Early error detection in syntax-driven parsers

by A. V. Moura

The early error detection capabilities of syntax-driven parsers are studied. The classes of weak precedence and simple mixed-strategy precedence parsers are chosen as the object of study. Very similar techniques could be used to obtain related results for other classes of syntax-driven parsers. We investigate whether the correct-prefix and the viable-prefix properties can be enforced within these classes: A negative result is obtained for the first class and a positive one for the second. Moreover, for the simple mixed-strategy class the relationship between early error detection and parser size is studied. Some lower bounds on the parser size are proven for simple mixed-strategy precedence parsers that have the viable-prefix property.

1. Introduction

In the process of translating from a programming language source code into machine code, the ability to detect errors in the source code as soon as possible is particularly attractive. Such an ability can considerably improve the adequacy of error messages that are issued to the user. It can also be of great value to any process that attempts some form of error correction. Here, we investigate the error detection capabilities of some syntax-driven translation mechanisms.

Early error detection in connection with a parsing algorithm that scans the input string from left to right can be expressed naturally by requiring that the parser shall not read past the point of an error. More precisely, let L be the language being analyzed and assume that the parser has read in a prefix, say x, of the input string. Then there must exist a continuation, z, such that xz (which might be different from the string originally given as input) is in L. If that is the case, we say that the parser satisfies the correct-prefix property.

Most bottom-up parsing algorithms operate in a shift-reduce fashion. That is, at any moment, either the next symbol is shifted from the input into the stack, or a few of the topmost symbols on the stack are reduced to a new symbol. The LR parsers [1] and the precedence parsers [2–5] are particular examples of shift-reduce parsers. It is known that the LR parsers and, more strongly, those parsers obtained by the characteristic parsing technique [6, 7], all enjoy the correct-prefix property. Precedence parsers do not, in general, obey the correct-prefix property. For the class of shift-reduce parsers, however, the correct-prefix property does not properly express the fact that the parser will stop at the *earliest* possible moment upon encountering an error. It says that the parser will not *shift* new symbols, but does not prevent it from further *reducing* the stack. The LALR [8] and SLR [9] variants of the LR-style parsing technique all display this behavior: If the next input symbol causes an error it will not be shifted, but the parser can perform an arbitrarily large number of reductions before coming to a halt and announcing the error. This is not the case with "pure" LR parsers: Upon encountering the first error, they will stop without making any additional shift or reduce moves. On the other hand, in most cases, LR parsers incorporate much larger parsing tables in comparison with precedence parsers or any of their SLR or LALR variants. One interesting fact that will be established later is that, in

general, we *must* resort to much larger parsing tables if very early error detection is to be enforced.

In the rest of this paper we concentrate on precedence parsers. It should be noted, however, that the intuition developed for precedence parsers can easily be transported to a study of the SLR and LALR classes as well. In fact, very similar results could be obtained for these classes using techniques that are close to the ones we develop in the sequel. Precedence parsers are chosen as our specific object of study based, in part, on the fact that it is possible to obtain, in practice, very simple, efficient, and compact parsing algorithms within this class.

The stack contents of a precedence parser is always a string of variables from the underlying grammar. In such cases, we can strengthen the correct-prefix property and capture the notion of "stopping at the earliest possible moment": We require the stack to always be a prefix of some right sentential form of the underlying grammar. When the stack has this property, we say that the parser has the viable-prefix property [10]. To be specific, let xy be the input and assume that the parser has read x up to this moment. Let the stack be α, a prefix of a right-sentential form like $\alpha\omega$. This means that from α we can derive x. But, if the grammar is reduced, from ω we can also derive some string of terminals, say z, in such a way that xz is in the language. Hence, at any moment during the parse there is always a "correct suffix," namely z, that could replace the yet-to-be-read part of the input y and drive the parse to a successful termination from this point on. It is clear that we could not have stopped earlier. If we had, the parser would not be operating properly since a correct input, namely xz, would lead to an error. It is interesting to note that "pure" LR parsers behave in a similar way: From any point during the parse, there is always a way to drive it to a successful conclusion, provided one can replace the yet-to-be-read part of the input string. The LR items and the LR table ensure that what has been shifted and reduced so far behaves like a "viable prefix" of the underlying grammar.

In the sequel we investigate and expose the nature of grammatical transformations that can be used to convert a grammar into an equivalent grammar whose precedence parser can be guaranteed to obey the viable-prefix property. We pay special attention to the structural transformations the original grammar will undergo [11, 12] as well as to the impact such transformations will have on the size [13] of the newly produced grammars and corresponding parsers. The transformations of interest, most certainly, will have to be of such a nature as to preserve not only the original source language syntax, but also the original semantic processes involved in the translation mechanism. In other words, we must be able to apply the transformations in a way that is totally transparent to the other components of the translation algorithm and, hence, in a way that will not disturb their operation.

The paper is organized as follows. This section continues by introducing these concepts rigorously. The next section investigates the class of weak precedence grammars. The third section considers the class of simple mixed-strategy precedence grammars. In the fourth section the space cost of the suggested transformation is studied. The last section contains some concluding remarks. Although we state our results rigorously, proofs are omitted. This makes the material more readable. All important intermediate results, however, are stated. We refer the reader to [14], where the details of proofs can be found.

We assume that the reader is conversant with the standard notation used in the theory of formal languages. Any undefined terms may be found in [13, 15, 16]. For any relation α, α^* represents its reflexive and transitive closure, whereas α^+ represents the transitive closure of α. The null string is denoted by λ. For all $\alpha \in V^+$, the first and last symbols of α are denoted by $1{:}\alpha$ and $\alpha{:}1$, respectively. For any language L we write L^* for the Kleene star operator on L and L^+ for the set $L^*\backslash\{\lambda\}$. A context-free grammar (grammar for short) is a system $G = (V, T, P, S)$, where V is the set of all variables, T is the set of terminal symbols, S is the start symbol, and P is the set of productions of G. The set of nonterminals of G is denoted by $N = V\backslash T$. Derivations and rightmost derivations in G are represented by $\Rightarrow$ and by $\Rightarrow_r$, respectively. The language generated by G is the set $L(G) = \{x \in T^* \mid S \Rightarrow^* x\}$. G is said to be reduced if for all $A \in N$ there is an $x \in T$ and there are $u, z \in V^*$ such that $S \Rightarrow^* uAz \Rightarrow^* x$. Next, the concept of a viable prefix is made precise.

