基于状态回朔技术的规范LR分析算法

王强华 侯 红 耿国华
(西北大学信息科学与技术学院 西安 710069)

摘 要 LR分析技术以其自身的优点在实际当中有着非常广泛的应用,但是,能够识别LR(1)语言的规范LR分析器由于其下推自动机的复杂性,其实用性受到比较大的限制。通过回朔下推自动机的状态迁移路径能够从根本上解决这一问题。主要讨论了基于状态回朔技术的规范型LR分析器的基本原理与构造技术。

关键词 LR分析,规范型LR分析器,Yacc
中图法分类号 TP301.2 **文献标识码** A

Canonical LR Parsing Algorithm Based on State Backtracking Technique

WANG Qiang-hua HOU Hong GENG Guo-hua
(School of Information Science and Technology, Northwest University, Xi'an 710069, China)

Abstract With its excellent characteristics, LR analysis technique has gained extensive use in pratical. But the practicability of the canonical LR parser, recognizer for the LR(1) language, is strictly limited for the sake of its push down automaton's complexity. Taking advantage of state backtracking technique, we can solve this problem radically. The principles and constructing techniques of the canonical LR parser based on states backtracking technique were presented.

Keywords LR parsing, Canonical LR parser, Yacc

在实际构造LR(1)分析器的过程中,为了减少下推自动机的状态数,通常对具有相同LR(0)项目集(同核)的状态进行合并,这种合并对LR(1)文法可能会引入"归约-归约"冲突[7,8],针对这一问题,有两种典型策略,第一是仅合并不会产生冲突的状态,文献[1]介绍了该策略的一种实现方法,由于合并效果较差,其实用性受到了比较大的限制;第二种策略是一种较为常见的策略,其基本思想是对分析过程中的冲突采用不确定性原则,分析器沿着各种可能的路径前进,直到达到一个确定的结果,这种方法的主要缺点是实现不确定下推自动机所引入的复杂性,分析器需要管理多重分析栈及上下文环境,其优点是可适用于二义性文法,对此文献[2-4]都给出了较为详细的论述,该策略已被应用于yacc[9]工具basil[6]和btyacc[5]的实现中。

本文介绍一种基于状态回朔技术的规范LR分析器算法,该算法通过回朔分析过程中的状态迁移路径来构造归约状态下确定句柄的前看符号集,避免用LR(1)项目区分前看符号集或者用不确定自动机解决冲突的做法。该分析器能够识别LR(1)文法语言,具有与SLR和LALR分析器相同的状态数目,其分析表易于构造,优化后执行效率能够接近SLR和LALR分析器的水平,具有较好的实用性。

1 基本思想

分析LR分析器的模型,考察任意拓广[7]LR(1)文法G',设$p:A\to X_1X_2\cdots X_n$(设$n=0$时,$X_1X_2\cdots X_n=\varepsilon$)是$G'$中的产生式,根据识别活前缀DFA的构造方法,识别$G'$活前缀的DFA包含图1所示的状态迁移路径片段:

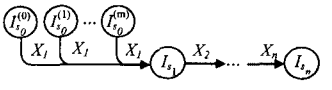

图1 DFA中的状态迁移路径片断

图中,状态$I_{s_0}^{(r)}$($r=0,1,\cdots$)的项目集中存在核项目$[B\to\gamma\cdot C\delta]$,且$C\overset{*}{\underset{rm}{=\!>}}A\eta$,状态$I_{s_i}$($1\leqslant i\leqslant n$)的项目集中包含核项目$[A\to X_1\cdots X_i\cdot\cdots X_n]$,分析器在$I_{s_n}$下可以执行$p$的规约,能够激励这一规约的前看符号的集合决定了分析器的类型。在SLR中,该前看符号集是所有$S'\overset{*}{\underset{rm}{=\!>}}\alpha Aw$的最右推导中,能够紧跟在$A$之后的终结符的集合;在LALR中,是所有满足$S'\overset{*}{\underset{rm}{=\!>}}\alpha Aw\underset{rm}{=\!>}\alpha X_1X_2\cdots X_nw$,$\alpha X_1X_2\cdots X_n$是状态$I_{s_n}$识别的活前缀的最右推导中,能够紧跟在$A$之后的终结符的集合;在规范型LR中,是在$\alpha X_1X_2\cdots X_n$一定的条件下,所有满足上述条件的最右推导中,能够紧跟在A之后的终结符的集合。显然,规范型LR的前看符号集最精确,因此,其识别能力最强,但该前看符号集也因进入状态I_{s_n}时所经过的$I_{s_0}^{(r')}$不同而不同,需要在I_{s_n}下能够确定该符号集。通常的做法是为每一个不同的前看符号集单独复制一条I_{s_1}至I_{s_n}的状态迁移

