曹泰丰对murphi修改的相关说明

# 一、目的

本次对murphi的修改实现了两个功能。第一，改变murphi软件的输出，将输出从.cpp文件改为.m文件。第二，将输出文件中的if语句和rule卫式中的‘|’、‘->’进行拆分（for循环中的情况不在考虑范围）。

# 二、软件的使用环境

修改后的murphi的使用环境与原来相同。请参考“Murphi工具调研报告和使用规范.docx”。

# 三、软件修改说明

## 3.1头文件

1.为了添加新函数，将rule.hpp加入到了expr.hpp中，原rule.hpp改名为ctfrule.hpp。并对makefile文件进行了相关修改。ctfrule.hpp文件并不会被用到，保留的目的只是作为rule.hpp的备份。

2. 为了添加新函数，对stmt.hpp加入了新的include说明。

3. 为了添加新函数和新成员，在新函数对应的.hpp文件中加入了相关声明

新添加函数及成员如下：

1. simplerule类中的void ifsearch(simplerule \*sr);
2. simplerule类中的virtual const char \*generate\_code(simplerule \*sr);
3. simplerule类中的void addcondition(expr \*test,int if\_flag);
4. simplerule类中的void copy(simplerule \*\*srn);
5. simplerule类中的void conditionsign();
6. expr类及其派生类中的virtual void copy(expr \*\*exprn);
7. expr类及其派生类中的virtual void sign(simplerule \*sr);
8. boolexpr中的新成员 int flagor
9. stmt结构体以及继承他的结构体中的virtual void ifsearch2(simplerule \*sr,stmt \*ifn);
10. stmt结构体以及继承他的结构体中virtual void ifsearch1(simplerule \*sr,stmt \*ifn,stmt \*ifn2)
11. ifstmt类中的新成员int isdecleared 和int isdecleared2，在最后修改版本中isdecleared2并没有用到，因此不对其进行说明。
12. expr类及其派生类中的virtual void takeno(expr \*\*exprn,int flag);
13. 结构体Stack
14. 函数void blankadd()
15. 函数void blankred()
16. 函数void writeblank()
17. 函数void Init(Stack &s)
18. 函数void Push(Stack &s,stmt \*e)
19. 函数int IsEmpty(Stack s)
20. 函数void Pop(Stack &s,stmt \*\*e)
21. 函数void Copy(Stack &s1,Stack &s2)
22. 函数const char \*ste::search\_decls()

4.各个新成员以及新函数的设计目的

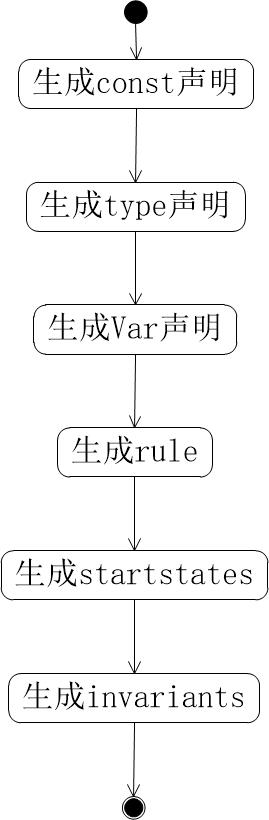
|  |  |
| --- | --- |
| 名称 | 设计目的 |
| simplerule类中的void ifsearch(simplerule \*sr); | 用于调用ifsearch2函数，对rule中if结构进行拆分。 |
| simplerule类中的virtual const char \*generate\_code(simplerule \*sr)； | 用于生产rule代码 |
| simplerule类中的void addcondition(expr \*test,int if\_flag); | 用于将if中的条件判断加入到卫式中，if\_flag = 0表示if判断为假的情况。if\_flag = 1表示if判断为真的情况。 |
| simplerule类中的void copy(simplerule \*\*srn)； | 用于复制rule，注意该函数只对卫式产生额外的副本。 |
| simplerule类中的void conditionsign()； | 用于调用sign函数，对卫式中的‘|’、‘->’进行拆分。 |
| expr类及其派生类中的virtual void copy(expr \*\*exprn)； | 用于expr类及其派生类的复制。 |
| expr类及其派生类中的virtual void sign(simplerule \*sr)； | 用于对卫式中的‘|’、‘->’进行拆分。 |
| boolexpr中的新成员 int flagor | flagor = 100表示该boolexpr没有被标记过。  flagor = 0 生成表达式左半部分。  flagor = 1 生成表达式右半部分。 |
| stmt结构体以及继承他的结构体中的virtual void ifsearch2(simplerule \*sr,stmt \*ifn); | 用于对if结构进行拆分。sr指if所在的rule指针。ifn指该语句的下一条语句的指针。 |
| stmt结构体以及继承他的结构体中virtual void ifsearch1(simplerule \*sr,stmt \*ifn,stmt \*ifn2); | 用于对if结构进行拆分。sr指if所在的rule指针。ifn指该语句的下一条语句的指针。ifn2指该语句所在结构（一般是if结构）的下一条语句的指针。 |
| ifstmt类中的新成员int isdecleared | isdecleared = 1：生成if中的内容  isdecleared = 0：生成else中的内容 |
| expr类及其派生类中的virtual void takeno(expr \*\*exprn,int flag) | 用于去掉!()的()，如！（A=B）替换为A！=B。其中expr \*\*exprn用于传递需要修改卫式的指针的指针。int flag用于表示该expr是否被！所标识。flag = 0表示没有被标识。flag = 1表示被标识。 |
| 结构体Stack | 用于创建存储stmt指针的栈用于拆分if |
| 函数void blankadd() | 用于增加缩进的长度 |
| 函数void blankred() | 用于减少缩进的长度 |
| 函数void writeblank() | 用于打印缩进 |
| 函数void Init(Stack &s) | 用于初始化Stack栈 |
| 函数void Push(Stack &s,stmt \*e) | 用于Stack栈的压栈 |
| 函数int IsEmpty(Stack s) | 用于判断Stack栈是否为空 |
| 函数void Pop(Stack &s,stmt \*\*e) | 用于Stack栈的弹栈 |
| 函数void Copy(Stack &s1,Stack &s2) | 用于Stack栈的复制 |
| const char \*ste::search\_decls() | 用于打印const、type和var |

