Towards a Discriminating-Reverse Tomita-like Parser with Reduced Nondeterminism

Fortes Gálvez, José
Dep. Informática, Universidad de Las Palmas de Gran Canaria
jfortes@dis.ulpgc.es

Jacques Farré
Laboratoire I3S, CNRS and Université de Nice - Sophia Antipolis
jf@essi.fr

Abstract A new approach to generalized parsing for context-free grammars is presented, which uses an underlying discriminating-reverse, DR(k), parser with a Tomita-like nondeterminisim-controlling graph-structured stack, GSS, algorithm.

The advantage of the new generalized discriminating-reverse, GDR, approach over GLR would lie on the possibility of using DR(k) parsers, which combine full LR(k) parsing power with a small number of states even for $k > 1$. This may allow to greatly reduce nondeterminism originating from limited parsing power (as it is typical of the restricted form of (direct) LR parsers currently used in Tomita algorithm) and to a further simplification of the GSS by associating nodes to symbols instead of direct-LR states. Moreover, DR(k) parsing time complexity has been shown to be linear for LR(k) grammars, and DR(k) parser efficiency has been practically found to be very similar to direct LR(k) parsers.

The paper shows the nondeterministic DR(k) generation algorithm (for non-LR(k) grammars) and the corresponding adaptation of the GSS algorithm.

1 *Introduction*

The discriminating-reverse, DR(k), method [3, 4, 6] accepts the full class of LR(k) grammars, while producing linear, small and efficient deterministic parsers. It is based on the idea of exploring from the stack top a (typically very) small, minimal parsing stack suffix, in order to discriminate what actions are compatible with the suffix portion already read, plus the k-lookahead. If the grammar is LR(k), the number of such compatible actions eventually reduces to one. As soon as this happens, that action can be decided, and thus performed.

The method is completely proven, theoretically and practically, with very good efficiency results [5], producing, for $k = 1$, deterministic parser automata smaller than the corresponding (deterministic or not) LR(0) ones, and with an average stack exploration depth per parsing action of less than 2 topmost symbols. Even larger values of k do not significatively enlarge automata size, due essentially to the typically very small depth of the discrimination process.

The well known generalized LR, GLR, method [12, 13] originally devised by M. Tomita employs the bottom-to-top LR(k) parser automaton [8, 11] originally developed by D. Knuth to guide an essentially nondeterministic parsing, with controlled complexity through the use of a "graph-structured stack", GSS. The origin of this nondeterminism can be located, in some cases, in the strict ambiguity of natural language grammars to which the method is addressed, but in other cases, in the impossibility for the underlying LR parser to determine the action because of restrictions imposed on it. These restrictions include merging significant automaton states, and using a too small value for lookahead window size k, as in LALR(1). Unfortunately, values of k as low as 1 usually produce unacceptably large LR(k) automata, because of a double combinatorial explosion in the number of states coding relevant stack-plus-lookahead content classes, and (for $k > 1$) in the combination of the allowed sequences of k terminal symbols.

In this paper we propose to use DR(k) as the underlying parser to combine with a Tomita-like GSS. This new approach would have the advantage of using small parsers, even for values of $k > 1$. Since these parsers have the same parsing power as LR(k), nondeterminism due to restrictions on the underlying parser would be significatively reduced. This effect is even of greater importance when parsing typical natural language sentences, because of the relatively large ratio between lookahead size and sentence length. On the other hand, DR(k) parsers are in principle non-correct prefix, what may introduce a different form of nondeterminism, although a correct-prefix variant is proposed in [6] at the cost of some increase in parser size.

In this paper we will first present in detail the underlying DR(k) parser generation algorithm for ϵ-free context-free grammars, and then an informal presentation of its application to a Tomita-like general parser.

We exclude in a first time grammars with ϵ-productions, since they involve problems with some "ill-designed" grammars [10], i.e., those with hidden-left recursion possibly including cyclicity. However, we feel that in a reverse approach (downwards exploration from the stack top), as ours, it is easier to deal with those problems—as it is hinted in [9]—, because no state is required to code every possible valid stack configuration.

