第 33 卷 第 4 期 计 算 机 工 程 2007 年 2 月
Vol.33 No.4 Computer Engineering February 2007
·人工智能及识别技术· 文章编号：1000—3428(2007)04—0193—03 文献标识码：A 中图分类号：TP311

一种 LR 语法分析中的错误恢复方法

肖 洋，姜淑娟
(中国矿业大学计算机科学与技术学院，徐州 221008)

摘 要：语法分析中的错误恢复是现代编译器中智能感知功能的重要组成部分，错误恢复的效果直接影响到智能感知功能的性能。在分析 LR 语法分析中 LR 分析表特性的基础上，提出了一种对 LR 分析表中的 Goto 表项进行改造来进行语法错误的诊断和恢复的方法。该方法充分利用了 LR 分析表中的空闲表项，在不增加空间需求的情况下，提高了语法错误的诊断和恢复的效率和准确率。

关键词：语法分析；编译器；智能；LR 语法分析表；Action-Goto 表

An Error Recovery Approach in LR Syntax Analysis

XIAO Yang, JIANG Shujuan
(School of Computer Science & Technology, China University of Mining & Technology, Xuzhou 221008)

【Abstract】 Error recovery in syntax analysis is one of the most important parts of the intelligence in the compiler. The effect of error recovery influences intelligence and syntax analysis. This paper presents a new method of modifying the items of Goto table in correcting syntax error based on analyzing the items of LR syntax analysis table. It uses the leisure items in LR syntax analysis table. It can improve the efficiency and precision in diagnosing and correcting syntax error without requiring more space.

【Key words】 Syntax analysis; Compiler; Intelligence; LR syntax analysis table; Action-Goto table

编译器在进行语法分析时经常采用LR分析法[1~5]，传统的LR分析表项比较稀疏，没有能够对语法错误进行有效的诊断和恢复。本文借鉴了对Action表的改造方法[3]，实现了对某些语法错误进行诊断和恢复，并结合智能感知，提出了对Goto表进行改造的方法，从而进一步加强了对语法错误的诊断和恢复功能。

LR 语法分析过程首先要建立产生式的项目集和分析表(Action-Goto 表)，分析程序每次从输入缓冲区读入一个符号，并使用栈来存储形如 s0X1s1X2…XmSm 的串，其中 Sm 在栈顶，Xi 是文法符号，Si 称为状态符号，每个状态符号概括了栈中位于它下面的信息。

LR 语法分析器在访问动作表时若遇到出错表项，就检测出一个错误，分析程序就会停止，然后对出现的语法错误进行诊断和恢复。诊断是将可能性比较大的错误原因呈现给开发人员，恢复的功能是要使语法分析程序能够继续执行下去的前提。

语法错误的恢复对语法分析产生重要影响，错误恢复要能够根据上下文准确地判断错误的原因、位置以及使用的错误恢复策略和具体方法。错误恢复不但要能使语法分析正常进行下去，而且要避免在后续的语法分析过程中出现更多的语法错误。通过分析发现，在进行语法分析时，对 LR 语法分析表中的 Goto 表进行改造可以较好地提高语法分析的效率。

1 LR 语法分析中的错误恢复策略

LR(k)分析法是一种有效的自底向上的语法分析技术。它适用于一大类上下文无关文法的语法分析。LR 语法分析器是由输入、输出、栈、驱动程序以及包含动作(Action)和转移(Goto)两部分的语法分析表构成的。不同的 LR 语法分析器的区别主要在于语法分析表的不同。

1.1 通常的错误恢复的策略

通常语法分析器可以采用的语法错误恢复策略主要有以下 4 种。

(1)紧急方式恢复策略

紧急方式恢复是最容易实现的方法，适用于多数语法分析方法。当发现错误时，语法分析器开始抛弃输入记号，每次抛弃一个记号，直到发现某个指定的同步记号为止。同步记号通常是定界符，如分号 ；或 大括号 } 。这种方法常常跳过大量的输入记号，而不检查其中是否有其它错误。这种方法比较简单，不会陷入死循环，但只合适于一个语句中出现的错误数较少的情况。

(2)短语级恢复策略

当发现错误时，语法分析器对剩余的输入字符串进行局部纠正，即用一个能使语法分析器继续工作的字符串来替代剩余输入的前缀。编译器的设计者必须仔细选择替换字符串，以免引起死循环。该方法首先被用于自顶向下的语法分析中，其主要缺点是难以应付实际错误出现在诊断点之前的情况。