5.各个新函数的实现位置

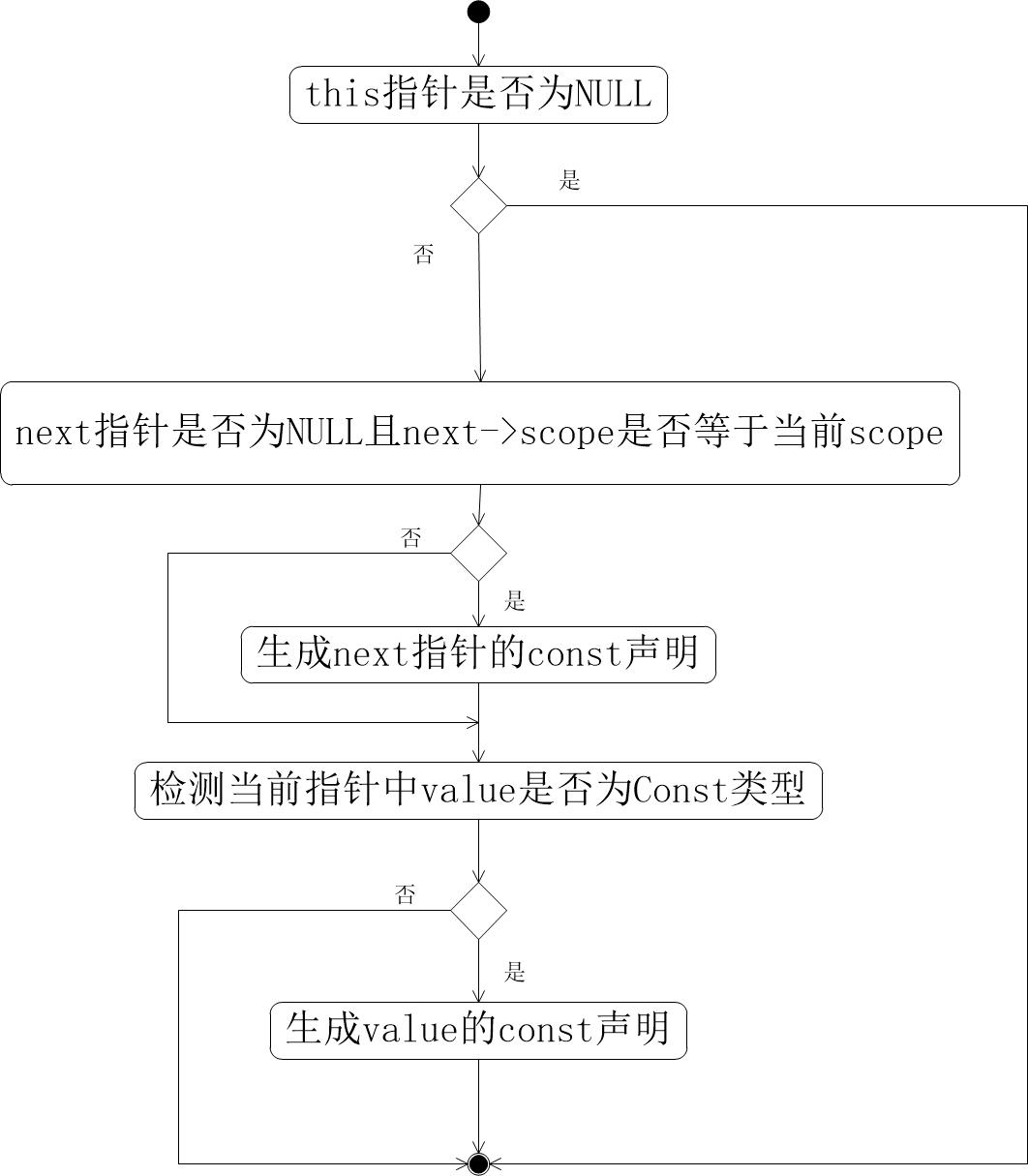
|  |  |
| --- | --- |
| 名称 | 位置 |
| simplerule类中的void ifsearch(simplerule \*sr); | cpp\_code.cpp:3918-3926 |
| simplerule类中的virtual const char \*generate\_code(simplerule \*sr)； | cpp\_code.cpp:3930-3992 |
| simplerule类中的void addcondition(expr \*test,int if\_flag); | cpp\_code.cpp:3994-4012 |
| simplerule类中的void copy(simplerule \*\*srn)； | cpp\_code.cpp:4419-4425 |
| simplerule类中的void conditionsign()； | cpp\_code.cpp:4545-4552 |
| expr类及其派生类中的virtual void copy(expr \*\*exprn)； | cpp\_code.cpp:4427-4647 |
| expr类及其派生类中的virtual void sign(simplerule \*sr)； | cpp\_code.cpp:4554-4676 |
| stmt结构体以及继承他的结构体中的virtual void ifsearch2(simplerule \*sr,stmt \*ifn); | cpp\_code.cpp:4217-4416 |
| stmt结构体以及继承他的结构体中virtual void ifsearch1(simplerule \*sr,stmt \*ifn,stmt \*ifn2); | cpp\_code.cpp:4017-4212 |
| expr类及其派生类中的virtual void takeno(expr \*\*exprn,int flag) | cpp\_code.cpp:4650-4836 |
| 结构体Stack | stmt.hpp:35-39 |
| 函数void blankadd() | cpp\_code.cpp:111-114 |
| 函数void blankred() | cpp\_code.cpp:115-118 |
| 函数void writeblank() | cpp\_code.cpp:119-126 |
| 函数void Init(Stack &s) | cpp\_code.cpp:128-136 |
| 函数void Push(Stack &s,stmt \*e) | cpp\_code.cpp:137-146 |
| 函数int IsEmpty(Stack s) | cpp\_code.cpp:147-157 |
| 函数void Pop(Stack &s,stmt \*\*e) | cpp\_code.cpp:158-162 |
| 函数void Copy(Stack &s1,Stack &s2) | cpp\_code.cpp:163-170 |
| 函数const char \*ste::search\_decls() | cpp\_code.cpp:4840-4860 |

