Available online at www.sciencedirect.com

ScienceDirect

Information Processing Letters 104 (2007) 195–199

Information Processing Letters

www.elsevier.com/locate/ipl

Some observations on LR-like parsing with delayed reduction

E. Bertsch [a], M.-J. Nederhof [b,*]

[a] *Ruhr University, Faculty of Mathematics, Universitätsstraße 150, D-44780 Bochum, Germany*
[b] *School of Computer Science, University of St Andrews, North Haugh, St Andrews, Fife, KY16 9SX, Scotland, UK*

Received 21 May 2007; received in revised form 20 June 2007

Available online 6 July 2007

Communicated by L. Boasson

Abstract

We discuss a bottom-up parsing technique based on delayed reductions, and investigate its capabilities and limitations. Some non-LR(k) grammars, for any k, are handled deterministically by this method. Surprisingly, and counter-intuitively from the viewpoint of LR(k), increase of delay may lead to decrease of determinism. We also present a variant that uses both delay and lookahead.

Keywords: Formal languages; Grammars; Parsing; LR(k)

1. Introduction

In a monograph that appeared in 1980, Marcus [1] described an innovative parsing method for natural languages that included various linguistically motivated features and the use of sentential buffers to delay parsing decisions in case of conflicts. Nozohoor-Farshi [2] applied Marcus' ideas to grammars of the kind studied in the theory of formal languages and commented on possible formalisations and generalisations. Leermakers [3] went a step further by precisely describing an item-based technique akin to that known for LR(k) grammars. To our knowledge, that article has been the only one that dealt with Marcus parsing in archival computer science journals. While Leermakers was also concerned with the usefulness of his work for linguists, he posed the challenge of describing the extent to which his formalisation of Marcus parsing leads to determinism. In particular, he assumed that by using more context, a parser "suffers from fewer reduce–reduce conflicts and is deterministic for more grammars". Our present study shows that this assumption cannot be upheld in its pure form, but it appears that a blend of delayed reduction and lookahead may be used to achieve a genuine increase of determinism over the LR(k) approach.

The Leermakers paper should be consulted for the theoretical background. He used a functional programming style. We provide a new description in very similar terms to those used in parser construction for programming languages [4]. Our designation of the parsing method and the terms referring to it will be "ML", for Marcus–Leermakers.

To introduce the problems that ML recognition and parsing address, let us consider the following fragment of a possible programming language syntax. Due to its nondeterminism in LR(k) terms, it cannot be expected to have appeared in any concrete grammar. It is well known that language designers prefer LR(1) constructs.

* Corresponding author.
E-mail address: markjan.nederhof@gmail.com (M.-J. Nederhof).

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2007.07.003

Our concern is to be able to deal with cases in which that precaution is disregarded.[1]

Statement $\rightarrow$ *ExprStatemt* | *Declaration*

ExprStatemt $\rightarrow$ *AccExpr* '[' *Expr* ']' | ...

AccExpr $\rightarrow$ *identifier* | ...

Declaration $\rightarrow$ *identifier* '[' *Expr* ':' *Expr* ']'

Expr $\rightarrow$ ...

The expressions separated by ':' might specify dynamically computable lower and upper bounds for arrays. The *ExprStatemt* refers to components of such arrays.

Having read the *identifier*, there is a shift–reduce conflict which cannot be resolved by any finite amount of lookahead. The decision is only possible after reading '[', reading and reducing to *Expr*, and examining the ensuing token, i.e., either ':' or ']'.

The ML solution is to use LR-like items including delay of reduction. Thus instead of the problematic item [*AccExpr* $\rightarrow$ $\bullet$ *identifier*], with appropriate ML delay there will be an item [*AccExpr* '[' *Expr* ']' $\rightarrow$ $\bullet$ *identifier* '[' *Expr* ']'], with a left-hand side of length 4. Its reduction is not performed until all of its right-hand side has been found and the item [*AccExpr* '[' *Expr* ']' $\rightarrow$ *identifier* '[' *Expr* ']' $\bullet$], with the dot at the end, is an element of the current state. The right-hand side is then replaced by its left-hand side essentially as in usual LR(k) recognition, with a Goto action over more than one symbol.

ML parsing is similar to some techniques of non-canonical LR parsing [6–8]. Our approach differs from them by needing only a single pushdown stack.

2. State construction

The following procedures are strongly reminiscent of LR(k) state construction. We presume that in view of the precise treatment in [3], no formal proof of correctness must be given here. It is a straightforward extension of LR theory.