Moreover, we are currently developing noncanonical extensions [2] to DR(k) which would naturally integrate that kind of ambiguous grammars when applied to a GSS algorithm. In this approach, conflicting actions are deferred, and stack "marks" code these conflicting actions, until sufficient right context is read that permit to choose. Deferred actions produce parsing steps in an essentially noncanonical order, but allow further reduction of explicit nondeterminism. An example of the behaviour of such a kind of parsers can be found in [1].

Notations

Notation will be conventional in its most part, see [11] for instance.

The original grammar G is augmented with $P' = \{S' \rightarrow \vdash S \dashv\} \cup P$, with $\text{Follow}_k(S') = \{\dashv^k\}$. Productions in P' are numbered from 1 to n, and a production with number i can be represented as $A \stackrel{i}{\longrightarrow} \alpha$.

Parsing action i represents by convention the production number to use in a reduction if $1 \leq i \leq n$, or to shift if $i = 0$.

The head operator is extended as follows for strings with dot: $k : x \bullet y = x' \bullet y'$, such that $x' = k : x$ and $x'y' = k : xy$.

2 *Discriminating-reverse parsing*

In essence, what is needed in a shift-reduce parser is some procedure to tell what parsing action to perform next. When parsing sentences from an LR(k) grammar [11], the language of legal stack-plus-lookahead contents is the union of several pairwise disjoint regular sublanguages, each of them mapped to a unique parsing action. Accordingly, in our method a *stack suffix* exploration automaton is built that begins reading from the top of the stack, and stops as soon as the current lookahead or the next stack symbol allows to determine the parsing action to perform.

2.1 DR(k) single-action reverse stack acceptors

These acceptors are DFA [7] which read the stack from its top, and accept precisely the language of all the stacks which are compatible with a certain parsing action. In their construction procedure each state is associated with a set of LR(k) *reverse* items, which have the form $[A \rightarrow \alpha \bullet \beta, x \bullet y]$, with $|xy| = k$. In the item, the *rightpart* dot, located in the rule rightpart between α and β, corresponds to the current exploration point within the stack, i.e, the left end of the suffix already read. And the lookahead string is divided by the *lookahead* dot into two parts: the lookahead part x already covered by a (possibly partially) explored phrase of β, and the lookahead part y not yet covered.

The following function computes the item set for the "context" surrounding some nonterminal A restricted to some dotted lookahead $x \bullet yz$.

$$\begin{aligned}&\mathcal{C}^0_k(A, x \bullet yz) = \\ &\quad \{[B \rightarrow \beta \bullet \gamma,\ xy \bullet z] \mid \gamma \underset{\text{rm}}{\Rightarrow}^* Av,\ B \rightarrow \beta\gamma \in P', \\ &\qquad S' \underset{\text{rm}}{\Rightarrow}^* \varphi B w, xy \bullet z = k : xv \bullet w \dashv^k\}.\end{aligned}$$

The closure of a set I of LR(k) reverse items is defined by:

$$\begin{aligned}&\mathcal{C}_k(I) = I \cup \\ &\quad \{\iota \mid \iota \in \mathcal{C}^0_k(A, x \bullet yz), [A \rightarrow \bullet \alpha,\ x \bullet yz] \in I\}\end{aligned}$$

The construction of an automaton $\mathcal{A}_k(i)$ for parsing action i from a reduced grammar $\mathcal{G}$ is as follows:

1. Initial state:
   $I^i_0 =$

$$\begin{cases} i=0: & \mathcal{C}_k(\{[A\to\alpha\bullet a\beta, k: av\bullet w\dashv^k] \mid \\ & \quad A\to\alpha a\beta \in P', \\ & \quad \beta \underset{\mathrm{rm}}{\Rightarrow}^* v, S' \underset{\mathrm{rm}}{\Rightarrow}^* \varphi A w\}) \\ \\ 0<i: & \mathcal{C}_k(\{[A\to\alpha\bullet, k: \bullet w\dashv^k] \mid \\ & \quad A\overset{i}{\longrightarrow}\alpha \in P', S' \underset{\mathrm{rm}}{\Rightarrow}^* \varphi A w\}) \end{cases}$$