到稿日期:2008-05-13 本文受国家自然科学基金重点项目(No. 60736008),西北大学博士科研启动基金(No. kyqdf192)资助。
王强华(1965-),男,博士,讲师,主要研究领域为计算理论、并行计算、分布式计算,E-mail:wqh@nwu.edu.cn,wqh88@sina.com.cn;侯 红(1967-),女,博士,副教授,主要研究领域为软件工程、数据挖掘;耿国华(1955-),女,博士,教授,博士生导师,主要研究领域为智能信息处理、模式识别。

路径，其结果是下推自动机的状态数急剧增大。实际上，根据下推自动机的原理，当分析器进入 I_{s_n} 状态后，其最近一次经过的 $I_{s_0}^{(r')}$ 状态记录在分析栈中当前栈顶位置下的第 $2n+1$ 单元处，因此，我们可以通过回朔分析栈来确定这一状态，由此得出 p 的前看符号集，并将其中的符号作为“reduce p”动作的输入激励加入到当前动作表中，以动态构造当前状态的动作表，为此，我们需要改进分析表的结构和分析算法。

2 分析表的构造

图 2 是分析表的结构。改进后的分析表在原来的结构上增加了 reduction 和 lookaheads 栏，其中，reduction 是当前状态下需要动态确定其规约激励的产生式的集合，lookaheads 是当前状态下能够静态确定的所有非核项目的产生式的前看符号集。

STATE	*action*			*goto*			*reduction*	*lookaheads*		
	a_0	a_1	……	A_0	A_1	……		p_0	p_1	……

图 2 分析表结构

分析表的构造方法如下：

1. 构造 G' 的 LR(0)项目集的集合 $C=\{I_0, I_1, \cdots, I_n\}$，为每一 I_i 建立状态 i。

2. 分析表初始为空，状态 i 的内容按以下规则确定：

a)如果$[A\rightarrow\alpha\cdot a\beta]$是 I_i 中的项目，且 $goto(I_i, a)=I_j$，则置 $action[i,a]$为“shift j”；

b)如果$[p: A\rightarrow\alpha\cdot]$是 I_i 中的项目，且 A 不是 S'，则将 p 加入 $reduction[i]$中；

c)如果$[S'\rightarrow S\cdot]$是 I_i 中的项目，则置 $action[i, \$]$为“*accept*”；

d)对所有满足 $goto(I_i, A)=I_j$ 的 A，置 $goto[i,A]=j$；

e)如果$[p: A\rightarrow\cdot\alpha]$是 I_i 中的非核项目，则置 *lookaheads* $[i,p]=ls$，ls 是按如下方式计算的 p 的前看符号集：$ls=\phi$，对 I_i 中所有核项目$[q: B\rightarrow\gamma\cdot C\delta]$，若 $C\overset{*}{\underset{m}{\Rightarrow}}A\eta$ 则

$$ls=\begin{cases} ls\cup FIRST(\eta\delta\Delta_{q,|\gamma|}) & B\neq S' \\ ls\cup FIRST(\eta\delta\$) & B=S' \end{cases}$$

其中，$\Delta_{q,|\gamma|}$ 是 q 的前看符号集引用符，$|\gamma|$ 是 γ 串的长度，\$ 是输入结束符。

3. 分析器的初态是包含$[S'\rightarrow\cdot S]$的 I_i 所对应的状态 i。

图 3 给出了一个示例性文法[7]及其 LR(0)项目集，图 4 是识别该文法活前缀的 DFA 的状态图，图 5 是在此基础上按照上述方法构造的分析表，表中 sj 表示“*shift j*”，*acc* 表示“*accept*”。

$p_0: S'\rightarrow S$	$I_0: S'\rightarrow\cdot S$	$I_4: L\rightarrow *\cdot R$
$p_1: S\rightarrow L=R$	$S\rightarrow\cdot L=R$	$R\rightarrow\cdot L$
$p_2: S\rightarrow R$	$S\rightarrow\cdot R$	$L\rightarrow\cdot *R$
$p_3: L\rightarrow *R$	$L\rightarrow\cdot *R$	$L\rightarrow\cdot$ **id**
$p_4: L\rightarrow$ **id**	$L\rightarrow\cdot$ **id**	$I_5: L\rightarrow$ **id** $\cdot$
$p_5: R\rightarrow L$	$R\rightarrow\cdot L$	$I_6: S\rightarrow L=\cdot R$
	$I_1: S'\rightarrow S\cdot$	$R\rightarrow\cdot L$
	$I_2: S\rightarrow L\cdot=R$	$L\rightarrow\cdot *R$
	$R\rightarrow L\cdot$	$L\rightarrow\cdot$ **id**
	$I_3: S\rightarrow R\cdot$	$I_7: S\rightarrow L=R\cdot$
		$I_8: L\rightarrow *R\cdot$
		$I_9: R\rightarrow L\cdot$