(3)出错产生式策略

如果对经常遇到的错误有很清楚的了解，可以扩充语言的文法，增加产生错误结构的产生式。然后用由这些错误产生式扩充的文法构造语法分析器。如果语法分析器使用了出错产生式，就可以产生适当的错误诊断信息，指出在输入字符串中识别出的错误结构。

(4)全局纠正策略

一个理想的编译器是在处理不正确的输入字符串时做尽

基金项目：中国矿业大学校基金资助项目(OD4527)
作者简介：肖 洋(1979－)，男，硕士生，主研方向：程序设计语言，编译技术；姜淑娟，副教授
收稿日期：2006-03-08 **E-mail：**xiaoyang790722@126.com

可能少的改动。有一些算法可以选择最小的修改序列，以获得全局代价最小的错误纠正。但是实现这些算法的时间和空间开销太大，目前只是进行了一些理论上的探讨。

1.2 LR 语法分析器的错误恢复策略

通常，LR 语法分析器在访问动作表(Action 表)时若遇到错误表项，就检测出一个错误，但它在访问转移表时决不会检测出错误。与算符优先文法分析器不同的是，LR 语法分析器只要发现已扫描的输入出现一个不正确的后继就会立即报告错误。规范 LR 语法分析器在报告错误之前不会进行任何无效规约。在 LR 语法分析器进行错误恢复的具体策略如下。

(1)紧急方式的错误恢复

从栈顶开始退栈，直至发现在特定的非终结符 A 上具有转移的状态 S 为止；然后丢弃零个或多个输入符号，直至找到符号 a 为止，a 是 A 的合法后随符号；接着，语法分析器把状态 goto[S,A]压进栈，并恢复正常分析。

(2)短语级恢复

通过检查 LR 分析表的每个出错表项，并根据语言的使用情况确定最可能引起该错误的开发人员最容易犯的错误，然后为该表项编一个适当的错误恢复例程。该例程大概会采用一种适合于相应出错表项的方式来修改栈顶符号和(或)第 1 个输入符号。

(3)出错产生式策略

可以把具体的最易出错的产生式加入到 LR 的分析表当中，但同时会加大分析表的空间，需要特别注意出错产生式是否会影响到正常的分析过程，以免带来更多的错误。

2 对 Action 表进行改造

当今进行错误恢复的手段都是通过修改Action表的空余表项来实现的[3]。下面以一个精简的C#语言规范中的using语句和命名空间声明的LR语法分析表为例，说明对Action表项所进行的具体改造方法。

2.1 一个具体的 Action-Goto 表(表 1)

表 1 LR 分析表(ActionGoto 表)

	id	{	}	.	;	u	n	#	S	U	U'	M	N	N'	B
0						S5	S7		1	2	4		3	6	
1								acc							
2						S5	S7				9		8	6	
3							S7							10	
4						R2	R2								
5	S12											11			
6							R3	R3							
7	S13														
8							S7							10	
9						R2	R2								
10							R3	R3							
11				S15	S14										
12				R5	R5										
13			S18										17		16
14						R4	R4								
15	S19														
16							R6	R6							
17			S20												
18							R8	R8							
19					R5										
20							R7	R7							

终结符：id(标识符) { } . ; u<using> n<namespace> #(结束符)

非终结符：S(编译单元) U(多个 Using 指令) U'(单个 Using 指令) M(命名空间名)N(多个命名空间成员声明) N'(单个命名空间成员声明) B(命名空间体)

产生式：

(0)S' → S #　(1)S → U N | N　(2)U → U'| U U'
(3)N → N'　| N N'　(4)U' → using M ;　(5)M → id | M . id
(6)N' → namespace id B　(7)B→ { N }　(8)B→ { }

2.2 改造原理与方法

当语法分析状态为 0 时，待输入符号为 id 或{或}或.或；时，会出现语法错误。语法分析状态为 0 时，即语法分析刚刚开始，期望输入符号应该为 u(using)或 n(namespace)，如果出现了 id、{、}等符号，根据恢复策略，可以采取忽略当前输入符号的方法，在对应的空白表项中填入 E0。下面以表 1 中的状态项 0 和状态项 11 为例进行改造，其结果如表 2 所示。