2. Transition function:

$$\delta(I,X) = \mathcal{C}_k(\{[A\to\alpha\bullet X\beta, x\bullet y] \mid [A\to\alpha X\bullet\beta, x\bullet y] \in I\})$$

2.2 The DR(k) deterministic automaton

A recognizing automaton $\mathcal{A}_k$ can be built for precisely the language of all legal stacks, by joining all these single-action automata $\mathcal{A}_k(i)$. For this purpose, all the initial states can be joined into a single initial state. Items from each $A_k(i)$ initial state are joined together, while retaining the action number i of their corresponding single-action automata, i.e., items take the form of $[i, A\to\alpha\bullet\beta, x\bullet y]$. $\mathcal{A}_k$ can now be built by a construction analogous to the algorithm for converting an NFA to DFA [7], or, equivalently, by applying the transition function δ from the joint initial state, and preserving item action numbers.

If G is LR(k), the automaton built according to the above procedure can always indicate the next parsing action after having read the whole stack since, according to the LR(k) property, then the lookahead string will always suffice to discriminate amongst the would-be different actions.

But in almost all practical LR(k) grammars a small section of automaton $\mathcal{A}_k$ is enough to decide the parsing action. In fact, the construction algorithm *begins* building such a recognizing automaton, but only generates its useful discriminating (and full rightpart verification, see below) section. This mechamism dramatically prunes the recognizing automaton, while preserving its LR(k) parsing power, since the same parsing action decisions are obtained. Moreover, the resulting discriminating automata is minimal in a sense, since its construction stops at the earliest possible point.

Finally, the automaton can frequently decide what rule to apply before having reached the left end of the rule's rightpart. However, the presence of the full rightpart on top of the stack must be verified in order to assure a valid reduction. This test is naturally included in the automaton, by continuing state construction at least until left ends of rightparts are read. To implement rightpart checking, rule numbers are marked in the initial state with a dot to the right as in $[i\bullet, A\to\alpha\bullet, x\bullet y]$, to indicate that rightpart of rule i must be completed before reduction could be decided. For the shift action as well as in the $\mathcal{C}_k^0$, new items are generated with the dot to the left of the action number only. In the following, this dot will be indicated only when significant.

2.3 Nondeterministic DR(k) generation algorithm

The algorithm is presented in Figure 1. At each state, there are two sources for action discrimination: next stack symbol and current lookahead. The method that seems most natural was chosen: first the lookahead is used to try to decide the parsing action, i.e., if all the situations with a given lookahead signal the same action. Nevertheless, when the state items show that it is useless to continue stack exploration because some set of actions cannot be discriminated from that point on, the compatible lookaheads are used to "decide" that set of actions.

For the remaining cases in which the lookahead test would not decide, the next possible stack symbols are used to discriminate the parsing action (i.e., all the remaining situations with a given next stack symbol signalling the same action) or, otherwise, to compute transitions to other states. However, since a lookahead cannot decide a set of actions when the rightpart of some reduction in the set has not yet been fully checked, it is decided by the stack symbol if it is the leftmost one of the longest rightpart amongst the reductions in the set.

According to this construction, empty lookahead table entries mean that the decision must be taken, when possible, by the next stack symbol, while empty stack table entries code error entries, since in those cases the explored suffix is found to be illegal.