Definition 1 Let $G = (V, T, P, S)$ be a grammar. A string $\alpha \in V^*$ is a *viable prefix* of G if and only if

1. $\alpha = S$ or
2. For some $x, y \in V^*$; $z \in T^*$ we have
   - $S \Rightarrow_r^* xAz \Rightarrow xyz$ and
   - α is a prefix of xy.

All the viable prefixes of G are collected in the set $VP(G)$.

Notation For any grammar $G = (V, T, P, S)$ we put $V' = V \cup \{\perp\}$ and $T' = T \cup \{\perp\}$, where $\perp$ is a new symbol.

The following definition deals with precedence relations.

Definition 2 Let $G = (V, T, P, S)$ be a grammar. Define $< \subset V' \times V$ and $> \subset V \times T'$ as follows, where $A, B, Z \in V$ and $x, y, w, z \in V^*$:

1. For all $X, Y \in V$,
   - $X < Y$ if and only if $A \rightarrow xXZw \in P$ and $Z \Rightarrow^* Yy$,
   - $\perp < X$ if and only if $S \Rightarrow^* Xx$.
2. For all $X \in V$ and all $a \in T$,
   - $X > a$ if and only if $Z \rightarrow xABz \in P$, $A \Rightarrow^+ yX$, and $B \Rightarrow^* aw$,
   - $X > \perp$ if and only if $S \Rightarrow^+ xX$.

Remarks

1. We preferred to define the relation $>$ over $V \times T$ as in [15] and not over $V \times V$, as was originally done in [5].
2. It is customary to adopt "end markers" for the input string and the stack. We adopted $\perp$ as our standard end marker. Moreover, the precedence relations were defined directly over the "extended" alphabets V' and T'.

Intuitively, $<$ represents a "stackability" condition: Whenever $X < a$, and X is the top symbol on the stack and a is the next input symbol, the parser will shift a onto the top of the stack. The "reducibility" condition is represented by $>$; i.e., whenever $X > a$, X and a as before, then we must select a production and perform the corresponding reduction.

Definition 3 Let $G = (V, T, P, S)$ be a grammar. We say that G is a *precedence grammar* if and only if the relations $<$ and $>$ are disjoint and we do not have $S \Rightarrow^+ S$ in G.

Usually, three precedence relations are introduced [2, 3, 15, 16]. Since we do not consider the class of simple precedence grammars, the three precedence relations can be combined as indicated in Definition 2. We refrain now from defining the classes of precedence grammars in which we are interested. This is done in later sections as needed. We now introduce the parsers formally.

Definition 4 Let $G = (V, T, P, S)$ be a grammar. A *shift-reduce parser* associated with G is a system $Q = (\perp, \vdash_s, \vdash_r)$. The symbol $\perp$, not in V, is a marker for Q. $\vdash_s, \vdash_r$ are respectively the *shift* and *reduce relations* of Q. $\vdash_s$ and $\vdash_r$ are defined as binary relations over $\perp V^* \times T^* \perp \times P^*$, the set of all configurations of Q, and must satisfy the following conditions:

1. $(t, x, \rho) \vdash_s (u, y, \omega)$ if and only if
   - $t{:}1 < 1{:}x$ and
   - $u = ta$, $x = ay$, and $\omega = \rho$, where $a \in T$.
2. $(t, x, \rho) \vdash_r (u, y, \omega)$ must imply
   - $t{:}1 > 1{:}x$ and
   - There is a $p = A \rightarrow v \in P$ such that $t = zv$, $u = zA$, $x = y$, and $\omega = \rho p$.

We further define the move relation of Q as $\vdash = \vdash_s \cup \vdash_r$.

Note that, whereas Part 1 above completely specifies the shift relation $\vdash_s$, that is not the case for Part 2 and the reduce relation $\vdash_r$. Note also that an improved mechanism for error detection could be obtained by requiring further that $z < A$ in Part 2 [17, 18]. This amounts to preventing A from being shifted onto the stack if the "stackability" condition does not hold between the symbol that was uncovered in the stack and the left-hand side of the production used in the reduction. Instead of requiring this extra condition, we maintain the usual definition and comment on this point as appropriate.

Definition 5 Let $G = (V, T, P, S)$ be a grammar and let $Q = (\perp, \vdash_s, \vdash_r)$ be a parser associated with G. We say that Q is *deterministic* if and only if $\vdash$ is a partial function. We say that Q is *valid* for G if and only if $(\perp, x\perp, \lambda) \vdash^* (\perp S, \perp, \rho)$ implies that the transpose of ρ is a right parse for x and vice versa, for all $x \in T^*$ and all $\rho \in P^*$.

It is clear that $\vdash_s$ is always a partial function. In the light of Definition 2 it is also clear that the domains of $\vdash_s$ and $\vdash_r$ are disjoint when G is a precedence grammar. Thus, in order for $\vdash$ to be a partial function, it remains to complete the definition of $\vdash_r$, ensuring that it is also a partial function. That is, we have to guarantee that there are no reduce-reduce conflicts. This is accomplished by turning Part 2 of Definition 4 into an if-and-only-if condition. That is, we have to adopt one of the parsing strategies used in connection with precedence grammars. The last definition in this section introduces the viable-prefix and correct-prefix properties.