## 3.2主要功能

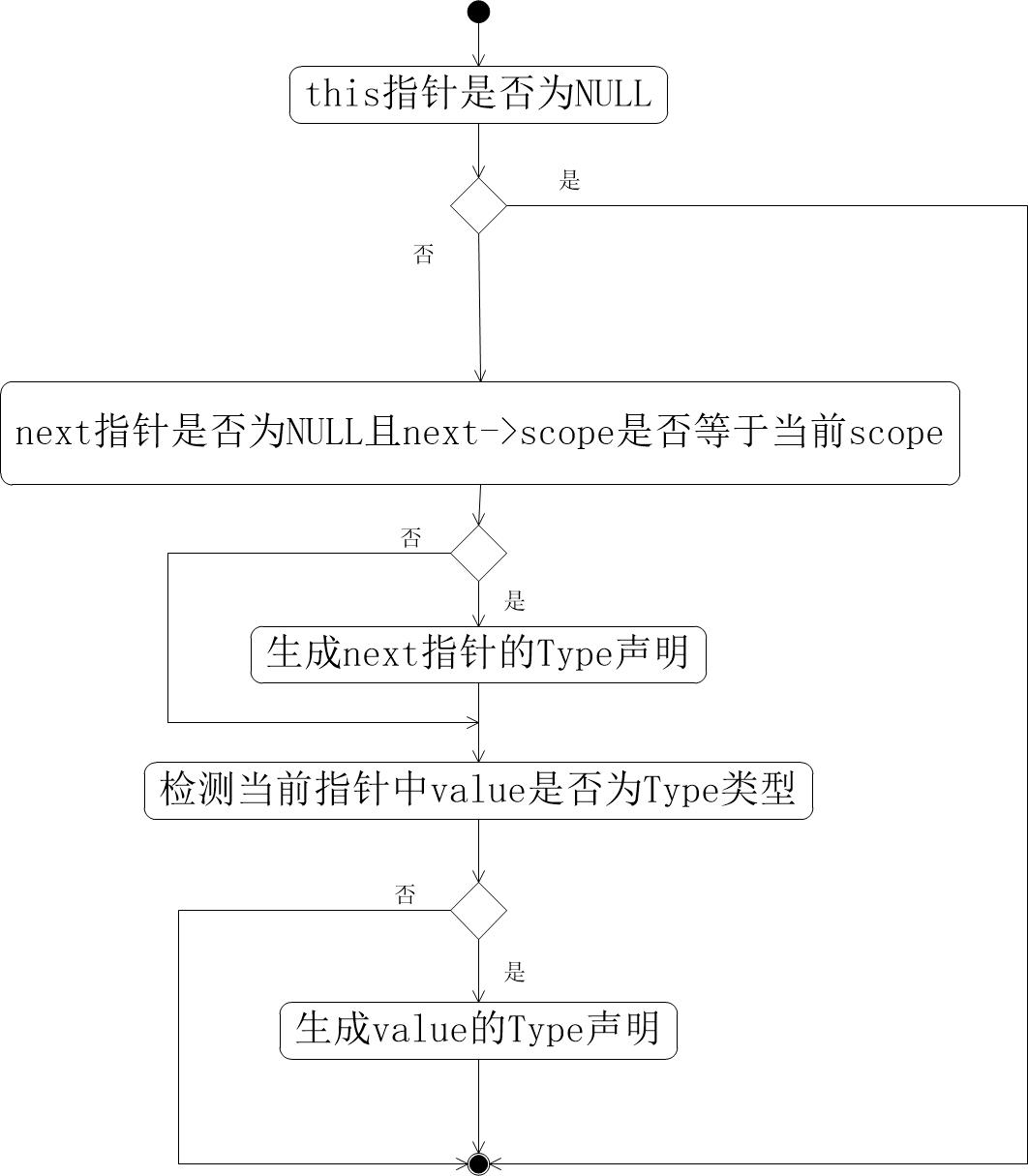
### 3.2.1代码生成顺序cpp\_code.cpp:4867-4882



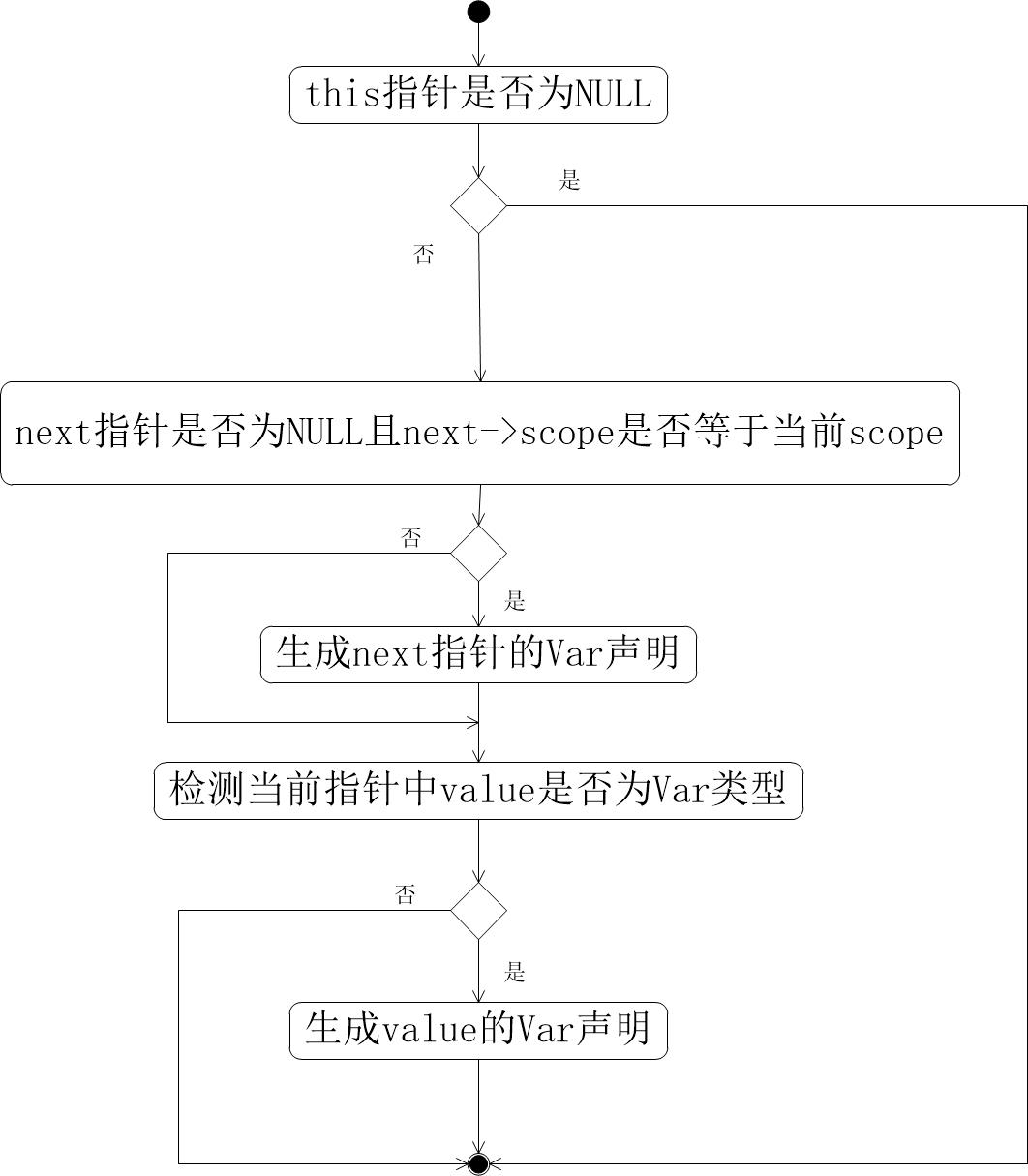
### 3.2.2生成const声明cpp\_code.cpp:3614-3626



