. The graph $\mathcal{G}'$ is called an elementary

division of the graph $\mathcal{G}$. A graph $\hat{\mathcal{G}}$ is called the division of the graph $\mathcal{G}$ if there exists a sequence of graphs $\mathcal{G} = \mathcal{G}_0, \mathcal{G}_1, \ldots, \mathcal{G}_n = \hat{\mathcal{G}}$ such that $\mathcal{G}_{i+1}$ is an elementary division of $\mathcal{G}_i$ for $i \in \langle 0, n-1 \rangle$. Two graphs $\mathcal{G}$ and $\mathcal{G}'$ are called homeomorphic if there is a graph $\hat{\mathcal{G}}$ such that $\hat{\mathcal{G}}$ is a division of $\mathcal{G}$ and $\mathcal{G}'$. Two derivations are similar if the corresponding derivation trees are homeomorphic up to relabeling of interior nodes. We say that a grammar G is ambiguous if there is some string in $L(G)$ for which there are two distinct non-similar rightmost derivations. G is otherwise called unambiguous. We say that G_1 is structurally equivalent to G_2 if every derivation in G_1 is similar to some derivation in G_2 and vice versa. We say that G_1 is equivalent to G_2 if $L(G_1) = L(G_2)$.

An e-free grammar is a context-free grammar in which no right-hand side of production is e. An operator grammar is a context-free grammar in which no production has a pair of adjacent nonterminal symbols on its right-hand side. Operator-precedence relations $\lessdot$, $\doteq$, $\gtrdot$ are defined on T in the following manner:

For $a, b \in T$:

(i) $a \doteq b$ if there is $A \to \alpha aBb\beta$ in P for $B = e$ or $B \in N$.

(ii) $a \lessdot b$ if $A \to \alpha aB\beta$ in P and $B \Rightarrow^+ Cb\gamma$ for $C = e$ or $C \in N$.

(iii) $a \gtrdot b$ if $A \to \alpha Bb\beta$ in P and $B \Rightarrow^+ \gamma aC$ for $C = e$ or $C \in N$.

An operator-precedence grammar is an e-free operator grammar in which operator-precedence relations $\lessdot$, $\doteq$, $\gtrdot$ are disjoint.

Assume M to be operator-precedence relations over the terminal alphabet T and $R = \{(a, b) \mid a \doteq b \text{ is in } M\} \subseteq T \times T$. Then (a, b) from R can be graphically represented as an oriented edge from the node a to the node b. Thus, R can be viewed as a directed graph $\mathcal{G}_R$. A path in $\mathcal{G}_R$ is a sequence of consecutive nodes in $\mathcal{G}_R$. A cycle C is the path of the length greater than one in which from any node in C one can reach an arbitrary node in C. We say that M contains a $\doteq$-cycle if there is a cycle in $\mathcal{G}_R$.

Using the parsing stack and the precedence relations derived from some operator-precedence grammar, the operator-precedence parsing algorithm operates on the input string in the following way:

Initially the input contains $a_1 \ldots a_n \dashv$ and the parsing stack contains $\vdash$.

```
begin
  repeat
    if topstack = ⊢ and current-input-symbol = ⊣
    then ACCEPT
    else
      if topstack = a and current-input-symbol = b
      then
        select
```

```
(1)          a ⋖ b or a ≐ b : shift b from the input onto the stack;
(2)          a ⋗ b : repeat pop the stack and put this most recently popped
                                 symbol to X
(3)                        until the topstack ⋖ X;
                    otherwise : ERROR
                 end ⟨select⟩
              fi
           fi
        until forever
     end
```

In order to construct a parse tree, the operator-precedence parsing algorithm must create a node for each terminal symbol shifted onto the stack during the statement at line (1). When the loop of line (2) reduces by some production, the parsing algorithm creates a node whose children are the nodes corresponding to whatever is popped off the stack. After the loop at line (2) the parsing algorithm places on the stack a pointer to the node created. This means that some of the "symbols" popped by line (2) will be pointers to nodes. The comparison of line (3) continuous to be made between terminal symbols only; pointers are popped with no comparison being made.