Definition 6 Let $G = (V, T, P, S)$ be a grammar and let $Q = (\perp, \vdash_s, \vdash_r)$ be a parser for G. Let $x, y \in T^*$; $\alpha \in V^*$, and $\rho \in P^*$. We say that Q has

1. The *correct-prefix property* if and only if $(\perp, xy\perp, \lambda) \vdash^* (\perp\alpha, y\perp, \rho)$ implies that there is some $z \in T^*$ such that xz is in the language generated by G.
2. The *viable-prefix property* if and only if $(\perp, xy\perp, \lambda) \vdash^* (\perp\alpha, y\perp, \rho)$ implies that α is a viable prefix of G.

2. Weak precedence

In this section we aim at establishing a negative result for the class of weak precedence grammars. It is shown that very early error detection cannot, in general, be attained by precedence parsers that use the weak precedence technique [4] to break reduce-reduce conflicts.

Definition 7 Let $G = (V, T, P, S)$ be a grammar. G is a *weak precedence* (WP, for short) grammar if and only if

1. G is a precedence grammar.
2. For all $A \rightarrow x$ and $B \rightarrow x$ in P, we must have $A = B$.
3. For all $A \rightarrow xXy$ and all $B \rightarrow y$ in P, $X < B$ does not hold.
4. G has no null rules.

Observe that Conditions 2 and 3 above are "static" in the sense that they depend only on P and not on the dynamic behavior of the stack. As will be seen later, this point seems to be crucial in looking for parsers that preserve the viable-prefix property. In some texts G is also called an uniquely invertible precedence grammar.

Definition 8 Let $G = (V, T, P, S)$ be a grammar and let $Q = (\perp, \vdash_s, \vdash_r)$ be a parser associated with G. We can call Q a *WP parser* associated with G if and only if $\vdash_r$ satisfies

$(zv, x, \rho) \vdash_r (zA, x, \rho p)$ if and only if

1. $(zv):1 > 1:x$ and
2. $p = A \rightarrow v \in P$ is such that $|v| = \max\{|u|:B \rightarrow u \in P$ and u is a suffix of $zv\}$.

Observe that we are simply completing Definition 4 for the reduce relation $\vdash_r$. When G is a WP grammar, its WP parser is valid.

Theorem 9 Let G be a WP grammar and let Q be a WP parser associated with G. Then G is unambiguous and Q is deterministic and valid for G.

Now we investigate the correct-prefix and viable-prefix properties in conjunction with WP grammars and WP parsers. It is easy to see that WP parsers do not, in general, preserve these properties.

Example 10 Consider the grammar $G = (\{S, A, a, b, c\}, \{a, b, c\}, P, S)$, where P is given by the productions below:

$S \rightarrow a, \quad S \rightarrow bA,$

$A \rightarrow aAc, \quad A \rightarrow ac.$

Clearly, $L(G) = \{a\} \cup \{ba^nc^n:n \geq 1\}$. It is easily seen that G is a WP grammar and that $\perp < a$ and $a < c$. Hence, the WP parser for G would yield

$(\perp, ac\perp, \lambda) \vdash_s (\perp a, c\perp, \lambda) \vdash_s (\perp ac, \perp, \lambda),$

violating the correct-prefix property. □

The fact that the WP parser for G in the previous example does not have the correct-prefix property is not accidental: *Any* WP parser that correctly analyzes $L(G)$ will violate the correct-prefix property.

Theorem 11 Let $G = (V, T, P, S)$ be a WP grammar such that $L(G) = \{a\} \cup \{ba^nc^n:n \geq 1\}$. Let $Q = (\perp, \vdash_s, \vdash_r)$ be a WP parser for G. Then Q violates the correct-prefix property.

The viable-prefix property, of course, is also violated by any WP parser whose language includes $\{a\} \cup \{ba^nc^n:n \geq 1\}$. It is interesting to note that only Parts 2 and 4 of Definition 7, namely, the unique invertibility of G and the absence of null rules, are important to prove Theorem 11. In fact, if we removed Part 3 from Definition 7 we would still get valid parsers, although they might not be deterministic. In this case, the result would still be valid in the sense that some computation of the new (possibly nondeterministic) parser would still violate the correct-prefix property. From this discussion we may also conclude that improving Part 2 in Definition 8 by requiring further that $z:1 < A$ would not help as far as the correct-prefix property is concerned.

Note also that the class of languages generated by WP grammars is a proper subset of the family of all deterministic context-free languages [15, 16]. In the next section we relax the unique invertibility condition, thereby obtaining a class of precedence grammars whose members are able to generate any deterministic context-free language. As will be seen, the situation changes abruptly: It will always be possible to enforce the viable-prefix property within this class.

3. Simple mixed-strategy precedence

The simple mixed-strategy precedence grammars were introduced in [2] as a restriction to the mixed-strategy class considered in [19]. As can be seen in the definition below, unique invertibility is relaxed. The reduce-reduce conflicts caused by noninvertible productions are now broken by imposing a "dynamic" restriction on the stack, as dictated by Part 2 in that definition.

It is already known that grammatical transformations (to generate equivalent grammars whose parsers obey the correct-prefix property) do exist for this class of grammars [20]. In this section we describe, and prove the correctness of, a new transformation that has two added values: It is much simpler and it permits a very simple cover morphism to be defined from the new grammar into the original one. This last property guarantees that the transformed grammar can be used to replace the original one in the translation process without having to modify any of its other existing functions.

Definition 12 Let $G = (V, T, P, S)$ be a grammar. G is a *simple mixed-strategy precedence* (SMSP, for short) grammar if and only if

1. G is a precedence grammar.
2. For all $A \rightarrow x$ and $B \rightarrow x$ in P with $A \neq B$, we do not simultaneously have $Z < A$ and $Z < B$, for any $Z \in V$.
3. For all $X \in V$ and all $A \rightarrow xXy$, $B \rightarrow y$ in P, $X < B$ does not hold.
4. G has no null rules.

The corresponding parsers are now defined.