The algorithm minimizes parsing nondeterminism for sentences[1], that is, an action set is decided when, according to the grammar, it cannot be further refined by the re-

[1] If decisions are decided on the lookahead only, a discriminatory power equivalent to full LR(k) is also obtained for erroneous inputs.

algorithm Generator of nondeterministic DR(k) parsing table with full rightpart verification
input ϵ-free context-free grammar $\mathcal{G}$
output Parsing table *ActTrans(state,kind,seq_or_symb)*
begin
  $Aut := \emptyset$; $ActTrans := \emptyset$
  $H_0 := \{[i\bullet, A\to\alpha\bullet, \bullet y] \mid A \xrightarrow{i} \alpha \in P' \text{ and } y \in \text{Follow}_k(A)\} \cup$
    $\{[\bullet 0, A\to\alpha\bullet a\beta, av\bullet w] \mid A\to\alpha a\beta \in P',\ v \in \text{First}_{k-1}(\beta), w \in \text{Follow}_{k-1-|v|}(A)\}$
  $I_0 := \mathcal{C}_k(H_0)$; add I_0 to Aut

  for $I \in Aut$ **do**
    $J := I$

    % decisions upon current lookahead
    for $w \in V'^k$ **do**
      $La := \{i \mid [i, A\to\alpha\bullet\alpha', x\bullet y] \in I, \nexists [j\bullet, B\to\beta Y\bullet\beta', x'\bullet y'] \in I, w = xy = x'y'\}$
      if $La = \{i\}$ **then**
        % discrimination upon current lookahead
        $ActTrans(I,\text{lookahead},w) := \{i\}$
        $J := J - \{[i, A\to\alpha\bullet\alpha', x\bullet y] \in I \mid w = xy\}$
      else if $La \neq \emptyset$ **then**
        % check if same left context for all actions in the set
        if $\forall A\alpha,\ \{i \mid [i, A\to\alpha\bullet\beta, x\bullet y] \in I, w = xy\} \in \{La, \emptyset\}$ and
        $\nexists [i, A\to\alpha\bullet\beta, x\bullet y], [i', A\to\alpha\bullet\beta', x'\bullet y'] \in I,\ xy = x'y' = w,\ \mathcal{C}^0_k(A, x\bullet y) \neq \mathcal{C}^0_k(A, x'\bullet y')$
        then
          % non-DR(k)-grammar nondeterminism
          $ActTrans(I,\text{lookahead},w) := La$
          $J := J - \{[i, A\to\alpha\bullet\alpha', x\bullet y] \in I \mid w = xy\}$

    % decisions upon next stack symbol
    for $X \in V'$ **do**
      $St := \{i \mid [i, A\to\alpha X\bullet\alpha', x\bullet y] \in J, \nexists [j\bullet, B\to\beta Y X\bullet\beta', x'\bullet y'] \in J\}$
      if $St = \{i\}$ **then**
        % discrimination upon next stack symbol
        $ActTrans(I,\text{stack},X) := \{i\}$
        $J := J - \{[i, A\to\alpha X\bullet\alpha', x\bullet y] \in J\}$
      else if $St \neq \emptyset$ **then**
        % check if same contexts for all actions in the set
        if $\forall Aw,\ \{i \mid [i, A\to X\bullet\beta, x\bullet y] \in I, xy = w\} \in \{La, \emptyset\}$ and
        $\nexists [i, A\to X\bullet\beta, x\bullet y], [i', A\to X\bullet\beta', x'\bullet y'] \in I,\ xy = x'y',\ \mathcal{C}^0_k(A, x\bullet y) \neq \mathcal{C}^0_k(A, x'\bullet y')$
        then
          % non-DR(k)-grammar nondeterminism
          $ActTrans(I,\text{stack},X) := St$
          $J := J - \{[i, A\to\alpha X\bullet\alpha', x\bullet y] \in J\}$

    % state transitions upon next stack symbol
    for $X \mid [i, A\to\alpha X\bullet\beta, x\bullet y] \in J$ **do**
      $K := \mathcal{C}_k(\{[j, B\to\gamma\bullet X\delta, v\bullet w] \mid [j, B\to\gamma X\bullet\delta, v\bullet w] \in J\})$
      add K to Aut; $ActTrans(I,\text{stack},X) := \{\text{goto } K\}$
end

Figure 1: Generation algorithm of nondeterministic DR(k) parsing table.

maining stack (plus lookahead) context. Nevertheless, under such restriction, the generator produces an automaton with minimal stack exploration depth per action.

