PL/0 编译器的错误与错误恢复

PB09210183 何春晖

2012.03.24

1 错误类型与举例

根据 PL/0 文档记载，PL/0 编译器一共有近 27 个错误号，此处不再详细列出各个号码的具体含义。

错误号虽然很多，但是通过分析，大致可以分为三类。下面使用添加了错误说明的修改版 PL/0 编译器说明。

1.1 语法错误

此类错误的模式一般是遗漏某个记号或某个记号错误，错误号为：

```
1 2 3 4 5 6 7 8 9 10 13 14 16 17 18 19 20 22 23 24
```

程序中出现此种错误的一般是程序员疏忽大意造成的，如下例：

```
const a=3;
var b;
if a=3 than b:=4.
```

报错为：

```
 ****           ^23
errmsg(23): 因子后不可为此记号
 ****                 ^16
errmsg(16): 应为 then
```

1.2 语义错误

此类错都与类型有关，如调用变量、过程名参与计算等，错误号为：

```
11 12 15 21
```

例子如：

```
var a;
procedure b;
begin a:=1 end;
begin b=a+1 end.
```

报错为：

```
 ****       ^12
errmsg(12): 不可向常量或过程赋值
 ****        ^13
errmsg(13): 应为赋值运算符:=
 ****        ^24
errmsg(24): 表达式不能以此记号开始
```

1.3 由于实现的限制产生的错误

此类错误其实不是“错误”，只是由于数量超出了预先规定的或机器本身的限度而错，错误号为：

```
30 31 32
```

此类错误，如：

```
procedure a;
procedure b;
procedure c;
procedure d;
procedure e;
....
```

报错为

```
 ****         ^32
errmsg(32): 程序嵌套层次太多
```

2 错误恢复

2.1 error 和 test 函数

error 和 test 函数是负责处理错误的两个关键函数。

error 用于报错。经过修改的报错函数首先用 cc 变量中存储的列信息，在错误成分下部打印“^”符号，然后用错误号查错误表 errmsg，打印错误号的含义。最后将统计变量 err 自增。

test 用于错误的检测和恢复，它有三个参数 s1、s2 和 n。这三个参数含义是：

- s1: 准备接受的终结符集合

- s2: 同步到的终结符集合
- n: 发送的错误号 test 语义为：检查当前记号是否在 s1 中，若在，不做任何操作；若不在，报 n 号错，一直扫描记号直到记号在 s1 或 s2 中。

2.2 语法错误的检测和恢复

2.2.1 直接检测

当某个产生式直接产生一个关键字终结符（如逗号、“if”、“then”等）时，只需直接判断 sym 是否是对应记号名。若不是，报错。发生错误后，直接将向后继续处理。如 5 号错误的处理如下模式：

```
if(sym==semicolon){
  getsym();
}else{
  error(5);
}
```

2.2.2 检测 Follow 集合

Follow 集合检测涉及变量 fsys。它是每个非终结符对应处理函数的参数名。猜测其含义就是 **follow symbol set**。一个比较明显的例子是在 statement 中最后一句 test(fsys,0,19)。它的意义是：**当前已经分析完一个 statement，查看当前记号是否是 Follow 集合中的元素，若不是，说明语法有错，报 19 号错并同步到出现 Follow 集合中的记号为止**。

2.2.3 检测 First 集合

First 集合的检测涉及到 declbegsys、statbegsys 和 facbegsys 的理解。这三个变量经常出现在 test 的参数中，根据变量的值和使用情况猜测其含义分别为：**declaration begin symbol set**、**statement begin symbol set 和 factor begin symbol set**。

如 facbegsys=ident|number|lparen，正好是 First(factor) 中的元素。它的唯一一次出现是在 factor 函数开头的 test(facbegsys,fsys,24)。其意义很明显：**若当前记号不在 First(facbegsys) 中，则说明当前非终结符 factor 不可能产生记号流的记号，因此源程序有语法错，报 24 号错并将流同步到 fsys 集合（跳过当前语法结构）中** 。

2.3 其他错误的检测和恢复

对语义错误和由于实现的限制产生的错误，直接在语义动作中判断即可。从源码看，当此类错误发生时，报错，不产生或产生默认的机器代码。**由于此类错误没有语法上的问题，因此不会干扰整个编译流程，不需要做特别的同步处理**。

3 不足及改进方案

上面分析了 PL/0 代码的错误恢复方案，主要值得讨论的是语法错误的恢复。**PL/0 的恢复方案优点是每一种恢复策略都有根有据，因为 LL(1) 文法给出后，First 集合和 Follow 集合都是可以机械地求出的**。但是实践上有几个问题：

- **有时报错不清晰**。如下面的“遗漏分号”错：

```
var a, b;
begin
    a := 3  /* miss ; */
    b := 4
end.
```

报错为：

```
    3 b:=4
 **** ^23
errmsg(23): 因子后不可为此记号
```

显然报“缺少分号”更合理。改进方案是：**报错时把 test 函数中的 s1 集合中的符号报出**。

- **代码不清晰**。每个非终结符过程都有一个 fsys 参数，随着分析流程的深入，上一层 fsys 参数传给下一层，到最后 fsys 中究竟包含那些终结符，已经搞不清楚。这给改进和调试错误恢复机制带来了困难。改进方案是：**取消 Follow 集合的检测，即把检测错的任务交给调用者，而非被调用者**。

这两个改进方案将在 C0 编译器中实现。