Definition 13 Let $G = (V, T, P, S)$ be a grammar and let $Q = (\perp, \vdash_s, \vdash_r)$ be a parser associated with G. We call Q a *SMSP parser* for G if and only if $\vdash_r$ satisfies

$(zv, x, \rho) \vdash_r (zA, x, \rho p)$ if and only if

1. $(zv):1 > 1:x$ and
2. $p = A \rightarrow v \in P$ is such that
   a. $|v| = \max\{|u|:B \rightarrow u \in P$ and u is a suffix of $zv\}$.
   b. $z:1 < A$.

Observe that the "stackability" condition alluded to earlier is now present as item 2(b) of the definition above. As was the case before, a SMSP parser is always valid.

Theorem 14 Let G be a SMSP grammar and let Q be a SMSP parser associated with G. Then G is unambiguous and Q is deterministic and valid for G.

We now turn to the correct-prefix and viable-prefix properties in SMSP parsers. From Example 10 it is already clear that not all SMSP parsers obey these properties. A more interesting case is given by the next example.

Example 15 Consider the grammar $G = (\{S, A, Z, a, b, c\}, \{a, b, c\}, P, S)$, where P is given by the productions below:

$$S \to a, \qquad S \to bA,$$

$$A \to ZAc, \qquad A \to Zc,$$

$$Z \to a.$$

It is easily seen that G is a SMSP grammar. Moreover, it can be checked that the SMSP parser associated with G does have the viable-prefix property. Note that $L(G) = \{a\} \cup \{ba^nc^n : n \geq 1\}$ and compare to Theorem 11. □

We now attack the problem of transforming any SMSP grammar into an equivalent SMSP grammar whose SMSP parser has the viable-prefix property. We want the stack contents, at any moment, to represent a viable prefix of the underlying grammar. The basic idea will be to introduce new nonterminal symbols in the form of pairs (x, A), where x represents all the stack contents that lay before A on the stack. Equivalently, we must have $S \Rightarrow_r^* xAy$ in the grammar for some y and so, clearly, x is a viable prefix of the grammar. The problem with such an approach, of course, is that there might be infinitely many such strings x, which would render the idea useless. Therefore, instead of using x as the first component in these pairs, we shall define an equivalence relation on the set of all viable prefixes, and use the equivalence classes as first components in the new nonterminals to be created. Hence, the new nonterminals will have the form (C, Z), where Z is a variable and C is a subset of V^*. The intuition here is that for any x in C there is a parse where x is the string appearing before Z on the stack; that is, for all x in C we have $S \Rightarrow_r xZu$, for some $u \in T^*$. Further, assume that $Z \to Z_1 \cdots Z_n$ is a production. Then we get $S \Rightarrow_r^* xZ_1 \cdots Z_n u$ and so the extended string $xZ_1 \cdots Z_{j-1}$ will appear before Z_j in the stack during some parse. Note that, if y is another string in C, then the extension $yZ_1 \cdots Z_{j-1}$ will also represent the stack contents before Z_j in some parse. We would like to say that two arbitrary viable prefixes x and y are "equivalent" exactly when they have the same such extensions. The class C, in the nonterminal (C, Z), will be a group of such "equivalent" prefixes.

It turns out that having information about the class, C, of elements that may precede Z in the stack, rather than knowing precisely which element of C one has at any moment before Z, is sufficient to guarantee the viable-prefix property. Another observation is that the parser will never stack more than m symbols in a row, where m is the maximum length of the right-hand side of a production in the grammar. Therefore, we need only consider extensions up to m symbols when determining which prefixes are "equivalent." In fact, since a new nonterminal (C, Z) will indicate that xZ is a viable prefix, x in C, we can restrict the extensions to $m - 1$ symbols, for Z will always be part of the extension. These are the ideas behind the sets $R(x)$ and the relation $\phi(G)$ that we now make precise.

Notation Let $M(G) = \max\{|x| : A \to x \text{ is in } P\}$ and for all $x \in VP(G)$, let $R(x) = \{z \in V^* : xz \in VP(G) \text{ and } 1 \leq |z| \leq M(G) - 1\}$.

Definition 16 Let $G = (V, T, P, S)$ be a grammar. Define the relation $\phi(G) \subset VP(G) \times VP(G)$ such that $x\ \phi(G)\ y$ if and only if $R(x) = R(y)$.

Assuming always that P is nonempty and G is reduced, it follows that if $M(G) = 0$, then $S \to \lambda$ is the only production in P. Under the same circumstances, if $M(G) = 1$, then the productions must be in the form $A \to Z$, where $Z \in V$, or $A \to \lambda$. In any case, it is immediate that the SMSP parser for G has the viable-prefix property. Therefore, from now on we assume $M(G) \geq 2$.

The next task is to verify that, indeed, $\phi(G)$ is an equivalence relation. Also, $\phi(G)$ must have a finite index. Otherwise, we would end up with an infinite number of nonterminals in the "new grammar." Moreover, we also need a property of right invariance from $\phi(G)$. To see this, note that from a new nonterminal (C, Z) and a production $Z \to Z_1 \cdots Z_n$ we would like to derive a new production in the form $(C, Z) \to (C_1, Z_1) \cdots (C_n, Z_n)$ where each C_j is an equivalence class. More precisely, C_j should be the class which contains the extension $xZ_1 \cdots Z_{j-1}$, where x is in C. Clearly, the definition of C_j must not depend on any particular choice for x. Therefore, we must ensure that $xZ_1 \cdots Z_{j-1}$ and $yZ_1 \cdots Z_{j-1}$ are equivalent, for any x and y in C. The next result guarantees that $\phi(G)$ has the desired properties.

Theorem 17 Let $G = (V, T, P, S)$ be a reduced grammar. Then $\phi(G)$ is a right-invariant equivalence relation of finite index.

Notation The equivalence class of x under $\phi(G)$ is denoted by $[x]$.

We are now in a position to present the grammatical transformation. Recall that λ is always a viable prefix, for we are assuming that P is not empty.