Let $G = (V, T, P, S)$ be a context-free grammar. Fix integer $k \geqslant 0$. We define a k-item for G as an entity of the form $[A\alpha \rightarrow \beta \bullet \gamma]$ such that for some rule $A \rightarrow \delta \in P$, $\delta\alpha = \beta\gamma$, and $|\alpha| \leqslant k$. Informally, $\beta\gamma$ consists of the right-hand side of a rule for A and the context α of A that is also represented in the left-hand side of the item.

Further, the notation $k : \beta$ designates the prefix of length k of β if $|\beta| > k$, and all of β otherwise. Overloading the colon operator, the rest of string β behind $k : \beta$ is designated as $\beta : k$. Thus $\beta = (k : \beta)(\beta : k)$.

Define a function $close(Q)$, where Q is a set of k-items, to return Q' computed by:

$Q' := Q$
repeat
  for all $[\alpha \rightarrow \beta \bullet B\gamma] \in Q'$ and $B \rightarrow \delta \in P$
    add $[B(k : \gamma) \rightarrow \bullet\, \delta(k : \gamma)]$ to Q'
until nothing new added to Q'
return Q'

Define a set of items $q_0 := close([S^\dagger \rightarrow \bullet\, S\#^k])$, where # is a distinguished symbol that acts as end-of-sentence marker and $S^\dagger$ is a new symbol. The set *States* of ML states is computed by:

$States := \{q_0\}$
repeat
  for all $Q \in States$
    for all $\delta \in \{a \in T \mid [\alpha \rightarrow \beta \bullet a\gamma] \in Q\}$
        $\cup \{\zeta \mid [\zeta \rightarrow \bullet\, \eta] \in Q\}$
      $Goto(Q, \delta) := close(\{[\alpha \rightarrow \beta\delta \bullet \gamma] \mid$
          $[\alpha \rightarrow \beta \bullet \delta\gamma] \in Q\})$
      add $Goto(Q, \delta)$ to *States*
until nothing new added to *States*

This simultaneously defines $Goto(Q, \delta) \neq \emptyset$ for appropriate choices of $Q \in States$ and strings $\delta \in V^*$. Here δ may consist of several symbols and thereby the dot may traverse over several symbols in one goto action. This requires a reduction for each item $[\alpha \rightarrow \beta\, \bullet]$ that includes $pop\_length(\beta)$ pop steps, where $pop\_length(\beta)$ is defined recursively as the following (ε stands for the empty string):

if $\beta = \varepsilon$ **return** 0
else if $1 : \beta \in T$ **return** $1 + pop\_length(\beta : 1)$
else return $1 + pop\_length(\beta : (k + 1))$

As an example, if we assume $k = 1$ and $\beta = ABcDeFG$, then *pop_length* is called recursively for $ABcDeFG$, $cDeFG$, $DeFG$, FG, ε, and the returned value is 4. Note that where terminal symbol c is the first symbol, nothing else is cut off.

Declare *States* as deterministic for ML(k, 0) recognition if states containing an item $[\alpha \rightarrow \beta\, \bullet]$ contain no

[1] For interesting background reading, Section 19.1 "Grammatical difficulties" of the original Java(TM) Language Specification [5] is recommended. It describes "problems" that the Java designers encountered with LALR(1), and the syntactical changes they were forced to administer.

$i := 0$
repeat
  if state Q on top of stack contains
    item of form $[\alpha \to \beta \bullet]$
   pop $pop\_length(\beta)$ states
   let Q' be the new state on top
   push $Goto(Q', \alpha)$
  else if $Goto(Q, a_i)$ is undefined
   report failure and halt
  else
   push $Goto(Q, a_i)$
   $i := i + 1$
until top of stack is $\{[S^\dagger \to S\#^k \bullet]\}$
report success

Fig. 1. Parse time procedure for ML(k, 0) grammars.

other items. We then call the grammar ML(k, 0), and the parse time procedure is as sketched in Fig. 1. This assumes the input is extended on the right by k copies of #. The input symbols, including the end-of-sentence markers, are named $a_1, a_2, \ldots$.

As an example, consider the following ML(1, 0) grammar $\mathcal{G}_1$.

$S \to A\ B \mid A'\ C$

$A \to a$

$A' \to a$

$B \to D\ b$

$C \to D\ c$

$D \to d \mid e\ D$

With $k = 1$ we obtain a deterministic set *States* containing 16 states, which we cannot all show due to length restrictions. The most critical state is $q_1 = Goto(q_0, a) = close(\{[A\ B \to a \bullet B], [A'\ C \to a \bullet C]\})$. By the closure, q_1 also contains $[B \to \bullet\ D\ b]$, $[C \to \bullet\ D\ c]$, $[D\ b \to \bullet\ d\ b]$ and $[D\ c \to \bullet\ d\ c]$, and $[D\ b \to \bullet\ e\ D\ b]$ and $[D\ c \to \bullet\ e\ D\ c]$. In the ML parser, the choice between $A \to a$ and $A' \to a$ is effectively postponed until either b or c is read.