### 3.2.3生成type声明cpp\_code.cpp:3628-3640

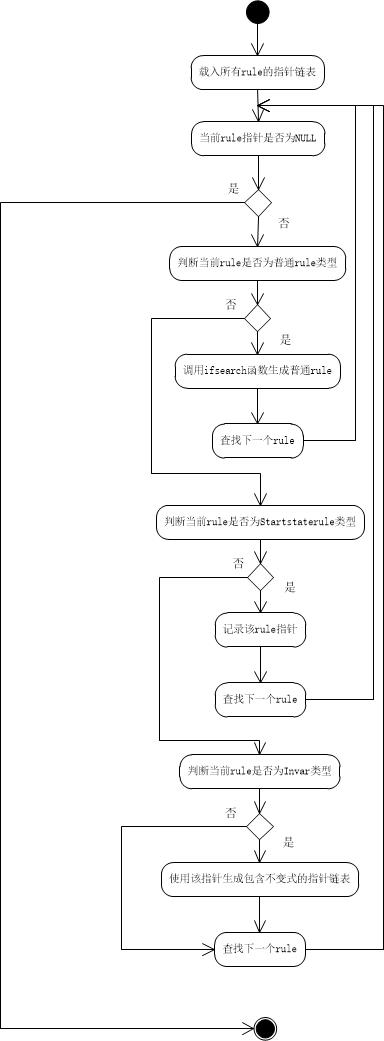


### 3.2.4生成var声明cpp\_code.cpp:3653-3665



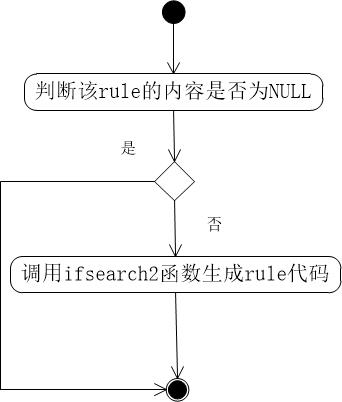
### 3.2.5生成rule代码1 cpp\_code.cpp:3889-3916

该部分描述的是函数generate\_ruleset();



### 3.2.6生成rule代码2 cpp\_code.cpp: 3918-3926

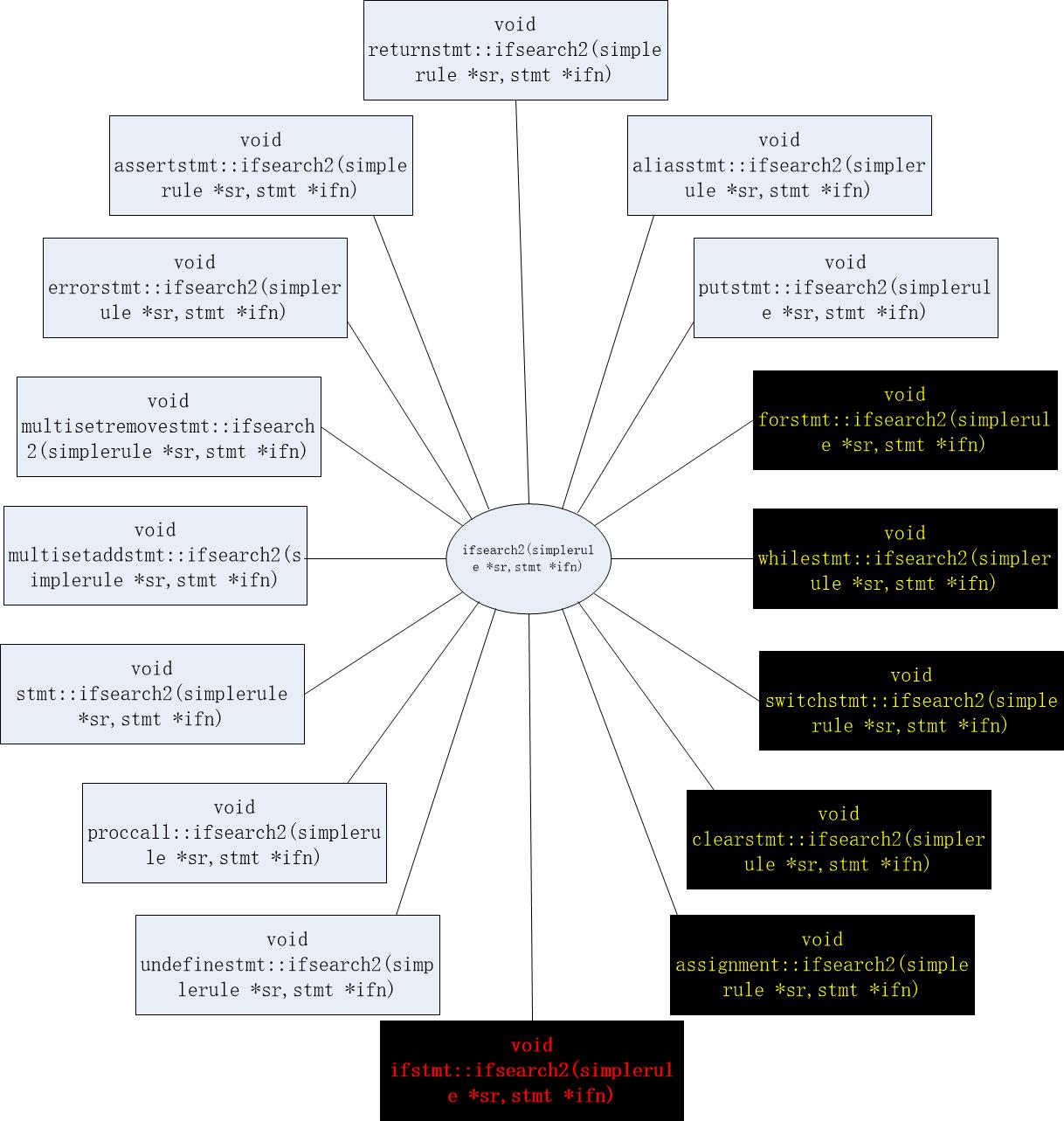
该部分描述的是函数void simplerule::ifsearch(simplerule \*sr)