Floyd's operator-precedence parsing algorithm $\mathscr{A}_M$ is an operator-precedence parsing algorithm driven by precedence relations M derived from some operator-precedence grammar.

3. VALIDITY TEST

It has been already remarked by Fischer [4] that operator-precedence grammars suffer one serious drawback, namely that though Floyd's shift-reduce operator-precedence parsing algorithm accepts all input strings of the operator-precedence grammar, there is no guarantee that such a parsing algorithm will also accept invalid input strings of that grammar. To illustrate this disadvantage a simple grammar G with the following set of productions can be chosen: $S_0 \to \vdash S \dashv$, $S \to BaB$, $B \to b$. We see that $L(G) = \{\vdash bab \dashv\}$. However, if one parses the string $\vdash b \dashv$, one finds that it holds $\vdash \lessdot b$ and $b \gtrdot \dashv$ and so the input string $\vdash b \dashv$ is reduced to $\vdash B \dashv$. This sentential form has $\vdash \doteq \dashv$ now and if no check is performed to ensure that both terminal and nonterminal symbols match the right-hand side of some production, $\vdash b \dashv$ will be accepted as a valid string of the language.

The construction **repeat** S **until** p means that statements S are repeated until the condition p is fulfilled (the case **until forever** is equal to **until false**) and the construction **select** $p_1 : S_1; \ldots;$ $p_n : S_n$; **otherwise** : S_{n+1} **end** means that the statement S_i for $1 \leqq i \leqq n$ is performed when the condition p_i is **true** and all p_j for $1 \leqq j \leqq i - 1$ are **false**. If all $p_1, \ldots, p_n$ are **false**, then the statement S_{n+1} is performed.

We present an algorithm to decide whether an operator-precedence parsing algorithm accepts exactly the language generated by the underlying grammar. The principal idea behind this decision algorithm is the grammatical characterization of the operator-precedence parsing algorithm and the reduction of our test to the equivalence problem for two backwards-deterministic operator-precedence grammars with the same operator-precedence relations. The latter problem is shown to be algorithmically decidable.

We begin with a technical assertion that a single production removal does not affect the operator-precedence property.

Lemma 3.1. Given an arbitrary operator-precedence grammar, one can find a structurally equivalent operator-precedence grammar without single production.

Proof. Let $G = \langle N, T, P, S\rangle$ be an operator-precedence grammar. Let $\hat{G} = \langle N, T, \hat{P}, S\rangle$, where $\hat{P} = \{A \to \alpha \mid \alpha \notin N$ and for some $B \in N$, $A \Rightarrow^* B$ in G and $B \to \alpha$ in $P\}$. It is readily verified that $L(G) = L(\hat{G})$ and that $\hat{G}$ is an operator-precedence grammar without single productions and that every derivation in G is similar to some derivation in $\hat{G}$ and vice versa. □

The key idea of our validity test is obtained in the following assertion.

Lemma 3.2. Given two backwards-deterministic operator-precedence grammars G_1 and G_2 with equal operator-precedence relations, there is an algorithm to decide whether it holds $L(G_1) = L(G_2)$.

Proof. Let $G_1 = \langle N_1, T, P_1, S_1\rangle$ and $G_2 = \langle N_2, T, P_2, S_2\rangle$ be two backwards-deterministic operator-precedence grammars with equal operator-precedence relations. We first show that $L(G_1) = L(G_2)$ if and only if G_1 is structurally equivalent to G_2. Then we show that structural equivalency of two context-free grammars is decidable.