图 3 示例文法及其 LR(0)项目集

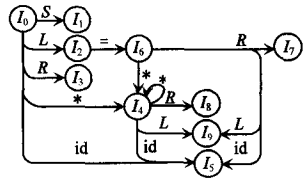

图 4 DFA 状态图

STATE	action				goto			reduction	lookaheads					
	=	*	**id**	$	S	L	R		p	p_1	p_2	p_3	p_4	p_5
0		s4	s5		1	2	3		{$}	{$}	{=,$}	{=,$}	{$}	
1				acc										
2	s6							{p5}						
3								{p2}						
4		s4	s5		9	8						$\{\Delta_{p_3},1\}$	$\{\Delta_{p_3},1\}$	$\{\Delta_{p_3},1\}$
5								{p4}						
6		s4	s5		9	7						$\{\Delta_{p_1},2\}$	$\{\Delta_{p_1},2\}$	$\{\Delta_{p_1},2\}$
7								{p1}						
8								{p3}						
9								{p5}						

图 5 分析表

3 分析算法

改进后的分析算法如下：

初始条件：分析栈中包含初态 s_0，终结符串 w\$ 在输入缓冲区中，ip 指向第一个输入符号；

```
repeat forever begin
  let s = 当前栈顶中的分析器状态, a = ip 指向的输入符号;
  let current_action = action[s];
  if not empty reduction[s] then
      for each A→α in reduction[s] begin
          merge(current_action, |α|, A→α, reduceA→α);
          end
      if current_action[a] = shift s' then begin
      push(a); push(s');
      ip= ip+1;
      end
  else if current_action[a] = reduceA→α then begin
          pop(|α|×2 个元素);
          let s' =当前栈顶中的分析器状态;
          push(A); push(goto[s', A]);
          执行 A→α 的语义子程序;
          end
  else if current_action[a] = accept then
          return;
          else error();
end
```

上述算法中调用了一个关键的例程：*merge*，该例程回朔分析栈，取出给定产生式的前看符号集，将其中符号作为指定动作的输入激励加入当前动作表 *current_action* 中，*merge* 的伪代码如下：

调用参数：current_action-当前动作表，n-回朔状态数，p-产生式，op-动作；

（下转第 198 页）

参 考 文 献

[1] Pawlak Z. Rough sets[J]. International Journal of Computer and Information Sciences, 1982, 11: 341-356

[2] Pawlak Z. Rough Sets: Theoretical Aspects of Reasoning about Data[M]. Boston: Kluwer Academic Publishers, 1991

[3] Pei D W. A generalized model of fuzzy rough sets[J]. International Journal of General Systems, 2005, 34: 603-613

[4] Wu W-Z, Zhang W-X. Constructive and axiomatic approaches of fuzzy approximation operators[J]. Information Sciences, 2004, 159: 233-254

[5] Yao Y Y. Constructive and algebraic methods of the theory of rough sets [J]. Journal of Information Sciences, 1998, 109: 21-47

[6] Yao Y Y. Relational interpretations of neighborhood operators and rough set approximation operators[J]. Information Sciences, 1998, 111: 239-259

[7] Yao Y Y. Two views of the theory of rough sets in finite universes[J]. International Journal of Approximate Reasoning, 1996, 15: 291-317

[8] 张文修，梁怡，吴伟志. 信息系统与知识发现有[M]. 北京：科学出版社，2003

(上接第 180 页)

```
merge(current_action,n,p,op) begin
  let s′ = 当前栈顶下第 2n+1 单元中的分析器状态, ls = lookaheads[s′,p];
  for each a in ls begin
      if a = Λq,m then
          merge(current_action,n+m,q,op);
      else if current_action[a]≠nil and current_action[a]≠op then
              error();
      else  current_action[a] = op;
      end
  end
```