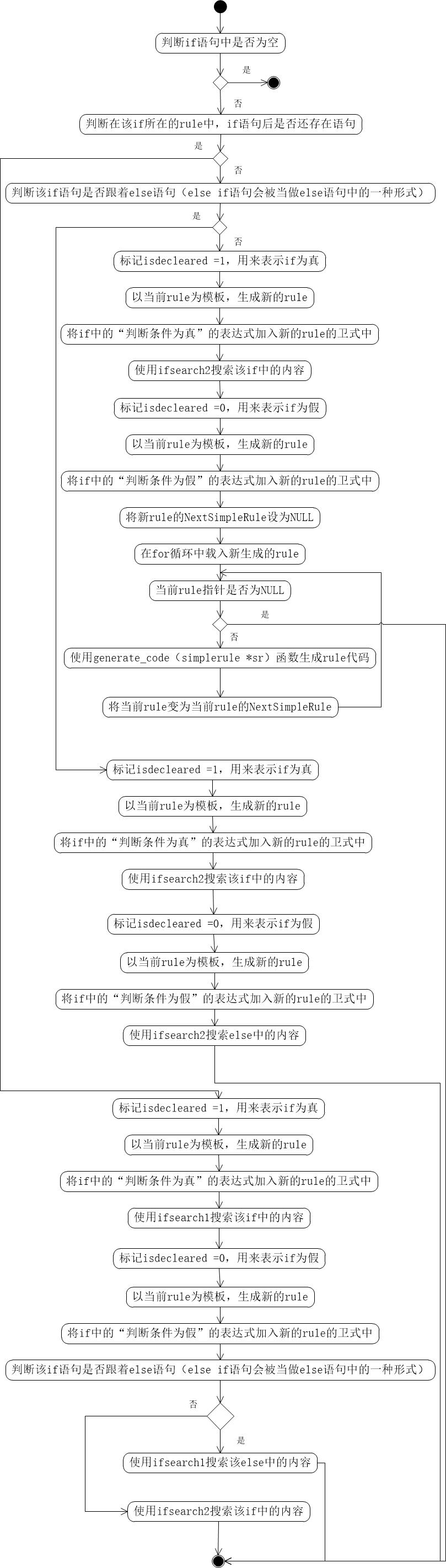
### 3.2.7生成rule代码3 cpp\_code.cpp: 4217-4416

该部分描述的是stmt结构体以及继承他的结构体中的virtual void ifsearch2(simplerule \*sr,stmt \*ifn);

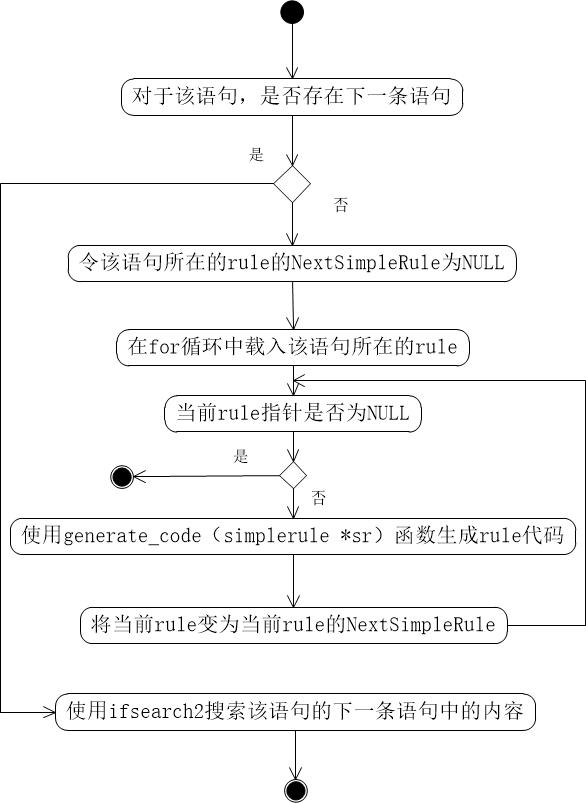
下图中黑色底色表示实现功能的函数，其中红字表示重要功能函数，黄字表示实现过渡功能函数。浅蓝底色表示实际上没有实现的函数。



下图描述实现主要功能的函数void ifstmt::ifsearch2(simplerule \*sr,stmt \*ifn); cpp\_code.cpp:4217-4302