1. Assume that $L(G_1) = L(G_2) = L$. Following Lemma 3.1 there are backwards-deterministic structurally equivalent operator-precedence grammars $G_1' = \langle N_1', T, P_1', S_1'\rangle$ and $G_2' = \langle N_2', T, P_2', S_2'\rangle$, respectively, without single productions. We show that an arbitrary input string $w = a_1 \dots a_n$ from L has (up to relabling of interior nodes) the same derivation tree according to G_1' as according to G_2'. We first note that

   Proposition 1. The string w has a unique derivation in G_1'.

   This is true because G_1' is a backwards-deterministic operator-precedence grammar without single productions. To see it in more detail we have to distinguish two cases. In the first case there are two various rightmost derivations of w in G_1' of the form

$$S_1' \Rightarrow_R^* \beta A u \Rightarrow_R \beta_1 B u_1 \Rightarrow_R^* a_1 \dots a_n$$
$$S_1' \Rightarrow_R^* \beta A u \Rightarrow_R \beta_2 C u_2 \Rightarrow_R^* a_1 \dots a_n$$

where $B \in N_1'$, $u, u_1, u_2 \in T^*$, $|u_2| > |u_1|$. In this case a contradiction with uniqueness of operator-precedence relations is obtained by the same argument as the one described in the proof of the Proposition 2 for the case $k = i$. In the second case there are two various rightmost derivations of w in G_1' of the form

$$S_1' \Rightarrow_R^* \beta A u \Rightarrow_R \beta_1 B u_1 \Rightarrow_R^* a_1 \dots a_n$$

$$S_1' \Rightarrow_R^* \beta A u \Rightarrow_R \beta_2 C u_1 \Rightarrow_R^* a_1 \dots a_n$$

where $B, C \in N_1'$, $B \neq C$, $u, u_1 \in T^*$. In this case a contradiction either with backwards-deterministic property or as in the previous case with uniqueness property of operator-precedence relations is obtained.

Furthermore, we prove

Proposition 2. The righmost derivation of w in G_1' is similar to the rightmost derivation of w in G_2'.

The proof of this proposition is by induction on the length h of derivations. The proposition is trivial for $h = 0$. Assume that the proposition holds for initial parts of derivations of the length shorter than h. We then prove the proposition for h (i.e. when initial parts of derivations of w in G_1' and in G_2' are of the length h). Indirectly assume that there are two rightmost derivations of the form

$$S_1' \Rightarrow_R^{(h-1)} \beta_1 A_1 a_i \dots a_n \Rightarrow_R \gamma_1 B_1 a_k \dots a_n \Rightarrow_R^* a_1 \dots a_n \tag{3.1}$$

$$S_2' \Rightarrow_R^{(h-1)} \beta_2 A_2 a_i \dots a_n \Rightarrow_R \gamma_2 B_2 a_s \dots a_n \Rightarrow_R^* a_1 \dots a_n \tag{3.2}$$

where $1 \leqq s < k \leqq i \leqq n$ and β_1 equals to β_2 up to the renaming of nonterminal symbols. In case $k < i$, from the derivation (3.1) it holds $a_{k-1} \gtrdot a_k$, but from the derivation (3.2) it must be $a_{k-1} \doteq a_k$, a contradiction with the assumption that G_1' and G_2' have equal operator-precedence relations.

In case $k = i$, the rightmost symbol of the string γ_1 must be the terminal symbol, because G_1' is the operator grammar. Let $\gamma_1 = \gamma a_t$ for some $t \in \langle 1, k-1)$, $\gamma \in (N_1' \cup T)^*$. Assume that in the derivation (3.1) the first occurrence of a_k will be in a rightmost sentential form α_1, the first occurence of a_t will be in α_2 and the first occurrence of a_{k-1} will be in α_3. Then the derivation (3.1) can be written either in the form

$$S_1' \Rightarrow_R^+ \alpha_1 \Rightarrow_R^* \alpha_2 \Rightarrow_R^+ \alpha_3 \Rightarrow_R^* a_1 \dots a_n$$

or in the form

$$S_1' \Rightarrow_R^+ \alpha_2 \Rightarrow_R^* \alpha_1 \Rightarrow_R^+ \alpha_3 \Rightarrow_R^* a_1 \dots a_n .$$