表 2 改造后的状态项 0 和 11

	id	{	}	.	;	u	n	#	S	U	U'	M	N	N'	B
0	E0	E0	E0	E0	E0	S5	S7	E1	1	2	4		3	6	
11	E2	E2	E2	S15	S14	E3	E3	E4							

原有的空白表项已经被改换成对出错表项的调用，可以把某些出错表项改成归约，这样 LR 分析表中就可以包含对错误的诊断和恢复，出错表项的具体含义如下：

E0：要求输入符号为 u 或者 n，输入不符要求时调用此例程。

从输入中删除此符号。即忽略当前符号，维持当前状态。

出错信息：“ 应输入 using 或 namespace ”。

E1：要求输入符号为 u 或者 n，但输入符号已经结束。

终止分析。

出错信息：“ 应输入 using 或 namespace ”

E2 :要求输入符号为.或者 ;输入不符要求时调用此例程。

从输入中删除此符号。

出错信息：“ 缺少分号 ”。

E3：当输入符号为 u 或者 n 时，调用此例程。

把；压入栈，并盖以状态 14。

出错信息：“ 缺少分号 ”。

3 智能感知中错误恢复策略

智能感知功能一般包括语法错误的显示，类体系结构的呈现，对象的属性和方法的呈现，函数方法参数的提示等功能。智能感知的实现依赖于即时的词法分析，即时的语法分析以及部分语义分析。当今智能感知功能在编译器中起着重要的作用，强大的智能感知必然要求高效的语法分析以及语法错误的诊断和恢复。智能感知对语法分析的性能要求比较苛刻，对语法错误的恢复要求能够做到及时和尽量准确。传统的通过改变 Action 表的方法很难满足智能感知对语法分析以及错误恢复的时间和性能要求。为此，本文提出了一种新的方法，对 Goto 表进行改造来满足智能感智对语法分析的要求。

3.1 智能感知中的语法分析

实现智能感知功能必须采用动态的语法分析方法。动态的语法分析要求在编写代码的同时要对程序代码即时进行语法分析，以便判断当前的分析状态和上下文环境。实现即时的语法分析，必然要求要尽可能地减少语法分析的范围，即要用一个适当的算法来保存以前的语法分析结果。利用已经存在的分析结果，可以大大提高语法分析的速度，也可以提高语法错误的恢复程度。下面给出一个智能感知进行语法分析后简单保存语法分析结果例子。

图 1 是一段程序伪代码，图 2 是保存的语法分析结果。如果对程序源代码进行修改，如把第 3 行的源代码 *using System.Net*；修改为 *using System.Net.Sockets*；进行语法分析

时，待输入符号的序列就变为[U’ U’ using id . id . id ; N]，这样可以节省大量的重复分析时间，完整地保存了语法分析的上下文，如果出现语法错误就可以根据上下文对错误进行有效地诊断和恢复。因为传统的 LR 分析器的输入符号串全部为终结符，那么实现对终结符和非终结符混合的输入串进行语法分析就必须改造 Action-Goto 表，主要是改造 Goto 表。

```
1   using System;
2   using System.Data;
3   using System.Net;
4   namespace abc
5   {
6           class A
7     {
8     }
9   }
```

图 1 程序伪代码

```
1—3 行 非终结符 U（using语句）
  1 行  非终结符U’
  2 行  非终结符 U’
  3 行  非终结符 U’
4—9 行 非终结符 N（命名空间声明）
    4 行   命名空间头
  5—9 行 命名空间体
    6—8 行 类声明
        …… ……
```

图 2 保存的语法分析结果

3.2 对 Goto 表进行改造

智能感知功能要求发生语法错误时要即时对语法错误进行恢复。由于对以前的分析结果进行保存，在语法分析过程中就会直接遇到非终结符，因此可以对 Goto 表进行改造，以便实现更快捷的语法分析，同时也会大大提高错误恢复的成功率和减少进行错误恢复的代价。下面举一个例子来阐述如何对 Goto 表进行改造。

假设把图 1 中第 3 行的源代码 *using System . Net ；*修改为 *using System* ，即程序开发人员选中 *. Net* ；5 个字符，并且进行删除操作。此时，由于只改变了第 3 行的代码，可以认为，代码的改动“ 破坏 ”了原有的语法分析结果，但“ 破坏 ” 范围是有限的。原有的语法分析结果可以用非终结符的序列表示为：[U N]，虽然代码发生了改动，但完全可以通过对程序开发人员的操作来判断“ 破坏 ” 的范围。利用原有的语法分析结果，可以获得需要重新进行语法分析的符号序列为[U’ U’ using id N]。表 3 示出了根据改造后的 Action-Goto 表对此序列进行分析的过程。