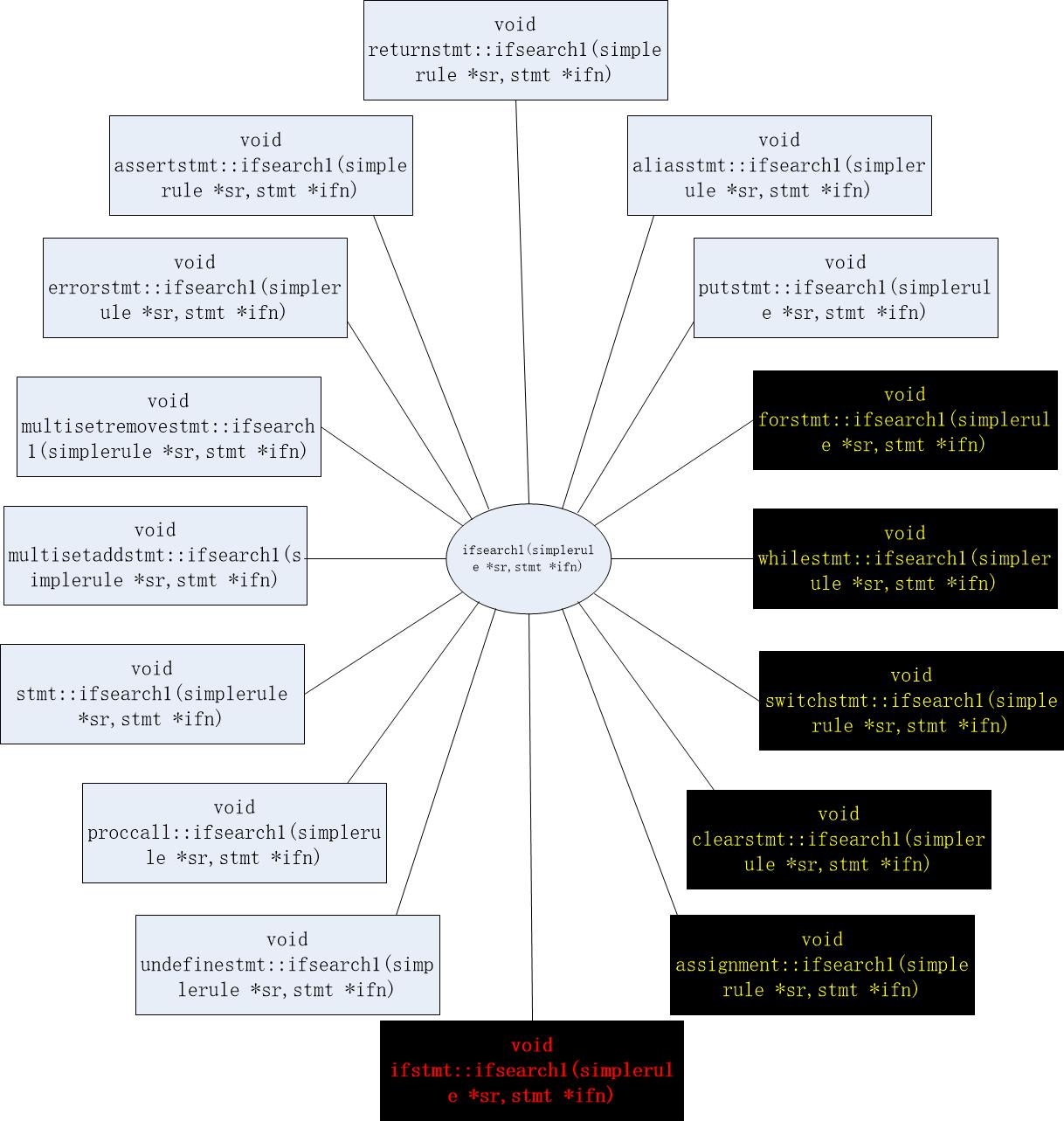
下图描述实现过渡功能的函数



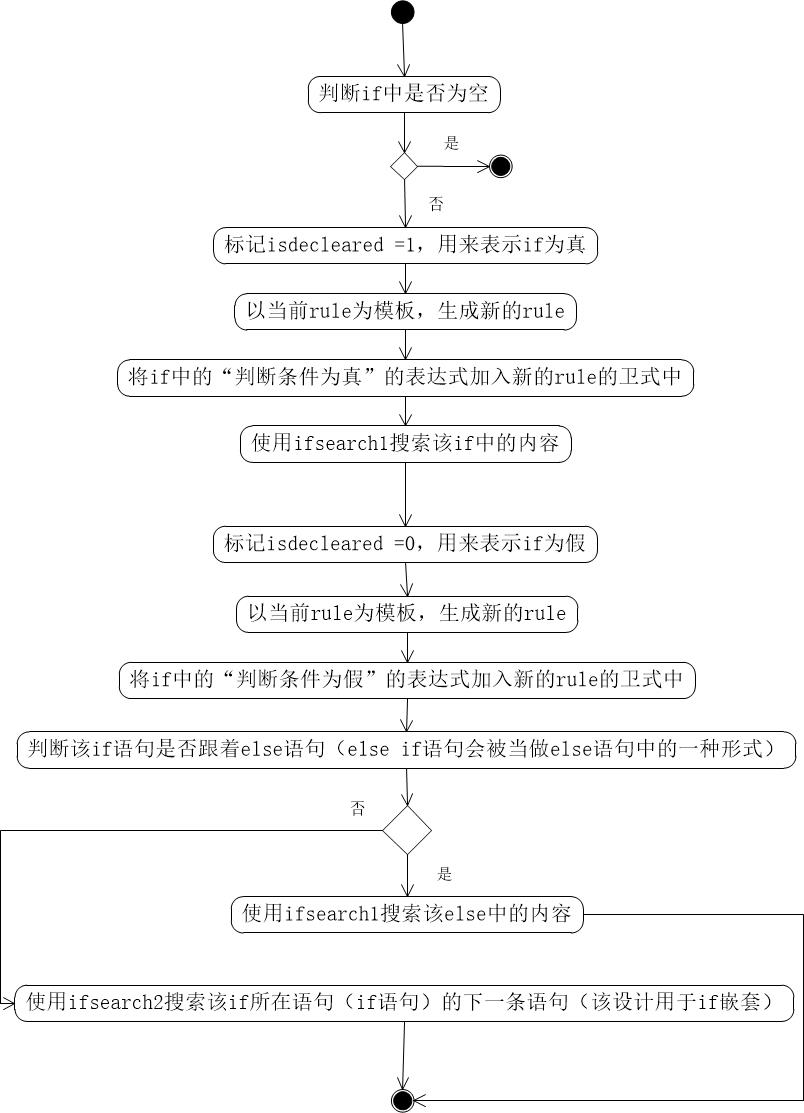
### 3.2.8生成rule代码4 cpp\_code.cpp:4017-4212

该部分描述的是stmt结构体以及继承他的结构体中的virtual void ifsearch1(simplerule \*sr,stmt \*ifn);

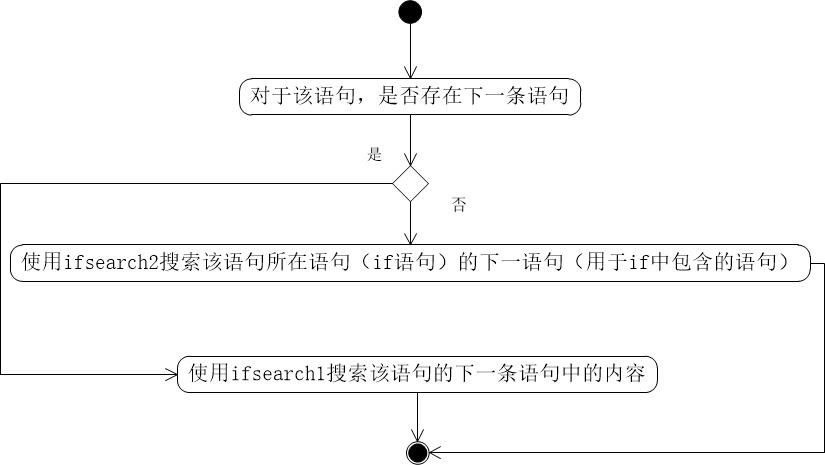
下图中黑色底色表示实现功能的函数，其中红字表示重要功能函数，黄字表示实现过渡功能函数。浅蓝底色表示实际上没有实现的函数。



下图描述实现主要功能的函数void ifstmt::ifsearch1(simplerule \*sr,stmt \*ifn,stmt \*ifn2)cpp\_code.cpp:4017-4068