Assume that in the derivation (3.2) the first occurrence of a_k will be in a rightmost sentential form α_4, the first occurrence of a_{k-1} will be in α_5 and the first occurrence of a_t will be in α_6. Then the derivation (3.2) can be written in the form

$$S_2' \Rightarrow_R^+ \alpha_4 \Rightarrow_R^+ \alpha_5 \Rightarrow_R^* \alpha_6 \Rightarrow_R^* a_1 \dots a_n .$$

There must be some $p \in \langle t, k-1\rangle$ such that from (3.1) it holds $a_{p+1} \lessdot a_p$, but from (3.2) it holds either $a_{p+1} \stackrel{\circ}{=} a_p$ or $a_{p+1} \gtrdot a_p$, a contradiction with the assumption that G_1' and G_2' have equal operator-precedence relations. This ends the proof of the Proposition 2.

Hence, we have proved that if $L(G_1) = L(G_2)$, then G_1 is structurally equivalent to G_2.

2. If G_1 is structurally equivalent to G_2, then by the definition it holds $L(G_1) = L(G_2)$.

Now it is sufficient to construct an algorithm to decide whether G_1 is structurally equivalent to G_2. Paull and Unger [10] showed that it is decidable whether two context-free grammars are "structurally equivalent" in the sense that they generate the same strings, and that their derivation trees are the same except for labels in interior nodes. Independently, McNaughton [9] showed that equivalence is decidable for paranthesis grammars, which are grammars in which the right-hand side of each production is surrounded by a pair of parantheses, which do not appear elsewhere within any production. Two grammars are "structurally equivalent" in the sense of Paull and Unger if and only if the paranthesis grammars constructed from them are equivalent.

It is immediately seen that an algorithm due to Paull und Unger can be used to decide whether G_1' is structurally equivalent to G_2', where $G_1'(G_2')$ is an operator-precedence grammar without single productions constructed following Lemma 3.1. □

In order to characterize Floyd's operator-precedence parsing algorithm grammatically it is possible to apply only such constructions which preserve operator-precedence relations. We show how for the Floyd's parsing algorithm a grammar can be constructed which will generate the same language as it is accepted by Floyd's parsing algorithm and in what cases the operator-precedence grammar will be backwards-deterministic. We make the following claim.

Lemma 3.3. Given any Floyd's operator-precedence parsing algorithm $\mathscr{A}_M$ ruled by operator-precedence relations M without $\stackrel{\circ}{=}$-cycle, we can find an equivalent backwards-deterministic operator-precedence grammar G with the same operator-precedence relations M.

Proof. Let M be a set of disjoint precedence relations over the set T of terminal symbols not containing a special symbol #. We construct the grammar

$$\hat{G} = \langle N, T \cup \{\#\}, P, S_0\rangle\,, \quad \text{where } S_0 \text{ is the start symbol},$$

$$N = \{[aSb] \mid a, b \in T \cup \{\#\}\} \cup \{S_0\} \quad \text{is the set of nonterminal symbols},$$

$P = \{[aSb] \to [aSa_1]\, a_1\, [a_1Sa_2]\, a_2 \ldots a_k\, [a_kSb] \mid$ there exist $a, b \in T \cup \{\#\}$ such that in M it holds $a \lessdot a_1$, $a_k \gtrdot b$, $a_i \stackrel{\circ}{=} a_{i+1}$ for $a_i \in T$, $1 \leqq i < k\} \cup \{S_0 \to \to \# [\# S \#] \#\} \cup \{[aSb] \to e \mid$ there exist $a, b \in T \cup \{\#\}$ such that in M it holds $a \lessdot b$ or $a \stackrel{\circ}{=} b$ or $a \gtrdot b\}$