Please note that the algorithm is presented with emphasis on clarity rather than efficiency. In particular, k-lookaheads strings are treated as units, while in practice it might be interesting to apply some optimization to the process of matching them against current lookahead. It it might also be interesting to adopt a minor variant[6] in order to decide in the same state on the stack symbol instead on the lookahead, if possible.

Moreover, a variant of the decision mechanism is also proposed in [6], consisting in delaying the consultation of the lookahead window until it is certain that it will produce a parsing decision. At the cost of some enlargement of the table size and automaton depth, this variant has the benefit of usually reducing the number of table consultations per action, and also reducing the cost of frequently handling $k > 1$ windows.

Finally, we include in the reverse-exploration automaton the verification of the remaining prefixes of rightparts in cases where the corresponding reduction action can be already discriminated on a rightpart suffix.

2.4 Example of nondeterministic DR(1) parsing table

As an example, Figure 2 shows the nondeterministic DR(1) parsing table for the following ambiguous grammar G_{cp}, extracted from [13]:

	Grammar G_{cp}
1:	$S' \rightarrow \vdash S \dashv$
2:	$S \rightarrow NP\ VP$
3:	$S \rightarrow S\ PP$
4:	$S \rightarrow S\ and\ S$
5:	$NP \rightarrow n$
6:	$NP \rightarrow det\ n$
7:	$NP \rightarrow NP\ PP$
8:	$NP \rightarrow NP\ and\ NP$
9:	$VP \rightarrow v\ NP$
10:	$VP \rightarrow v\ S$
11:	$PP \rightarrow p\ NP$

Note that the DR(1) parsing table (with full LR(1) parsing discrimination for sentences) is smaller than the SLR(1) table shown in [13], which has 18 states. Nevertheless, at the cost of increasing nondeterminism, our table could pass to 10 states by removing states with lookahead consultation only (e.g., in q_6 the action set $\{0, 9\}$ could be decided, instead of the transition to q_8).

3 *Generalized discriminating-reverse parsing*

We first describe a single-decision parsing process, and then its adaptation to a GSS. As we have seen, differently from GLR parsers, our GSS will not, in principle, contain automaton states, but plain stack (terminal and nonterminal) symbols. This actually considerably simplifies (and clarifies) the GSS in comparison with a more complex LR-state-based GSS as in classical Tomita parsers.

3.1 Single-decision DR(k) parsing

The underlying DR(k) parser uses a previously generated action and transition table for some context-free grammar. State item sets themselves need not to be stored at parsing time, and thus state codes are used instead.

The parsing stack is initialized with a bottom-of-stack marker at the beginning of parsing. For each parsing-action decision, stack exploration begins from the top at initial state. In each state, checking the lookahead is made first, else the next stack symbol is used for a decision or transition. This exploration will eventually finish with deciding a nonempty set of actions (shift, reduce, or accept), or detecting that the stack suffix read is not syntactically valid.

Since the parser is not intended to check whether the whole stack is syntactically valid, the stack is not necessarily a viable prefix, although the explored stack suffix might be at the same time compatible with some set of actions.

3.2 GSS-based DR(k) parsing

It is straightforward to extend this single-decision DR(k) parser for use with a Tomita-like GSS, and here we simply sketch our algorithm adaptation, since it essentially follows the same general approach of Tomita's algorithm[13]. The parsing algorithm, and the computation of the packed shared forest, are shown in Figure 3.

Before each input-shift step is performed, the generalized DR (GDR) parser has to explore all the possible paths beginning from the *current tops* of the GSS, following the same steps of the single-decision parser, and finally performing the corresponding (reduce, or removal[2]in case of error) actions on the GSS. That is, a new exploration has, in principle, to be restarted for each possible GSS topmost path allowed by the DR(k) automaton. However, in order to avoid to repeatedly make the same transitions, parser states can be temporarily attached to the correspond-

[2]Not shown in the algorithm. Removal is implicit, since incorrect paths will not produce new top nodes. Node deletion can be left to a garbage collector.