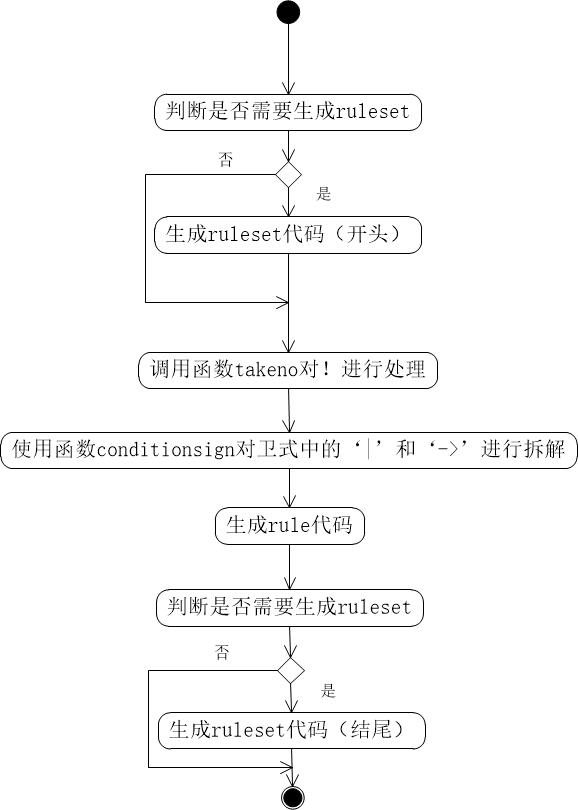


下图描述实现过渡功能的函数



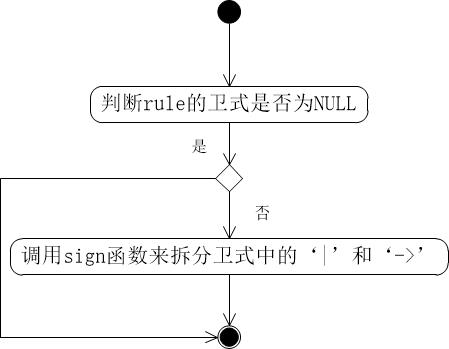
### 3.2.9生成rule代码5 cpp\_code.cpp:3930-3992

该部分主要描述函数const char \*simplerule::generate\_code(simplerule \*sr)



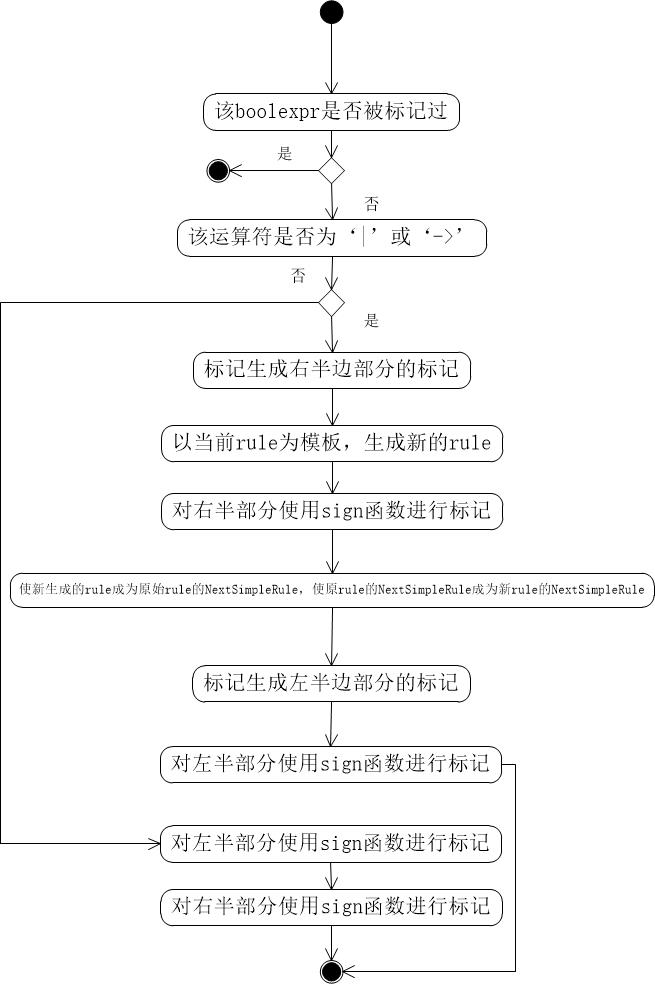
### 3.2.10对卫式中的‘|’和‘->’进行拆解1 cpp\_code.cpp:4545-4552

该部分描述的是函数void simplerule::conditionsign()



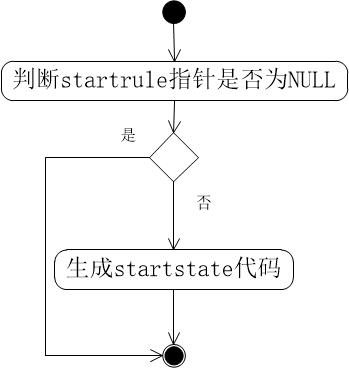
### 3.2.11对卫式中的‘|’和‘->’进行拆解2 cpp\_code.cpp:4554-4648

该部分描述的是expr类及其派生类中的virtual void sign(simplerule \*sr)；其中boolexpr结构体中的sign函数为主要函数，其他函数均为过渡函数，由于过渡函数比较简单，在此不做描述。下图为函数void boolexpr::sign(simplerule \*sr) cpp\_code.cpp:4563-4590



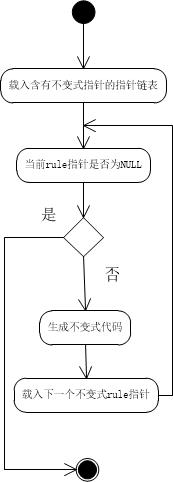
### 3.2.12生成startstates代码 cpp\_code.cpp:3878-3887

该部分描述函数int generate\_startstates()