is the set of productions. $\hat{G}$ is not in the reduced and e-free form, but using the standard transformation of a context-free grammar into reduced e-free form one can obtain $\hat{G}$ to be in reduced e-free form. We can see that this transformation preserves operator-precedence relations and it does not violate the backwards-deterministic property. Hence, we can assume that $\hat{G} = \langle \hat{N}, \hat{T}, \hat{P}, \hat{S} \rangle$ is reduced and e-free, and backwards-deterministic operator-precedence grammar with precedence relations M. To prove the proposition it suffices to show that $L(\hat{G}) = \# L(\mathscr{A}_M) \#$.

(i) $\# L(\mathscr{A}_M) \# \subseteq L(\hat{G})$. Let $w \in L(\mathscr{A}_M)$ and let w is accepted by $\mathscr{A}_M$ in k reduction steps. It can be easily proved by the induction on the number of reduction steps done by $\mathscr{A}_M$ in accepting w that: "If in $\mathscr{A}_M$, the i-th reduction of w transforms $\alpha a u b \beta$ into the form $\alpha a b \beta$ for $u = u_1 u_2 \dots u_s$, then in $\hat{G}$, there exists a production of the form $[aSb] \to [aSu_1]^{\oplus} u_1 [u_1 S u_2]^{\oplus} u_2 \dots u_s [u_s S b]^{\oplus}$, where $[cSd]^{\oplus}$ means either e or $[cSd]$, such that there is a derivation of $\# w \#$ in $\hat{G}$ of the form $\hat{S} \Rightarrow_R^* \# \alpha a [aSb] b \beta \# \Rightarrow_R^{(i)} \# w \#, \beta \in \hat{T}^*$". Hence, it holds $\# w \# \in L(\hat{G})$.

(ii) $L(\hat{G}) \subseteq \# L(\mathscr{A}_M) \#$. Assume $\# w \# \in L(\hat{G})$. Then there is a derivation of the form $\hat{S} \Rightarrow^{(k)} \# w \#$ in $\hat{G}$ for some $k \geqq 1$. Again, it can be easily proved by the induction on the length of derivation of $\# w \#$ in $\hat{G}$ that "If the $(k - i)$-th derivation step of $\# w \#$ in $\hat{G}$ is of the form $\# \alpha a A b \beta \# \Rightarrow_R \# \alpha a B_1 u_1 B_2 u_2 \dots \dots B_s u_s B_{s+1} b \beta \#$, then $\mathscr{A}_M$ in the i-th step reduces $\alpha a B_1 u_1 B_2 \dots u_s B_{s+1} b \beta$, $\beta \in \hat{T}^*$, into the form $\alpha a b \beta$". Thus, $w \in L(\mathscr{A}_M)$. □

Now we are prepared to formulate the main result if this section.

Theorem 3.1. For any backwards-determinitic operator-precedence grammar G with operator-precedence relations M and for the Floyd's operator-precedence parsing algorithm $\mathscr{A}_M$ ruled by operator-precedence relations M there is an algorithm to decide whether it holds $L(G) = L(\mathscr{A}_M)$.

Proof. For any Floyd's operator-precedence parsing algorithm ruled by operator-precedence relations M containing $\doteq$-cycle there does not exist an equivalent operator-precedence grammar with equal operator-precedence relations. This follows from the following argument. Suppose that in M it holds $a_1 \doteq a_2 \doteq \dots \doteq a_n \doteq a_1$. Then $(a_1 \dots a_n)^k$, $k \geqq 1$, are substrings of some valid input strings in $L(\mathscr{A}_M)$. In order to derive substrings $(a_1 \dots a_n)^k$, $k \geqq 1$, of valid input strings in $L(G)$, it must hold either $a_n \gtrdot a_1$ or $a_n \lessdot a_1$ in M, a contradiction. Thus, if M contains $\doteq$-cycle, then $L(G) \neq L(\mathscr{A}_M)$.