Note that an unbounded number of terminal symbols in the input separate the a from either b or c. It is clear that the set of LR(m) states cannot be deterministic for any m, because of a reduce–reduce conflict involving $A \to a$ and $A' \to a$. In particular, the grammar is not LR(0), which concurs with ML(0, 0).

By an increase of k, the context can be extended. Fix $j > 1$ and consider the grammar $\mathcal{G}_2^j$ given by:

$S \to A\ D^{j-1}\ B \mid A'\ D^{j-1}\ C$

$A \to a$

$A' \to a$

$B \to D\ b$

$C \to D\ c$

$D \to d \mid e\ D$

The grammar $\mathcal{G}_2^j$ is ML(j, 0) but not ML($j-1$, 0). (In fact, it is ML(k, 0) iff $k \geqslant j$.) Now $q_1 = Goto(q_0, a) = close(\{[A\ D^{j-1}\ B \to a \bullet D^{j-1}\ B], [A'\ D^{j-1}\ C \to a \bullet D^{j-1}\ C]\})$, and q_1 also contains $[D^{j-1}\ B \to \bullet\ d\ D^{j-2}\ B]$ and $[D^{j-1}\ C \to \bullet\ d\ D^{j-2}\ C]$, and $[D^{j-1}\ B \to \bullet\ e\ D^{j-1}\ B]$ and $[D^{j-1}\ C \to \bullet\ e\ D^{j-1}\ C]$. As before, the choice between $A \to a$ and $A' \to a$ is effectively postponed until either b or c is read.

3. ML with lookahead

The limitations of delayed reduction can be witnessed in the following sLR(1) grammar $\mathcal{G}_3$, which is not ML(k, 0) for any k.

$S \to A \mid S\ A$

$A \to a \mid a\ b$

Assume a fixed k. The ML(k, 0) state reached after reading an initial a contains amongst others all items of the form $[AA^j\#^{k-j} \to a \bullet A^j\#^{k-j}]$ and $[AA^j\#^{k-j} \to a \bullet bA^j\#^{k-j}]$ where $0 \leqslant j \leqslant k$. Context of length $k > 0$ following a or ab avoids a shift–reduce conflict. However, this context becomes shorter by one symbol for each subsequent a until the state $\{[A \to a \bullet], [A \to a \bullet b]\}$ is reached, and the conflict becomes unavoidable.

In order to allow grammars such as the above to be handled deterministically within the ML framework, we introduce lookahead. This leads to ML(k, m) parsing, which reduces to LR(m) parsing for $k = 0$.

We first define $First_m(\beta) = \{m : w \mid \beta \Rightarrow^* w\}$, for each string β of terminals and nonterminals. Here $\Rightarrow^*$ stands for derivation (of a terminal string) in one or more steps.

Next, we extend the concept of k-items to (k, m)-items, by an additional component referring to the (terminal) context behind the end of the right-hand side, as done in the case of LR(m). The initial state is now $q_0 := close([S^\dagger \to \bullet\ S\#^k, \varepsilon])$, and the closure operation is refined to:

$Q' := Q$
repeat
  for all $[\alpha \to \beta \bullet B\gamma, x] \in Q'$ and
    $B \to \delta \in P$ and $y \in First_m((\gamma : k)x)$
   add$[B(k : \gamma) \to \bullet\ \delta(k : \gamma), y]$ to Q'
until nothing new added to Q'
return Q'

As usually, the lookahead component is used to decrease the number of shift–reduce and reduce–reduce conflicts. If none remain, we say the grammar is ML(k, m), which generalises our previous definition of ML(k, 0) in a natural way.

We mention in passing that analogues of sLR(m) and LALR(m) can be introduced as well. For example, sML(1, 1) might designate the case that ML(1, 0) states allow resolution of conflicts by checking one symbol of terminal lookahead against the 'follow' sets of left-hand sides of candidate items, which can now consist of several grammar symbols.

Those example grammars from [6,7,2] that generate deterministic languages are all ML(1, 1) but mostly not ML(1, 0). There are non-ML(k, m) grammars of deterministic languages, for all k and m, that can be recognised noncanonically by means of two stacks, such as the first grammar from [8].