Algorithm 18
INPUT: a grammar $G = (V, T, P, S)$
OUTPUT: a grammar $G' = (V', T, P', S')$

1. Let NEW := $\{([\lambda], S)\}$; $V' :=$ NEW; $P' := \emptyset$.
2. While NEW $\neq \emptyset$ do
   a. Let $(C, Z) \in$ NEW,
   b. Remove (C, Z) from NEW and add it to V',
   c. If $Z \in T$,
      Then add $(C, Z) \rightarrow Z$ to P';
      Else for all $Z \rightarrow Z_1 \cdots Z_n \in P$, $n \geq 0$, do
      1) Add $(C, Z) \rightarrow (C_1, Z_1) \cdots (C_n, Z_n)$ to P'
         where $C_j = [xZ_1 \cdots Z_{j-1}]$,
         for some $x \in C$ and all j, $1 \leq j \leq n$.
      2) For all j, $1 \leq j \leq n$,
         if (C_j, Z_j) is not in NEW $\cup$ V',
         then add (C_j, Z_j) to NEW.
3. Define $S' = ([\lambda], S)$.

A careful reading of Algorithm 18 shows that all it does is to construct the new grammar according to the ideas presented before. The following properties are important to note.

Fact 19 Let $G = (V, T, P, S)$ be a reduced grammar. Then

1. Algorithm 18 stops.
2. Step 2.c.1 of Algorithm 18 is well defined.
3. For all new variables (C, Z) introduced by Algorithm 18 we must have $xZ \in VP(G)$, for all $x \in C$.

Next we turn to an example.

Example 20 Consider the grammar $G = (\{S, A, B, a, b, c\}, \{a, b, c\}, P, S)$, where P is given by the productions below:

$S \rightarrow aA, \quad S \rightarrow aB,$

$A \rightarrow aaA, \quad A \rightarrow ac,$

$B \rightarrow aaB, \quad B \rightarrow b.$

Clearly, $L(G) = \{a^n c : n \geq 1$ and n even$\} \cup \{a^n b : n \geq 1$ and n odd$\}$. The SMSP parser for G does not obey the correct-prefix property, as can be seen by its behavior on the input string *aab*. Intuitively, the reason why the correct-prefix property fails is that the parser must proceed by first stacking all the a's and then deciding whether to stack the incoming b or c on top of them. Clearly, we would like to stack a symbol b if and only if the number of a's on the stack is odd. But this cannot be decided by looking at the topmost symbol on the stack, which is just an a. Similar reasoning applies for the symbol c.

The viable prefixes of G are partitioned into the classes

$C1 = \{\lambda\}$,
$C2 = \{a^n : n \geq 1$ and n even$\}$,
$C3 = \{a^n : n \geq 1$ and n odd$\}$,
$C4 = \{a^n A, a^n B, a^n b, a^n ac, S : n \geq 1$ and n odd$\}$

Algorithm 18 will produce the grammar $G' = (V', T, P', S')$ with productions

$(C1, S) \rightarrow (C1, a)(C3, A)$,

$(C1, S) \rightarrow (C1, a)(C3, B)$,

$(C3, A) \rightarrow (C3, a)(C2, a)(C3, A)$,

$(C3, A) \rightarrow (C3, a)(C2, c)$,

$(C3, B) \rightarrow (C3, a)(C2, a)(C3, B)$,

$(C3, B) \rightarrow (C3, b)$,

$(C1, a) \rightarrow a$,

$(C3, a) \rightarrow a$,

$(C2, c) \rightarrow c$,

$(C3, b) \rightarrow b$,

$(C2, a) \rightarrow a$.

It is easy to check that G' is a SMSP grammar whose SMSP parser has the viable-prefix property. The parser for G' stacks a symbol c if and only if the topmost symbol on the stack is $(C3, a)$. In fact, $C3$ indicates that we have so far stacked an even and nonzero number of a's. Similarly, there are two situations in which the parser stacks a symbol b: First, when it sees $(C2, a)$ on top of the stack, indicating that the number of a's the parser has already stacked is odd and greater than one; second, when it sees $(C1, a)$ on top of the stack, in which case it has seen exactly one a so far. Note that G' is no longer uniquely invertible and, hence, is not a WP grammar. □

It should be intuitively clear that the grammars produced by Algorithm 18 resemble the original ones very closely. This is made precise by the following definition [12].

Definition 21 Let $G = (V, T, P, S)$ and $G' = (V', T, P', S')$ be grammars. We say that G' *right-covers* G if and only if there is a morphism h from V' into V such that, for all $x \in T^*$,

1. If ρ is a right parse for x in G', then $h(\rho)$ is a right parse for x in G.
2. If ω is a right parse for x in G, then there is a right parse ρ for x in G' such that $h(\rho) = \omega$.

We also say that h is a *right-cover morphism* from G' into G.

From the definition it is clear that i) $L(G) = L(G')$ and ii) we can, using the cover morphism, replace G by G' in any translation mechanism that uses G as a basis for a syntax-driven parse without having to change any of the semantic functions already in operation. To see how this may be accomplished, note that all we need to do is to search the table that describes the cover morphism each time we perform a reduction when parsing according to G'. This will

give us a string ρ in P^* and we just perform all the semantic routines associated with the elements in ρ, taking one at a time from left to right.

Notation Let $G = (V, T, P, S)$ be a reduced grammar and let $G' = (V', T, P', S')$ be the grammar produced by Algorithm 18 when G was taken as input. Then define the morphism g from V' into V such that (C, Z) is mapped into Z for all $(C, Z) \in V'$. Define the morphism h from P' into P such that

1. Each production in the form $(C, Z) \rightarrow Z$ is mapped into λ.
2. Each production in the form $(C, Z) \rightarrow x$, where $x \neq Z$, is mapped into $Z \rightarrow g(x)$.

Theorem 22 Let $G = (V, T, P, S)$ be a grammar and let $G' = (V', T, P', S')$ be the grammar produced by Algorithm 18 when G is taken as input. Then G' right-covers G.