If M does not contain $\doteq$-cycle, then using Lemma 3.3 we can find an equivalent backwards-deterministic operator-precedence grammar G_1. Using Lemma 3.2 given two backwards-determinitic operator-precedence grammars G and G_1 with the same operator-precedence relations, there is an algorithm to decide whether it holds $L(G) = L(G_1)$. Hence, given any backwards-deterministic operator-precedence grammar G with operator-precedence relation M it is algorithmically decidable whether it holds $L(G) = L(\mathscr{A}_M)$ or not, where $\mathscr{A}_M$ is the Floyd's operator-precedence parsing algorithm ruled by operator-precedence relations M. □

4. COMPLEXITY

Denote the size of a context-free grammar $G = \langle N, T, P, S\rangle$ by $|G| = \sum_{p \in P} (|\alpha_p| + 2)$, where p is a production of the form $A \to \alpha_p$ and $|\alpha_p|$ is the length of the string α_p. In order to determine the time complexity of the validity test for Floyd's operator-precedence parsing algorithms we have to consider the following steps and to analyze them according to the size of the grammar G.

Fact 1 (Lemma 3.1). An algorithm which transforms any operator-precedence grammar $G = \langle N, T, P, S\rangle$ into a structurally equivalent operator-precedence grammar without single productions works in time $O(|N| \,.\, |P| \,.\, |G|)$.

It is easy to verify the time bound $O(|N| \,.\, |P| \,.\, |G|)$ of the transformation, given in Lemma 3.1, on an input grammar G. We note that the set of nonterminal symbols of the resulting grammar is a subset of N and that the size of the resulting grammar can be asymptotically greater than the size of G. For example, if G is of the form $G = \langle \{S, A_1, \ldots, A_n, B_1, \ldots, B_n\}, \{a_1, \ldots, a_n, b_1, \ldots, b_m\}, \{S \to a_i A_i,\ A_i \to B,\ B \to b_1 \ldots b_m \mid 1 \leqq i \leqq n\}, S\rangle$, then $|G| = O(n + m)$, but the size of the resulting grammar $\hat{G} = \langle \{S, A_1, \ldots, A_n\}, \{a_1, \ldots, a_n, b_1, \ldots, b_m\}, \{S \to a_i A_i,\ A_i \to b_1 \ldots b_m \mid 1 \leqq i \leqq n\}, S\rangle$ is $O(n \,.\, m)$. But the size of the resulting grammar is bounded by $|N| \,.\, |P| \,.\, \max_{p \in P} (|\alpha_p| + 2)$, which gives the bound of Fact 1.

Fact 2 (Lemma 3.2). A test for structural equivalence of two reduced backwards-deterministic operator-precedence grammars $G_1 = \langle N_1, T, P_1, S_1\rangle$ and $G_2 = \langle N_2, T, P_2, S_2\rangle$ with equal precedence relations and without single productions can be performed in time $O(|N| \,.\, |G|^2)$, where $|G| = \max(|G_1|, |G_2|)$ and $|N| = \max(|N_1|, |N_2|)$.