With lookahead, our observations about the family of grammars $\mathcal{G}_2^j$ at the end of Section 2 can be refined: grammar $\mathcal{G}_2^j$ is ML(k, m) iff $k \geqslant j$ irrespective of m. Lookahead is ineffective here due to the arbitrarily long string separating a from either b or c in B or C, respectively.

4. Non-monotonicity

The class of LR(m) grammars is properly contained in the class of the LR($m + 1$) grammars [9]. More generally, the class of ML(k, m) grammars is properly contained in the class of the ML(k, $m + 1$) grammars, for any fixed k. The behaviour of k is not monotone however. A first illustration of this is the grammar $\mathcal{G}_4$:

$$S \to a \mid SSSb$$

This grammar is ML(0, 0) but not ML(1, 0), which is explained as follows. For $k = 1$, q_0 includes $[S\# \to \bullet\, SSSb\#]$, and by closure also $[SS \to \bullet\, aS]$. By a shift with a, we obtain state q_1 that includes $[SS \to a \bullet S]$, and by closure also $[S \to \bullet\, a]$ and $[S \to \bullet\, SSSb]$, as well as $[SS \to \bullet\, aS]$. By a further shift with a, we obtain state q_2 that includes $[S \to a\, \bullet]$ and $[SS \to a \bullet S]$, and by closure also $[S \to \bullet\, a]$, so that a shift–reduce conflict occurs.

The situation remains unchanged if we choose $m > 0$. The two relevant items above correspond to rule occurrences immediately to the left of an occurrence of S, and $a^m \in \mathit{First}_m(S)$, so that $q_2 = \mathit{Goto}(\mathit{Goto}(q_0, a), a)$ includes amongst others $[S \to a\, \bullet, a^m]$ and $[S \to \bullet\, a, a^m]$. This implies that a shift–reduce conflict occurs for following input a^m, and thereby the grammar is not ML(1, m) for any m.

An example of oscillating behaviour is witnessed for the grammar $\mathcal{G}_5$:

$$S \to c \mid SdA$$

$$A \to a \mid ab$$

This grammar is ML(k, 0) iff k is odd. For even k, q_0 consists of items $[S\#^k \to \bullet\, c\#^k]$ and $[S\#^k \to \bullet\, SdA\#^k]$, and by closure also items of the form $[S(dA)^j\#^{j'} \to \bullet\, c(dA)^j\#^{j'}]$ and $[S(dA)^j\#^{j'} \to \bullet\, SdA(dA)^j\#^{j'}]$, for $j = 1, \ldots, k/2$ and $j' = k - 2j$. For $k = 0$, or for $k > 0$ and $j = k/2$, A appears at the end of a right-hand side, and the absence of right context eventually leads to a shift–reduce conflict. For odd k however, A is always followed by either d or #, so that a conflict is avoided.

We can have arbitrary behaviour of the ML(k, m) property relative to a finite selection of positive values of k. Let X be a finite set of positive integers. Define the grammar $\mathcal{G}_6^X$ with the set of terminals $\{a, b, c, d, f\} \cup \{a_j \mid j \in X\}$, the set of nonterminals $\{S, A, B, C, D, E, F\} \cup \{A_j \mid j \in X\} \cup \{B_j \mid j \in X\}$ and the rules:

$$S \to a_j A_j A, \quad \text{for each } j \in X$$

$$S \to a_j B_j B, \quad \text{for each } j \in X$$

$$A_j \to Cd^{j-1}D, \quad \text{for each } j \in X$$

$$B_j \to Cd^{j-1}E, \quad \text{for each } j \in X$$

$$A \to Fa$$

$$B \to Fb$$

$$C \to c$$

$$D \to d$$

$$E \to d$$

$$F \to f \mid fF$$

This grammar is ML(k, m) iff $k \notin X$, for k positive and any m. Choose a fixed $k \in X$, and let $m = 0$. A shift with a_k, then a shift with c, and $k - 1$ shifts with d lead to a state consisting of $[Cd^{k-1}D \to cd^{k-1} \bullet D]$, $[Cd^{k-1}E \to cd^{k-1} \bullet E]$, $[D \to \bullet\, d]$ and $[E \to \bullet\, d]$. After another shift with d, a reduce–reduce conflict occurs. If we choose $m > 0$, this does not avoid the above conflict, as an occurrence of either a or b, needed to distinguish between the two cases, can be preceded by an arbitrarily long string of f's.

Such conflicts are avoided for any positive k not in X however, as occurrences of D and E in items are then always followed by right contexts starting with A and B, respectively.