Having the structural equivalence between G and G', it remains to be shown that G' is a SMSP grammar whose SMSP parser obeys the viable-prefix property. The first step is to examine the relationship between the precedence relations in G and G'.

Lemma 23 Let $G = (V, T, P, S)$ be a reduced grammar with no null rules. Let $G' = (V', T, P', S')$ be the grammar produced by Algorithm 18 when G was taken as input. Then

1. G' is reduced and has no null rules.
2. The precedence relation $<$ in G' satisfies the following:
   a. $(C, X) < (D, Y)$ in G' implies $X < Y$ in G and $zX \in D$, for all $z \in C$.
   b. $(C, X) < a$ in G' implies $X < a$ in G.
   c. $\perp < (C, X)$ in G' implies $\perp < X$ in G and $C = [\lambda]$.
   d. We cannot have $a < Z$ in G', for any Z in V' and any $a \in T$.
3. The precedence relation $>$ in G' satisfies the following: $(C, X) > a$ in G' implies $X > a$ in G.

Theorem 24 Let $G = (V, T, P, S)$ be a reduced grammar with no null rules. Let $G' = (V', T, P', S')$ be the grammar produced by Algorithm 18 when G was taken as input. Then G' is a SMSP grammar.

To check the correctness of Algorithm 18 with respect to the viable-prefix property of G', we need the next crucial lemma. It imposes a restriction on the right-sentential forms of G' as follows. Let x be a viable prefix of G, and assume that x is in the class C [under $\phi(G)$]. Then, in any right-sentential form in G' variables in the form (C, Z) can only be preceded by strings y in such a way that $g(y) \in C$, where g is the morphism defined above. That is, the equivalence class C "remembers" all the viable prefixes of G that can precede x. In order to decide whether or not to stack the next incoming symbol, all the parser has to do is look at the first component of the topmost symbol in the stack.

Lemma 25 Let $G = (V, T, P, S)$ be a reduced grammar with no null rules. Let $G' = (V', T, P', S')$ be the grammar produced by Algorithm 18 when G was taken as input. Take some $(C, X) \in V'$ and let $x = X_1 \cdots X_n \in C$, where $n \geq 0$ and $X_i \in V$, for all i, $1 \leq i \leq n$. Then there are equivalence classes C_i, $1 \leq i \leq n$, such that

1. $z = (C_1, X_1) \cdots (C_n, X_n)(C_{n+1}, X_{n+1}) \in VP(G')$, where $(C_{n+1}, X_{n+1}) = (C, X)$ and
2. $X_1 \cdots X_{j-1} \in C_j$, for all j, $1 \leq j \leq n + 1$.

We can now state the last result of this section.

Theorem 26 Let $G = (V, T, P, S)$ be a reduced grammar with no null rules. Let $G' = (V', T, P', S')$ be the grammar produced by Algorithm 18 when G was taken as input. Assume that Q' is a SMSP parser associated with G'. Then Q' has the viable-prefix property.

One interesting fact about the proof of the last theorem is that it needs only the assumptions that G and G' are reduced and have no null rules. It does not require G or G' to be a SMSP grammar. This indicates that Algorithm 18 might also work for other classes of grammars as well, where null rules are avoided and unique invertibility is not a problem.

4. The cost of the transformation

This section investigates the space efficiency of the transformation presented in the last section. This is done by comparing the sizes of the original and transformed grammars. It turns out that there is an infinite family of SMSP grammars, say G_n, for which Algorithm 18 yields another family, say H_n, in such a way that the size of H_n is exponentially related to the size of G_n. This shows that grammars and parsers produced by Algorithm 18 can be quite large. Interestingly enough, we also show that this is the best that one can hope for. More precisely, let $L_n = L(G_n)$. We show an exponential lower bound on the size of any family of SMSP grammars, F_n, for which we must have $L(F_n) = L_n$ and whose corresponding SMSP parsers must obey the correct-prefix property. In other words, the (possible) exponential growth in the size of the transformed grammars is due to the requirement imposed on the corresponding SMSP parsers, namely that they must obey the correct-prefix property, and is not a drawback inherent in the transformation itself. Further, with respect to space efficiency, the transformation is optimal. Note that the presence of the family G_n, in itself, does not render the transformation useless from a practical point of view. It does say, however, that, in general, one cannot construct a better one (that is, with respect to space efficiency).

We use the following measure for the size of a grammar.

Definition 27 Let $G = (V, T, P, S)$ be a grammar. Define $|G|$ as the sum of $|Ax|$, for all $A \to x \in P$.

Before introducing our family of languages, we discuss them informally. We need to construct a family of languages, L_n, in such a way that any grammar for L_n (and whose corresponding SMSP parser has the correct-prefix property) is very large: of the order of 2^n. This suggests that we, somehow, encode into L_n a description of all 2^n subsets of $\{1, \cdots, n\}$. In order to do that, we use the symbols a_i and b_i, $1 \le i \le n$, and force the sentences of L_n to be of the form $xb_k y$ where in x only the a_i's occur. The encoding is given by reading the indices of the a_i's. To make the language nonregular and to avoid trivial cases, we require y to have the same length as x; i.e., we let $y = c^r$, where c is a new terminal symbol and $r = |x|$. Finally, the idea behind the central symbol, b_k, is as follows: We require k to be an index *not* occurring in x. Now, if the parser is to have the correct-prefix property, it has to "remember," up to the moment when it encounters the b_k symbol, which subset of indices it has found so far among the a_i's. This will force the parser to have a lot of "states" and, in consequence, will ensure that the grammar from which it was built is also very large. Observe that, in case the correct-prefix property is *not* needed, the parser could proceed as follows: i) Stack all the a_i's; ii) read in b_k and "remember" it; iii) reduce back all the stack, one symbol at a time, making sure that each a_i taken from the stack has an index different from k (of course, it is also necessary to read all the c's from the input and check whether their number matches the number of a's on the stack).