从上述伪代码中可以看出，*merge* 是一个递归过程，这是由于产生式的前看符号集中可能包含对上级产生式前看符号集的引用。另外，由于当前动作表 *current_action* 的动态性，对文法自身的移动-规约、规约-规约冲突检查无法像通常那样在构造分析表时完成，只能在分析具体输入串构造 *current_action* 时实现，反映在上述代码中对 *error* 例程的调用。在具体实现中，文法的合法性可以由独立的文法检查阶段完成，毕竟，文法本身一经确定其可变性非常小。

图 6 所示的是分析算法基于图 5 的分析表对示例性输入 * **id** = * **id** 的分析过程，表中 *rj* 表示“reduction p_j”，由于图 3 所给的示例性文法是 LALR 文法[1]，下图也列出了其 LALR 分析表的 *action* 内容以示比较，从图中 9—12 步可以看出，current_action 比 LALR action 呈现出更加精确的特性。

STEP	STACK	INPUT	current_action				LALR action			
			=	*	id	$	=	*	id	$
(1)	0	* **id** = * **id**		s4	s5			s4	s5	
(2)	0 * 4	$		s4	s5			s4	s5	
(3)	0 * 4 **id** 5	**id** = * **id** $	r4			r4	r4			r4
(4)	0 * 4 L 9	= * **id** $	r5			r5	r5			r5
(5)	0 * 4 R 8	= * **id** $	r3			r3	s3			r3
(6)	0 L 2	= * **id** $	s6			r5	s6			r5
(7)	0 L 2 = 6	= * **id** $		s4	s5			s4	s5	
(8)	0 L 2 = 6 * 4	* **id** $		s4	s5			s4	s5	
(9)	0 L 2 = 6 * 4 **id** 5	**id** $				r4	r4			r4
(10)	0 L 2 = 6 * 4 L 9	$				r5	r5			r5
(11)	0 L 2 = 6 * 4 R 8	$				r3	r3			r3
(12)	0 L 2 = 6 L 9	$				r5	r5			r5
(13)	0 L 2 = 6 R 7	$				r1				r1
(14)	0 S 1	$				acc				acc
		$								

图 6 * **id** = * **id** 的分析过程

4 分析表的优化

上述分析算法在执行效率上的损失主要缘于其动作表的动态性，这种损失可以通过优化加以改善。实际上，对于那些前看符号集不受状态迁移路径影响的产生式来说，在构造分析表时，可直接将其规约动作和输入激励填入到相应的 *action* 中，避免执行时动态构造，因此，分析表中的 *action*，*reduction* 分别可以看作是动作表的静、动两方面的描述，可以通过在构造分析表时尽可能增加 *action* 的内容而减少 *reduction* 的内容来实现分析表的优化，优化后分析器的执行效率有望能接近甚至达到 SLR 和 LALR 的水平。

结束语 本文介绍了一种规范 LR 分析算法，该算法区别于其他 LR 分析算法的本质特点在于其识别活前缀 DFA 中归约状态下动作表的不确定性，这种不确定性的根本原因在于对 DFA 中状态迁移路径复用，而复用正是实现简化 DFA 的最根本的手段。如上所述，通过在可复用的前提下尽可能减少其不确定性能够在简化和执行效率之间达到最好的平衡。

参 考 文 献

[1] 金毅，陆蓓，王小华. 一种较少状态数的 LR 分析器. 杭州电子科技大学学报，2006，26(3)：74-77

[2] Adrian D, James R. A backtracking LR algorithm for parsing ambiguous context-dependent languages // Proceedings of the 2006 Conference of the Center for Advanced Studies on Collaborative research. Oct. 2006

[3] McPeak S, George C, Elkhound. A fast, practical GLR parser generator // Compiler construction; 13th International Conference (CC' 04), Volume 2985 of Lecture Notes in Computer Science. April 2004

[4] Grosch J. Lark-An LALR(2) parser generator with backtracking. Technical Report 32, CoCoLab-Datenverarbeitung, Sep. 2002

[5] Dodd C, Maslov V. Backtracking Yacc, 2006. http://www.siber.com/btyacc/

[6] Spencer M. Basil: A backtracking LR parser generator, 2006. http://www.lazycplusplus.com/basil/

[7] Aho A V, Sethi R, Ullman J D. Compilers: Principles, Techniques, and Tools. Addison Wesley, Pearson Education, Inc., 1986

[8] Hopcroft J E, Ullman J D. Introduction to Automata Theory, Languages, and Computation. Addison Wesley, Reading, Mass., 1979

[9] Levine J R, Mason T, Brown D. Lex & Yacc. O'Reilly & Associates, Inc., *1992*

[10] Scott M L. Programming Language Pragmatics. Elsevier Inc., 2000