表 3 改造后的 Action-Goto 表对此序列的分析过程

步骤	状态栈	符号栈	输入串	ACTION	GOTO
1	0	#	U’ U’ using id N#		5
2	05	# U’	U’ using id N#		R2
3	02	# U	U’ using id N#		9
4	029	# UU’	using id N#	R2	
5	02	# U	using id N#	S5	
6	025	# U using	id N#	S12	
7	025[12]	# U using id	N#		R5
8	025[11]	# U using M	N#		*F2*

当语法分析进行到步骤 8 时，符号栈栈顶元素为非终结符 M，待输入字符为非终结符 N，此时源代码出现错误，语法分析程序需要进行诊断和恢复。当前最有可能的出错原因是缺少分号，那么可以在 Goto 表中将对应的表项进行修改为 F2。F2 的含义为：在符号栈顶添加一个分号，给出错误提示“ 缺少分号 ”。语法分析程序得以继续，表 4 是语法分析的后续过程，表 5 是改造后的 Action-Goto 表。

表 4 改造后的 Action-Goto 表对此序列的分析过程(续)

9	025[11][14]	# U using	**N#**		R4
10	029	M [;]	**N#**		R2
11	02	# U U’	**N#**		8
12	028	# U	#	R1	
13	01	# UN	#	acc	
		# S			

表 5 改造后的 Action-Goto 表

	id	{	}	.	;	u	n	#	S	U	U’	M	N	N’	B
0	E0	E0	E0	E0	E0	S5	S7	E1	1	2	4	F0	3	6	F0
9	R2	R2	R2	R2	R2	R2	R2	R2	R2	R2	R2	R2	R2	R2	R2
11	E2	E2	E2	S15	S14	E3	E3	E4	F2	F2	F2	F1	F2	F2	F1

表 5 中的 F0、F1 和 F2 的具体含义如下：

F0：直接忽略此非终结符，避免了错误的大量出现。

出错信息：“ 应书写 using 语句或命名空间 ”。

F1：直接忽略此非终结符。

出错信息：“ 缺少分号 ”。

F2：把；压入栈，并盖以状态 14，进行规约。

出错信息：“ 缺少分号 ”。

原有的 Goto 表中空白表项已经被改换成对出错表项的调用，加上原有的对 Action 表的改造，这样 LR 分析表中的空间都具有了语法分析或对错误的诊断和恢复的功能。

改造 Goto 表的优点：可以尽可能利用 Goto 表中的空余表项，提高了错误诊断和错误恢复的效率和准确率，为实现智能感知功能提供了有利的前提条件。缺点是改造 Goto 表必须依赖于以前的语法分析结果，这种方法不适用于对单遍的语法分析方法。

4 总结

本文提出了一种对 LR 分析表中的 Goto 表项进行改造，以实现更好地对语法错误进行诊断和恢复的方法。结合文献[3]中的对 Action 表的改造方法，对 LR 分析表的空闲表项加以利用，在不增加空间需求的情况，对 Goto 表项进行改造。并把对 Action 表项的改造方法和对 Goto 表项的改造方法进行结合在一起，使对语法错误的恢复操作更准确、更有效。由于对 Goto 表的改造需要依赖以前语法分析结果，对于自上而下完整的单遍语法分析不适用。但是，现代编译器中词法分析、语法分析都是即时性的，这样对 Goto 表的改造就可以发挥重要的作用。对 Goto 表的改造的方法只是实现智能感知功能的一部分，对 Goto 表中表项所定义的动作还需要进行深入的研究，才能实现对语法错误高效、准确的诊断和恢复，这也是下一步研究和实现的目标。

参考文献

1 吕映芝, 张素芹, 蒋维杜. 编译原理[M]. 北京: 清华大学出版社, 1998.

2 Louden K C. 编译原理及实践[M]. 冯博琴, 冯 岚, 译. 北京: 机械工业出版社, 2000.

3 Aho A V, Sethi R, Ullman J D. 编译原理[M]. 李建中, 姜守旭, 译. 北京: 机械工业出版社, 2003.

4 Appel A W. 现代编译器的 Java 实现 M]. 陈 明, 译. 北京: 电子工业出版社, 2004.

5 张幸儿. 计算机编译原理：编译程序构造实践[M]. 北京: 科学出版社, 2005.