The next step is to verify that these ideas do, indeed, produce the necessary results. We define the following infinite families of sets, indexed by n:

- $\Delta_n = \{a_1, \cdots, a_n\}$ and $\Gamma_n = \{b_1, \cdots, b_n\}$ are new terminal symbols.
- $T_n = \{d, c\} \cup \Delta_n \cup \Gamma_n$. These are the set of terminal symbols of our family of grammars, to be defined below.
- $I_n = \{1, \cdots, n\}$, a set of indices.

Next, we need a function to form sets of indices. Let $\#_n$ be a function that maps Δ_n^* into subsets of I_n such that

$$\#_n(x) = \{i : x = ya_i z, \text{ for some } y, z \in \Delta_n^*\};$$

i.e., $\#_n(x)$ collects all the indices of symbols in x. We can now present the family of languages.

Definition 28 Let $n \ge 1$. For all k, $1 \le k \le n$, define $L_{n,k} = \{xb_k c^r : x \in \Delta_n^*, r = |x| \text{ and } k \text{ is not in } \#_n(x)\} \cup \{da_k\}$. Also let L_n be the union of all $L_{n,k}$ for all k, $1 \le k \le n$.

As defined, L_n is exactly as we introduced before, except for the components in the form da_k, added for technical reasons. It is not hard to devise a family of grammars to generate L_n.

Definition 29 Let $n \ge 1$ and define $G_n = (V_n, T_n, P_n, S_n)$, where T_n is as defined above, $V_n = T_n \cup \{S_n\} \cup \{A_1, \cdots, A_n\}$, and P_n is given by the productions below, with $1 \le i, j \le n$:

1. $S_n \to da_i$, $S_n \to A_i$.
2. $A_i \to b_i$, $A_i \to a_j A_i c$, where $i \ne j$.

The family G_n has some resemblance to another one used in [7].

Theorem 30 Let $n \ge 1$ and let G_n be the grammar of Definition 29. Then, for all $n \ge 3$, G_n is a SMSP grammar such that $L(G_n) = L_n$ and $|G_n| \le k \cdot n^2$, for some constant k. Moreover, the SMSP parser for G_n does not obey the correct-prefix property.

The statements about G_n follow easily from the definitions. To verify that the SMSP parser for G_n does not have the correct-prefix property, it suffices to observe its behavior on any string in the form $da_i a_j$, where $i \ne j$. In fact, the reason to add the elements da_k to the language was, precisely, to force the SMSP parser for G_n to violate the correct-prefix property. Further, since the parser for G_n does not have the correct-prefix property, we can use Algorithm 18 and transform each G_n into an equivalent SMSP grammar, say H_n, such that the SMSP parser associated with the latter always obeys the viable-prefix property.

Theorem 31 For all $n \ge 1$, let H_n be the grammar produced by Algorithm 18 when G_n is taken as input. Then $|H_n| \ge 2^n \cdot (2n + 1)n$, for all $n \ge 1$.

The result is obtained by analyzing the transformation carried out by Algorithm 18 on G_n. A careful counting argument suffices. The grammars G_n, therefore, exhibit the property of being much more (exponentially) space-efficient than the equivalent grammars H_n obtained by Algorithm 18. Our last task will be to show that this is true of *any* family of equivalent grammars, say F_n, whose parsers are required to obey the correct-prefix property. This will be accomplished by proving a lower bound of the form $n2^n$ on the size of F_n.

Definition 32 For all $n \ge 1$, let $F_n = (V_n, T_n, P_n, S_n)$ be a family of reduced SMSP grammars such that $L_n = L(F_n)$. Assume that Q_n, the SMSP parser associated with F_n, obeys the correct-prefix property.

The lower bound on $|F_n|$ is obtained with the aid of the following notion.

Definition 33 Let $n \ge 1$ and let F_n and Q_n be as specified in Definition 32. Also, let $x \in V_n^*$ and $a_i \in \Delta_n$, for some i, $1 \le i \le n$. We say that x is *invariant* with respect to a_i in F_n if and only if

1. $(\perp x)\colon 1 < a_i$ in F_n.
2. For all $r \geq 0$, if
   $(\perp x, (a_i)^r a_i \perp, \lambda) \vdash^* (\perp z, a_i \perp, \rho)$,
   then we must have $z = xy$, for some $y \in V_n^*$.

The important point in the previous definition is that the string x is left untouched in the bottom of the stack when the parser analyzes inputs in the form $a_i \cdots a_i$. Now, in order to prove the required properties of $|F_n|$, we associate with each pair (J, j), where $J \subset I_n$ and $1 \leq j \leq n$, a production $P_{J,j}$ of P_n. Next we show that $P_{J,j} \neq P_{K,k}$ whenever $(K, k) \neq (J, j)$. This should be enough to guarantee that $|F_n| \geq n2^n$. The particular productions are chosen based on two properties of F_n:

1. For all right-sentential forms x of F_n, there is a (sufficiently large) constant k such that, starting in the configuration $(\perp x, (a_i)^k \perp, \lambda)$, the SMSP parser for F_n is forced into a configuration $(\perp y, a_i \perp, \rho)$, where y is invariant with respect to a_i. That is, invariants do exist.
2. If x and y are invariants with respect to some a_i and $x \Rightarrow_r^+ u$, $y \Rightarrow_r^+ v$, where $u, v \in \Delta_n^*$, then we must have $\#_n(ua_i) = \#_n(va_i)$. In other words, sentential forms invariant with respect to the same symbol a_i can only generate terminal strings with the same set of indices. Invariance, thus, can be taken as a "memory" of the set of indices we have already laid down.

In possession of these properties we can state the following crucial lemma.