	$\dashv$	*and*	*n*	*det*	*v*	*p*	$\vdash$	$\dashv$	*S*	*NP*	*VP*	*PP*	*and*	*n*	*det*	*v*	*p*
q_0			0	0				q_1	q_4	q_6	q_2	q_3		q_5			
q_1									q_{14}								
q_2										2							
q_3									3	7							
q_4							0						q_{11}			q_{12}	
q_5							5						5		6	5	5
q_6							0						q_7			q_8	q_9
q_7									0	q_{10}							
q_8	9	0,9			0	0,9											
q_9	11	0,11			11	0,11											
q_{10}	8	0,8			8	0,8											
q_{11}									q_{13}								
q_{12}	10	0,10				0,10											
q_{13}	4	0,4				0,4											
q_{14}							1										

Figure 2: Nondeterministic DR(1) parsing table for grammar G_{cp}.

ing GSS suffix symbols leading to them, and thus exploration can be resumed from those states.

For the remaining part, the GDR method works very much like Tomita's parser, in a breadth-first manner, by replicating (and merging when possible, i.e., when two nodes for the same symbol have same predecessors in the GSS) the stack tops for all the reductions to perform, and eventually shifting the next input terminal jointly for the different tops.

3.3 Packed shared forest computation

Nodes of the GSS are easily and naturally reused for the packed shared forest representation. Single derivations from a node are represented by its corresponding set *Deriv* containing (*rule-number, leftmost-node, rightmost-node*) triples. Cyclic grammars are naturally supported. In the end, the different parses of the input text are found in the derivations of the unique remaining top node, which has S' as symbol.

As in Tomita's algorithm, ours also merges nodes, and thus its trees, in local ambiguity, i.e., when they correspond to a same symbol and the set of predecessor nodes in the GSS is the same.

4 Conclusions

It has been shown that Tomita's approach of using a GSS to control nondeterminism associated to parsing ambiguous natural-language grammars can be adapted to using a different, discriminating-reverse parser. This results in GDR parsers with some interesting properties:

- Increased LR(k) parsing power is expected to significatively reduce nondeterminism originating from using a restricted form of direct LR parser.
- The GSS is further simplified because of using plain vocabulary symbols instead of automaton states.
- Underlying DR(k) parsers are small, even for $k > 1$.
- They are linear for LR(k) grammars, and have been found to be efficient in practice (from $k = 1$ implementation).

On the other hand, it is not clear whether it would be interesting to use a correct-prefix variant, at the cost of a larger automaton, in order to avoid the introduction a new form of nondeterminism.

In conclusion, this new approach deserves further research from both the theoretical and practical viewpoints, in order to evaluate its interest as an alternative to conventional direct-LR based generalized parsing. It is currently being considered for refinement and implementation, both in terms of handling ill-designed grammars, in combination with noncanonical extensions to DR(k), and for using windows of size $k > 1$.

References

[1] J. Farré. Discriminant reverse LR parsing of context-free grammars. In *Proceedings of the Sixth International Workshop on Parsing Technologies, IWPT 2000*, pages 303–304, ITC-irst, Trento (Italy), Feb. 2000.

[2] J. Farré and J. Fortes Gálvez. A basis for looping extensions to discriminating-reverse parsing. In *5th International Conference*