5. Descriptional complexity

ML parsers can be more compact than LR parsers for LR(m) grammars. Consider the family of grammars $\mathcal{G}_7^j$ of the form:

$$S \to ACa \mid BCb \mid eSe \mid fSf$$

$$A \to d$$

$$B \to d$$

$$C \to c^j$$

where $j \geqslant 1$. These grammars are ML(1, 1) irrespective of j. Delayed reduction with context C and lookahead a or b are sufficient to allow a deterministic choice between $A \to d$ and $B \to d$. The size of the ML parser remains linear in j. However, for $k = 0$, m needs to be at least $j + 1$ to make the grammar ML(0, m), i.e., LR(m). Then for each string $x \in \{e, f\}^m$ there is, amongst others, a state consisting of $[S \to eSe\,\bullet; x]$, and the parser will have size exponential in j.

A more general result is obtained by replacing $S \to A\ C\ a$ and $S \to B\ C\ b$ in the above grammar by $S \to A\ C^i\ a$ and $S \to B\ C^i\ b$, respectively, for some $i \geqslant 1$. This grammar is ML(i, 1), with a parser of linear size in j, and ML($i - 1$, m) iff $m > j$, and for $m = j + 1$ the parser has exponential size in j.

6. Concluding remarks

The algorithms presented in this paper were implemented, and the discussed examples were all checked against this implementation.

Concerning classes of grammars that are ML(k, m), we can reach the following conclusions:

- For any k and m, ML(k, m) is properly contained in ML(k, $m + 1$).
- For any k, ML($k + 1$, 0) contains grammars not in ML(k, m) for any m.
- For any k, ML(k, 0) contains grammars not in ML($k + 1$, m) for any m.
- A ML(k, m) parser may be exponentially less compact than a ML($k + 1$, m') parser, where m and m' are the minimal values needed for determinism.

That the ML property is not monotone in k has far-reaching consequences. By increasing $k > 0$, some conflicts can be avoided that would occur in LR(m) parsers, but at the same time, fresh conflicts may arise elsewhere.

A subject for further study is whether delayed reduction can be used selectively. In this article, the context to delayed reduction has a uniformly determined maximal length. Selectivity would mean that for some states a longer context is chosen to resolve shift–reduce or reduce–reduce conflicts. This leads to a new range of constructional possibilities.

As our formulation of ML parsing is based on a machine model with a single stack, it is straightforward to apply dynamic programming techniques to handle non-ML grammars, in the sense first explored by [10]. For a recent study of non-deterministic constructs in programming language grammars, cf. [11].

Acknowledgements

The contributions of an anonymous reviewer are gratefully acknowledged.

References

[1] M. Marcus, A Theory of Syntactic Recognition for Natural Language, MIT Press, 1980.
[2] R. Nozohoor-Farshi, On formalizations of Marcus' parser, in: 11th International Conference on Computational Linguistics, University of Bonn, Bonn, 1986, pp. 533–535.
[3] R. Leermakers, Recursive ascent parsing: from Earley to Marcus, Theoretical Computer Science 104 (1992) 299–312.
[4] A. Aho, M. Lam, R. Sethi, J. Ullman, Compilers: Principles, Techniques, & Tools, Addison-Wesley, 2007.
[5] J. Gosling, B. Joy, G. Steele, The Java Language Specification, Addison-Wesley, 1996.
[6] T. Szymanski, J. Williams, Noncanonical extensions of bottom-up parsing techniques, SIAM Journal on Computing 5 (1976) 231–250.
[7] K.-C. Tai, Noncanonical SLR(1) grammars, ACM Transactions on Programming Languages and Systems 1 (1979) 295–320.
[8] S. Schmitz, Noncanonical LALR(1) parsing, in: O. Ibarra, Z. Dang (Eds.), Developments in Language Theory, 10th International Conference, in: Lecture Notes in Computer Science, vol. 4036, Springer-Verlag, Santa Barbara, CA, USA, 2006, pp. 95–107.
[9] S. Sippu, E. Soisalon-Soininen, Parsing Theory, vol. II: LR(k) and LL(k) Parsing, EATCS Monographs on Theoretical Computer Science, vol. 20, Springer-Verlag, 1990.
[10] B. Lang, Deterministic techniques for efficient non-deterministic parsers, in: Automata, Languages and Programming, 2nd Colloquium, in: Lecture Notes in Computer Science, vol. 14, Springer-Verlag, Saarbrücken, 1974, pp. 255–269.
[11] E. Scott, A. Johnstone, Right nulled GLR parsers, ACM Transactions on Programming Languages and Systems 28 (2006) 577–618.