Lemma 34 Let $n \geq 1$ and let F_n be as specified in Definition 32. Fix some i, $1 \leq i \leq n$. Then to each pair (J, j), where $J \subset I_n$, $1 \leq j \leq n$, $J \cup (i, j) \neq I_n$, and j is not in J, we can associate a production $P_{J,j}$ of P_n in such a way that $P_{J,j} \neq P_{K,k}$ whenever $(K, k) \neq (J, j)$ and i is not in $K \cup J \cup \{k, j\}$.

The lemma says that, with the exception of some pathological situations, there are about as many productions in H_n as there are pairs in the form (J, j). Finally, our lower bound is at hand: The number of such pathological cases can be neglected when compared to $n \cdot 2^n$, which is a lower bound on the number of productions in H_n.

Theorem 35 Let $n \geq 1$ and let F_n be as specified in Definition 32. Then we must have $|F_n| \geq n \cdot 2^n$ and $|V_n| \geq 2^n$.

5. Conclusions

We have studied the correct-prefix and the viable-prefix properties in connection with the weak and simple mixed-strategy precedence classes of grammars. Grammars in these classes usually do not give rise to parsers that have either of these properties. We showed and proved correct a simple transformation that takes any SMSP grammar G and produces an equivalent SMSP grammar G' in such a way that the SMSP parser associated with G' always obeys the viable-prefix property. The transformation is of such a nature that a very simple cover morphism can be defined from the productions of G' into the production of G. This enables one to use G' instead of G to parse $L(G)$ while still preserving the semantic routines designed for the original grammar G. For the weak precedence class it was established that no such transformation exists.

Using the sizes of G and G' as a measure, the cost of the transformation was also analyzed. In general, one must contemplate an exponential growth in the size of the transformed grammar if the correct-prefix property is to be enforced. Hence, there can be an exponential economy in describing SMSP languages when the correct-prefix property is not crucial to the parsing mechanism. The necessity of the exponential growth in the size of the transformed grammar was established by exhibiting a particular family of SMSP languages and proving a lower bound on the size of any SMSP grammars for these languages whose associated SMSP parsers are required to obey the correct-prefix property.

References

1. D. E. Knuth, "On the Translation of Languages from Left to Right," *Info. & Control* **8,** No. 6, 607–639 (1965).
2. A. V. Aho, P. J. Denning, and J. D. Ullman, "Weak and Mixed Strategy Precedence Parsing," *J. ACM* **19,** No. 2, 225–243 (1972).
3. R. W. Floyd, "Syntactic Analysis and Operator Precedence," *J. ACM* **10,** No. 3, 316–333 (1963).
4. J. D. Ichbiah and S. P. Morse, "A Technique for Generating Almost Optimal Floyd-Evans Productions for Precedence Parsers," *Commun. ACM* **13,** No. 8, 501–508 (1970).
5. N. Wirth and H. Weber, "A Generalization of ALGOL and Its Formal Definition: Part I," *Commun. ACM* **9,** No. 1, 13–25 (1966).
6. M. M. Geller and M. A. Harrison, "Characteristic Parsing: A Framework for Producing Compact Deterministic Parsers, I.," *Comput. & Syst. Sci.* **14,** No. 3, 256–317 (1977).
7. M. M. Geller and M. A. Harrison, "Characteristic Parsing: A Framework for Producing Compact Deterministic Parsers, II.," *Comput. & Syst. Sci.* **14,** No. 3, 318–343 (1977).
8. F. L. DeRemer, "Simple LR(k) Grammars," *Commun. ACM* **14,** No. 7, 453–460 (1971).
9. F. L. DeRemer, "Practical Translators for LR(k) Languages," Ph.D. Dissertation, Massachusetts Institute of Technology, Cambridge, MA, 1969.
10. K. R. Moll, "Left Context Precedence Grammars," *Acta Informat.* **14,** No. 4, 317–335 (1980).
11. J. N. Gray and M. A. Harrison, "On the Covering and Reduction Problem for Context-Free Grammars," *J. ACM* **19,** No. 1, 675–698 (1972).
12. A. Nijholt, "On the Covering of Parsable Grammars," *J. Comput. & Syst. Sci.* **15,** No. 4, 99–110 (1977).
13. M. A. Harrison, *Introduction to Formal Language Theory,* Addison-Wesley Publishing Co., Reading, MA, 1978.
14. A. V. Moura, "On the Cost of the Viable-Prefix Property in Precedence Parsers," *Technical Report CCB 029,* IBM Scientific Center, Brasilia, Brazil, 1985.
15. A. V. Aho and J. D. Ullman, *The Theory of Parsing, Translation and Compiling,* Vol. 1, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1972.
16. A. V. Aho and J. D. Ullman, *The Theory of Parsing, Translation and Compiling,* Vol. 2, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1973.

17. S. L. Graham and S. P. Rhodes, "Practical Syntactic Error Recovery," *Commun. ACM* **18,** No. 11, 639–650 (1975).
18. R. P. Leinius, "Error Detection and Recovery for Syntax-Directed Compiler Systems," Ph.D. Dissertation, University of Wisconsin, Madison, WI, 1970.
19. W. M. McKeeman, J. J. Horning, and D. B. Wortman, *A Compiler Generator*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1970.
20. P. Wirostek, "On the 'Correct Prefix Property' in Precedence Parsers," *Info. Proc. Lett.* **17,** No. 10, 161–165 (1983).

Received July 15, 1985; accepted for publication May 28, 1986

Arnaldo V. Moura *IBM Software Technology Center, Rua Tutoia 1157-8° Andar, 04007 São Paulo, Brazil.* Dr. Moura is a member of the Research staff at the Software Center in São Paulo. He received a B.S. degree in electrical engineering from the Aeronautics Institute of Technology, São Jose dos Campos, Brazil, an M.S. from the same institution, and a Ph.D. in computer science from the University of California, Berkeley. From 1980 to 1984 he was a Research staff member at the IBM Scientific Center in Brasilia, where he conducted research in formal languages and algorithms. Dr. Moura's current activities and interests include algorithms, formal languages, and rigorous methods for software specification, design, and verification.