algorithm GDR(k) parsing algorithm
input A parsing table *ActTrans* and an input string z
output Forest of parsing trees rooted at *reduced* or rejection on erroneous input
begin
$shifttops := \emptyset$
$x := \vdash$; **for** $i \in \{2, \ldots, k\}$ **do** read(a); $x := xa$
repeat
% Parsing actions on next input symbol
read (a); $\nu_0 := NewNode(1 : xa)$; $Prev(\nu_0) := shifttops$; $x := xa : k$
$currtops := \{\nu_0\}$; $shifttops := \emptyset$; $reduced := \emptyset$
repeat
% GSS exploration
$reductions := \emptyset$
$curr := \{(\nu_t, \nu_t, q_0) \mid \nu_t \in currtops\}$
repeat
$next := \emptyset$
for $(\nu, \nu_t, q) \in curr$ **do**
$at := ActTrans(q, \text{lookahead}, x)$
if $at = \emptyset$ **then** $at := ActTrans(q, \text{stack}, Symb(\nu))$
if $at = \{\text{goto } q'\}$ **then for** $\nu' \in Prev(\nu)$ **do** add (ν', ν_t, q') to *next*
else
if $0 \in at$ **then** add ν_t to *shifttops*
for $i \in at - \{0\}$ **do** $\nu_l := \text{leftmost}(i, \nu, \nu_t)$; add (i, ν_l, ν_t) to *reductions*
$curr := next$;
until $curr = \emptyset$

% Compute new tops from reductions
$currtops := \emptyset$
for $(i, \nu_l, \nu_t) \in reductions$ and $A \xrightarrow{i} \alpha$ **do**
if $\exists \nu_t' \in reduced$ and $Symb(\nu_t') = A$ and $Prev(\nu_t') = Prev(\nu_l)$ **then**
add (i, ν_l, ν_t) to $Deriv(\nu_t')$
else
$\nu_t' := NewNode(A)$; $Prev(\nu_t') := Prev(\nu_l)$; $Deriv(\nu_t') := \{(i, \nu_l, \nu_t)\}$
add ν_t' to *reduced, currtops*
until $currtops = \emptyset$

until $shifttops = \emptyset$
if $reduced = \{\nu\}$ and $Symb(\nu) = S'$ **then** accept **else** reject input % Syntactic error
end

Figure 3: Generalized discriminating-reverse parsing algorithm.

on Implementation and Applications of Automata, CIAA 2000, to appear in Lecture Notes in Computer Science. Springer-Verlag, 2000.

[3] J. Fortes Gálvez. A discriminating reverse automaton for LR(1) parsing. Research Report 91-23, Laboratoire I3S, Bât. 4, 250 Avenue Albert Einstein, F-06560 Valbonne, France, Oct. 1991.

[4] J. Fortes Gálvez. Generating LR(1) parsers of small size. In *Compiler Construction. 4th Internatinal Conference, CC'92*, Lecture Notes in Computer Science #641, pages 16–29. Springer-Verlag, 1992.

[5] J. Fortes Gálvez. Experimental results on discriminating-reverse LR(1) parsing. In P. Fritzson, editor, *Proceedings of the Poster Session of CC'94 - International Conference on Compiler Construction*, pages 71–80. Department of Computer and Information Science, Linköping University, Mar. 1994. Research report LiTH-IDA-R-94-11.

[6] J. Fortes Gálvez. *A Discriminating Reverse Approach to LR(k) Parsing.* PhD thesis, Universidad de Las Palmas de Gran Canaria and Université de Nice-Sophia Antipolis, 1998.

[7] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[8] D. E. Knuth. On the translation of languages from left to right. *Information and Control*, 8(6):607–639, 1965.

[9] M. J. Nederhof and J. J. Sarbo. Increasing the applicability of LR parsing. In H. Bunt and M. Tomita, editors, *Recent Advances in Parsing Technology*, pages 35–57. Kluwer Academic Publishers, 1996.

[10] R. Nozohoor-Farshi. Handling of ill-designed grammars in Tomita's parsing algorithm. In *International Workshop on Parsing Technologies*, pages 182–192, Carnegie Mellon University, Aug. 1989.

[11] S. Sippu and E. Soisalon-Soininen. *Parsing Theory*. Springer-Verlag, 1988–1990.

[12] M. Tomita. *Efficient Parsing for Natural Language*. Kluwer Academic Publishers, 1986.

[13] M. Tomita. The generalized parsing algorithm. In M. Tomita, editor, *Generalized LR Parsing*, pages 1–16. Kluwer Academic Publishers, 1991.