MATHEW

Mathew(马修) lalr(1) 分析器：

命名方式

Mathew 对名字分割不明显的采用下滑线命名，对函数采用下滑线命名和骆驼命名还有少量的帕斯卡命名。对外接口函数全部以大写字母开头。

文件结构：

1. 语法文件(.y)

2. 生成模板文件(template\_语言类型.tpl 例如template\_c.tpl template\_java.tpl)

命令行情况：

mathew gram.y -?

-b 只打印核心项目

-c 不压缩动作表

-r 打印gram.out语法文件时不打印动作

-m 输出外部gram.h文件

-q 不生成gram.out文件

-s 输出统值信息

-v 输出版本信息

filesuffix=<string> "输出分析器文件名后缀,默认为”.c”"

macroinc=<integer> 宏数组每次预留空间大小

echo(<string>) 原样输出指定参数

<macro> 定义%ifdef宏变量

例如：

mathew gram.y –v –r Win32 –m filesuffix=.cpp echo(“asda”) echo(123)

所有函数调用都传回一个字符串由调用函数进行参数拆解。

macroinc 与宏定义的位置可以动态改变宏预留空间的大小。

语法文件

终结符：

必须以大写字母开头后接“\_”和大写字母。

非终结符：

必须以小写字母开头后接“\_”和小写字母。

产生式：

expr -> expr PLUS expr.

expr -> expr TIMES expr.

expr -> expr LPAREN expr RPAREN.

expr -> VALUE.

加入动作的产生式:

expr($A) -> expr($B) PLUS {printf(“just enter the addition symbol”);}expr($C).{$A=$B+$C;}

或:

expr($A) -> expr($B) PLUS ($plus)

{

printf(“just enter the addition symbol”);

}

expr($C).{$<int>A=$B+$C;}

别名技术：

必须以$开头后面接以字母开头的标识符。

特殊声明符

%code %destructor(NT&& T) %extra\_arg

%include %left %parse\_name

%nonassoc %nonterminal\_destructor %nonterminal\_type

%parse\_accept %parse\_failure %right

/\*%stack\_overflow %stack\_size\*/ %start

%syntax\_error %token\_destructor %type(NT&& T)

%token\_prefix %token\_type %fallback

错误处理

笔者将错误消息存放在了一起，这样易于语言平台的相互装换。

关于编码习惯

编码呈现树状结构底部为主干函数，越上下功能就实现得越细，也就是树的枝干。

Table.c

哈希表的处理放在table.c里，使用了统一接口初始化哈希表，统一接口向哈希表进行查找和插入，这里目前有四张哈希表，分别为字符串哈希表，符号哈希表，状态哈希表，项目哈希表

语法文件分析器状态转换表

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | 状态 | % | 小写字母 | 大写字母 | 别名 | number | { | [ | ] | -> | ( | ) | . | < | code等 |
| 0 | INITIALIZE |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1 | WAITING\_FOR\_DECL\_OR\_RULE | 2 | 3 |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 | WAITING\_FOR\_DECL\_KEYWORD |  |  |  |  |  |  |  |  |  |  |  |  |  | 13 |
| 3 | WAITING\_FOR\_ARROW |  |  |  |  |  |  |  |  | 4 | 5 |  |  |  |  |
| 4 | IN\_RHS |  | 4 | 4 |  |  | 4 |  |  |  | 8 |  | 12 |  |  |
| 5 | LHS\_ALIAS\_1 |  |  |  | 7 |  |  |  |  |  |  |  |  |  |  |
| 6 | LHS\_ALIAS\_2 |  |  |  |  |  |  |  |  |  |  | 7 |  |  |  |
| 7 | LHS\_ALIAS\_3 |  |  |  |  |  |  |  |  | 4 |  |  |  |  |  |
| 8 | RHS\_ALIAS\_1 |  |  |  | 9 |  |  |  |  |  |  |  |  |  |  |
| 9 | RHS\_ALIAS\_2 |  |  |  |  |  |  |  |  |  |  | 4 |  |  |  |
| 10 | PRECEDENCE\_MAK\_1 |  |  | 11 |  |  |  |  |  |  |  |  |  |  |  |
| 11 | PRECEDENCE\_MAK\_2 |  |  |  |  |  |  |  | 12 |  |  |  |  |  |  |
| 12 | WAITING\_FOR\_RULE\_CODE |  |  |  |  |  | 1 | 11 |  |  |  |  |  |  |  |
| 13 | DECL\_... |  | 1 | 1 |  | 1 | 1 |  |  |  |  |  | 1 |  |  |
| 14 | RESYNC\_AFTER\_ERROR |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