The idea of the test is to group those productions (in both grammars separately) which have the same "terminal pattern" on their right-hand sides together with nonterminal symbols in the same positions on the right-hand sides (i.e. two productions of the form $A \to A_1 a_1 A_2 \ldots A_k a_k A_{k+1}$ and $B \to B_1 b_1 B_2 \ldots B_k b_k B_{k+1}$ of a grammar belong to the same group if and only if $a_i = b_i$ for $1 \leqq i \leqq k$ and for $1 \leqq i \leqq k + 1$ both A_i, B_i are either nonterminal symbols or the empty string e). Then a correspondence between nonterminal symbols of both grammars is determined by their positions on the left-hand and/or the right-hand sides of productions in the corresponding groups of productions. For example, consider grammars $G_1 = \langle \{S_1, A\}, \{a, b\}, \{S_1 \to aAb,\ A \to aAb,\ A \to ab\}, S_1\rangle$ and $G_2 = \langle \{S_2\}, \{a, b\}, \{S_2 \to aS_2b,\ S_2 \to ab\}, S_2\rangle$. According to the right-hand side terminal patterns we obtain two groups $\{S_1 \to aAb,\ A \to aAb\}$ and $\{A \to ab\}$ for G_1 and two groups $\{S_2 \to aS_2b\}$ and $\{S_2 \to ab\}$ for G_2. From corresponding groups $\{A \to ab\}$ and $\{S_2 \to ab\}$ we obtain the correspondence $A \leftrightarrow S_2$. Moreover, from corresponding groups $\{S_1 \to aAb,\ A \to aAb\}$ and $\{S_2 \to aS_2b\}$ we obtain $S_1 \leftrightarrow S_2$

and $A \leftrightarrow S_2$. So G_1 and G_2 are structurally equivalent. The time bound follows from the fact that it requires time $O(|G|^2)$ to determine the correspondence between groups of G_1 and G_2.

Fact 3. A simple test determining whether a set of operator-precedence relations contains a $\overset{\circ}{=}$-cycle or not can be performed in time $O(|T|^3)$.

The complexity bound follows from the fact that the presence of a $\overset{\circ}{=}$-cycle can be determined by the transitive and reflexive closure of the incident matrix in which the entry (a, b) equals to 1 if and only if $a \overset{\circ}{=} b$ holds.

The valid grammatical description of the Floyd's parsing algorithm described in Lemma 3.3 is of exponential size. To determine the grammatical description of Floyd's parsing algorithm $\mathscr{A}$ efficiently one can use backwards-deterministic operator-precedence grammar G without single productions. The "trick" is to "extend" G to cover $L(\mathscr{A})$ (proving $L(G) \neq L(\mathscr{A})$) or to show that it is not possible to "extend" G (proving $L(G) = L(\mathscr{A})$). We first try to "extend" G. We determine the number of different path from a to b, $a, b \in T$, through $\overset{\circ}{=}$ relations. This can be determined in time $O(|T|^3)$. Consider a pair (a, b) for which there exists at least one $\overset{\circ}{=}$-path and for which there is a valid sentential form in G of the form $\alpha cd\beta$, where $c \lessdot a$, $b \gtrdot d$. If the number of different paths for a pair (a, b) differs from the number of productions in G of the form $A \to Ba\alpha bC$, $B, C \in N \cup \{e\}$, where α contains different terminal patterns, then we have to add new productions to G in order to cover $L(\mathscr{A})$, thus $L(G) \neq L(\mathscr{A})$. If the number of different paths is equal to the number of productions of the form $A \to Ba\alpha bC$, then we try for some productions in G to add new productions to G with the same terminal pattern. For each nonterminal symbol D we determine quadruples of terminal symbols (a, b, c, d) such that $\gamma_1 aDb\gamma_2$ is a valid rightmost derivation in G and it holds $aDb \Rightarrow_R^* ac\gamma_3 db$. This can be computed in time $O(|T|^4 \cdot |N|)$ using tables of $\lessdot$, $\overset{\circ}{=}$, $\gtrdot$ relations and incidence matrices in which (A, B) equals to 1 if and only if $A \to \alpha B\beta$. Now we try to insert a nonterminal symbol D on the right-hand side of each production or we try to replace some nonterminal symbol on the right-hand side of some production by another nonterminal symbol D, where (a, b, c, d) is a quadruple in the list of D, such that D is surrounded by terminal symbols a, b in this production. If we succeed, then $L(G) \neq L(\mathscr{A})$. Otherwise $L(G) = L(\mathscr{A})$. The latter case can be performed in time $O(|N| \cdot |T|^4 \cdot |G|)$. Hence, we obtained

Fact 4. A test determining whether G is the valid grammatical characterization for the underlying Floyd's operator-precedence parsing algorithm $\mathscr{A}$ can be determined in time $O(|N| \cdot |T|^4 \cdot |G|)$, where G is a reduced backwards-deterministic operator-precedence grammar without single productions.