我们将%name后面分为六种形式进入五种状态:

%name {} /\* 代码块类型 \*/

%name <type> token [token...] . /\* 符号类型 \*/

%name token [token...] . /\* 优先级 \*/

%name {} token [token...] . /\* 析构类型 \*/

%name token /\* 值类型 \*/

%name number /\* 数值类型 \*/

插入动作参考yacc如下:

A->a {action1} b.{action2}

将被处理为

A->a nt\_000001 b.{action2}

nt\_0000001->.{action1}

关于多符号出错处理：

只要一遇到某个符号出错我们就进行错误从同步到下一声明或规则的起始，出错符号前面的都将被采纳。这里并不采用越过某些错误符号的方式来解决，为了与规则错误对应，错了就要放弃此规则重同步到新的规则。

%ifdef %ifndef %endif 后面一定要接一个空格，只有缺少%enif 如果只写了%endif没写%ifndef或%ifdef则mathew会自动将%endif删除并不提示出错

关于漏写>则将与下一个<type> 结合变为 <type类型出错检查将会晚点才能发现

插入动作利用了环形队列实现。First follow集用位来存放。

求first集利用递归方式以避免系统的重入性。

没有rule的noterminal;

如a->b. b没有再次出现,为b设置为b->. void ProcessNoRuleSymbol(struct global \*gp,struct symbol \*sp);实现(只为那些能以开始符号推倒出来的没有产生式的非终结符建立默认产生式symbol->.)。

如果开始符号在右边出现那么将建立一个新的开始符号start\_001开始寻找如果存在就继续往下找指导建立一个新的开始符号为止。动作为start\_001->old\_start. FindStartSymbol(struct global \*gp)并将产生式添加到产生式头部。

Rule->index 和dot 一样才作为相同的基本项目内存链技术：释放某一阶段产生的所有内存，初始化时为memory\_head申请一个指针的空间，在TempMem\_alloc(void \*memory\_head/\*多加一个参数代表哪个链的\*/)时候多在内存头部增加一个全局指针extern void \*memory\_head;的大小在下一次申请内存时移动memory\_head指向当前申请的内存块，当前内存块空出内存头指针指向空，及时记录memory\_head,到执行完部分不用代码后可释放着部分产生的空间。只对一些临时性空间进行如config,，rule,在某一阶段使用完后就要释放了。

Symbol config state 等插入时候就自动排好序。Config在插入基项目时作时候向bp按序插入，向next插入非基本项目时按序插入next中。

@也可以把Mathew当作生成firstset或者只求闭包的工具(给实验测试所用)。由option选项提供选项只提供生成到某一步。

产生式的优先级以precsymbol表示（产生式第一个终结符的优先级）因为排序按同一个先行符先移进后规约的形式，要用先行符与规约产生式的第一个符号进行比较。

Token\_prefix在模板文件中也可更改如%token <#define TK\_%s%-30s %2d\n> 加入TK前缀

模板文件结构：

Map映射部分

%%

头文件

%%

Map部分:

Map的key部分是以%开头空格符结尾，map的value部分是以<>括起来的字符串，!后面的字符原样加入到value里！不加。

%parse\_name 替换模板文件里指定的%parse\_name

%replace 特殊开关为了特殊语言使用，.y%parse\_name适合规范语言。Repalce开关可以进行内部替换如：

%repalce <@Parse,@YY\_Parse,int,unsigned int>

成对出现进行替换。

%replace 的内容拥有前面的替换规则优于后面出现的如:

<int,char,int,long>所有” in”将被替换为char。

<unsigned, , int,unsigned int>

来避免出现unsigned int 替换为unsigned unsigned int这样会缩进不美观建议<unsigned int,unsigned int,int,unsigned int>

子一级注释：！！被Map消掉了一级!所以要!!(char)实现对’,’的屏蔽

%replace <Parse,KK\_Parse,unsigned int,unsigned int!!,int,unsigned int>

%fallback A B C.

B C 的回退目标都为A。fallback 每出现一次后面的都以第一个为目标进行填充符号的fallback

关于析构处理:

%type <struct s\*> A B C D.

%destructor {free($$);} A B C.

D没有在析构里出现那么D出栈时将不会调用析构这原理为先以sp(D)->destructor判断是否生成析构函数映射关系。生成析构函数将会把sp(D)->destructor变0。所以要先调用tpl\_destructor\_init。

规约代码中!$A感叹号与规则部分一样有转义作用表示原样输出$, $A表示栈的minor值

未被登记的别名就不会写入到输出文件里使用转义字符!就会更改该$的性质使其可以原样写入输出文件中$-表示为$0-,$-0表示$0。