Fact 5. A simple transformation which reduces a backwards-deterministic operator-precedence grammar G can be performed in time $O(|G|^3)$.

This transformation uses the transitive and reflexive closure of incidence matrices in which (A, B) equals to 1 if and only if $A \to \alpha B\beta$, $B \in N \cup T$, $\alpha, \beta \in (N \cup T)^*$ and (A, B) equals to 1 if and only if $A \to \alpha B\beta$, $B \in N$, $\alpha, \beta \in (N \cup T)^*$ or $A \to \alpha B\beta$, $\alpha, \beta \in T^*$, $B \in T$.

Theorem 4.1. A decision, whether an operator-precedence grammar G is equivalent to the underlying Floyd's operator-precedence parsing algorithm, can be performed in time polynomial to the size of G.

4. CONCLUSIONS

Operator-precedence grammars are suitable for specifying a variety of programming language constructs using an information about the precedence and associativity of operators. However, operator-precedence parsing algorithms possess a curious property that one can accept inputs that are not in the language of the underlying grammars. In the preliminary version of this paper [11] we have answered natural questions concerning two classes of languages which are definable using the operator-precedence grammars and the Floyd's operator-precedence parsing algorithm. In this paper we concentrated on proving the decidability of the equivalence problem for these two models. The latter result can be reformulated as the necessary and sufficient condition for an operator-precedence grammar to be valid grammatical characterization for Floyd's operator-precedence parsing algorithm. This result solves the problem stated by Levy [8]. We have not yet looked at a modification of operator-precedence relations in order to obtain a valid characterization of Floyd's parsing algorithms. A partial solution of this problem has been obtained by Henderson and Levy [6] by defining extended operator-precedence relations.

(Received February 4, 1981.)

REFERENCES

[1] A. V. Aho, J. D. Ullman: The Theory of Parsing, Translation, and Compiling. Vol. I: Parsing. Prentice-Hall, 1972.

[2] A. J. Demers: Skeletal LR parsing. 15th Annual Symposium on Switching and Automata Theory 1974, 185—198.

[3] N. El Djabri: Extending the LR Parsing Techniques to Some Non-LR Grammars. TR-121, Princeton University, New Jersey 1973.

[4] M. J. Fischer: Some properties of precedence languages. 1st Annual ACM Symposium on Theory of Computing 1969, 181—190.

[5] J. N. Gray: Precedence Parsers for Programming Languages. Ph. D. Thesis, Department of Computer Science, University of California, Berkeley 1969.

[6] D. S. Henderson, M. R. Levy: An extended operator-precedence parsing algorithm. Comput. J. *19* (1976), 3, 229—233.

[7] D. Knuth: Top down syntax analysis. Acta Informatica *1* (1971), 2, 79—110.
[8] M. R. Levy: Complete operator precedence. Information Processing Lett. *4* (1975), 2, 38—40.
[9] R. Mc Naughton: Paranthesis grammars. J. Assoc. Comput. Mach. *14* (1967), 3, 490—500.
[10] M. C. Paull, S. H. Unger: Structural equivalence of context-free grammars. J. Comput. System Sci. *2* (1968), 1, 427—463.
[11] P. Ružička: Validity test for Floyd's operator-precedence parsing algorithms. In: Mathematical Foundations of Computer Science (J. Bečvář, ed.), Lecture Notes in Computer Science 74. Springer-Verlag, Berlin 1979, 415—424.

RNDr. Peter Ružička, Výskumné výpočtové stredisko (Computing Research Centre), Dúbravská cesta 3, 885 31 Bratislava. Czechoslovakia.