### 3.2.13生成不变式代码 cpp\_code.cpp:3866-3876

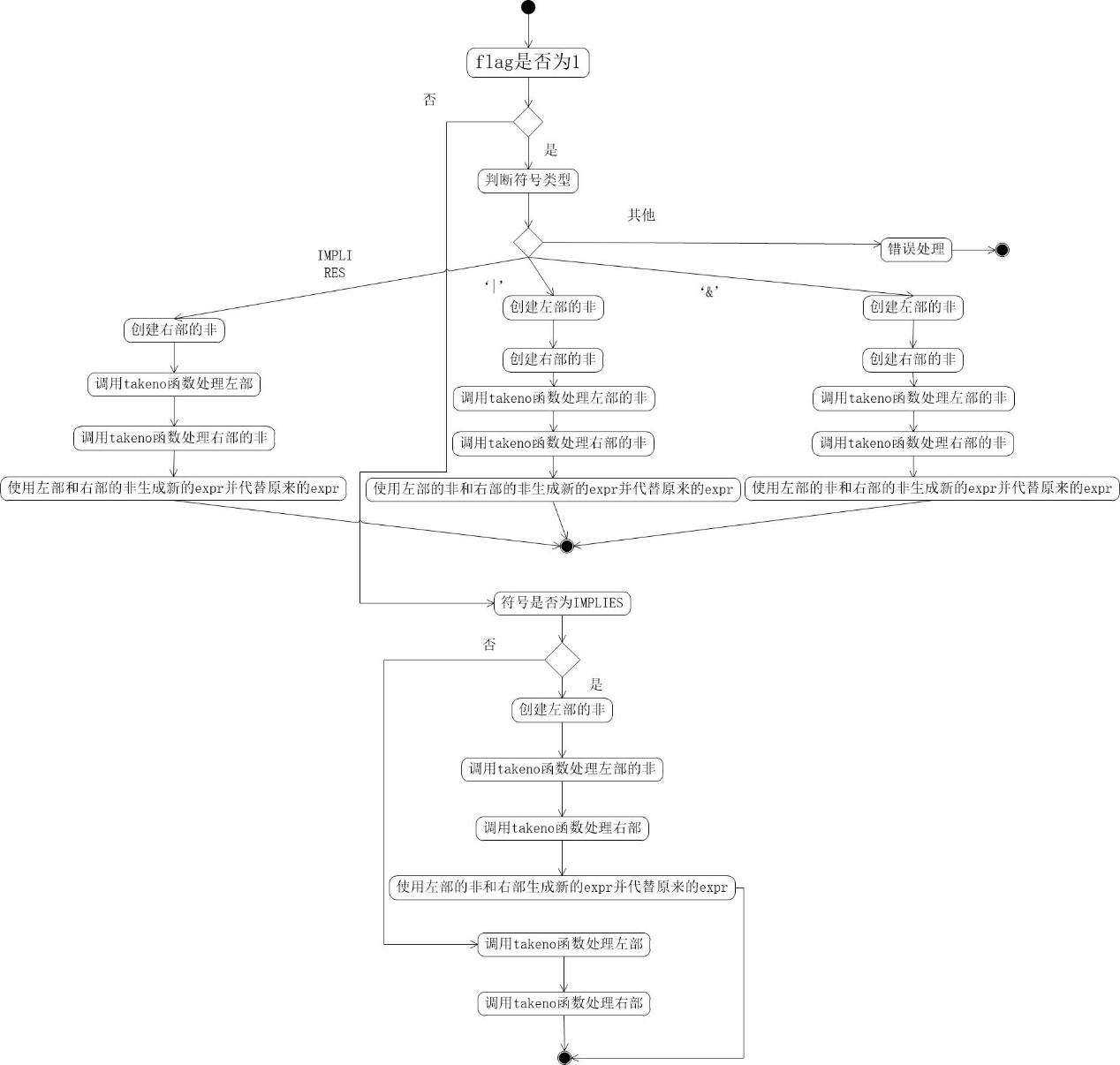
该部分描述函数int generate\_invariants()



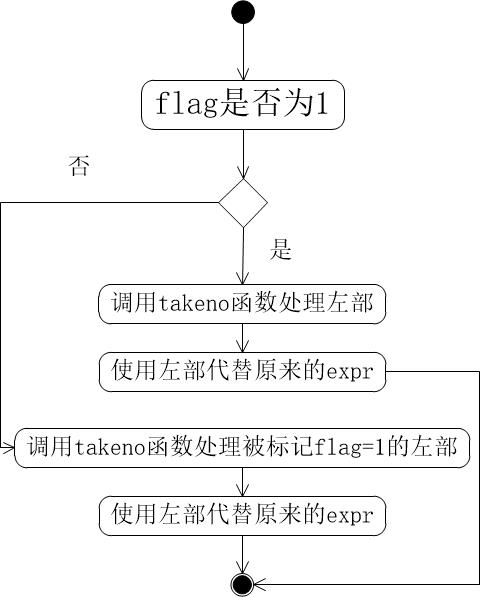
### 3.2.14 对！进行处理的函数takeno cpp\_code.cpp: 4650-4837

该部分描述的是expr类及其派生类中的virtual void takeno(expr \*\*exprn,int flag)；其中boolexpr、notexpr、compexpr、equalexpr、designator、quantexpr结构体中的sign函数为主要函数，其他函数均为过渡函数，由于过渡函数比较简单，在此不做描述。

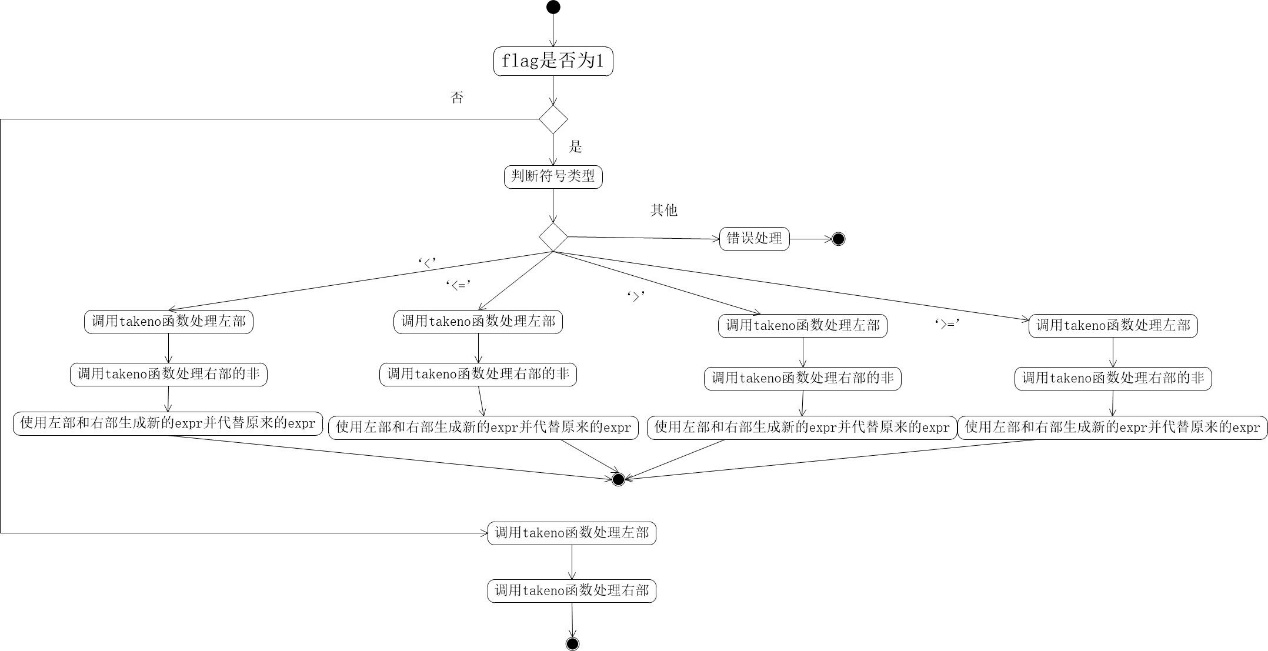
1.下图为函数void boolexpr::takeno(expr \*\*exprn,int flag) cpp\_code.cpp:4654-4740



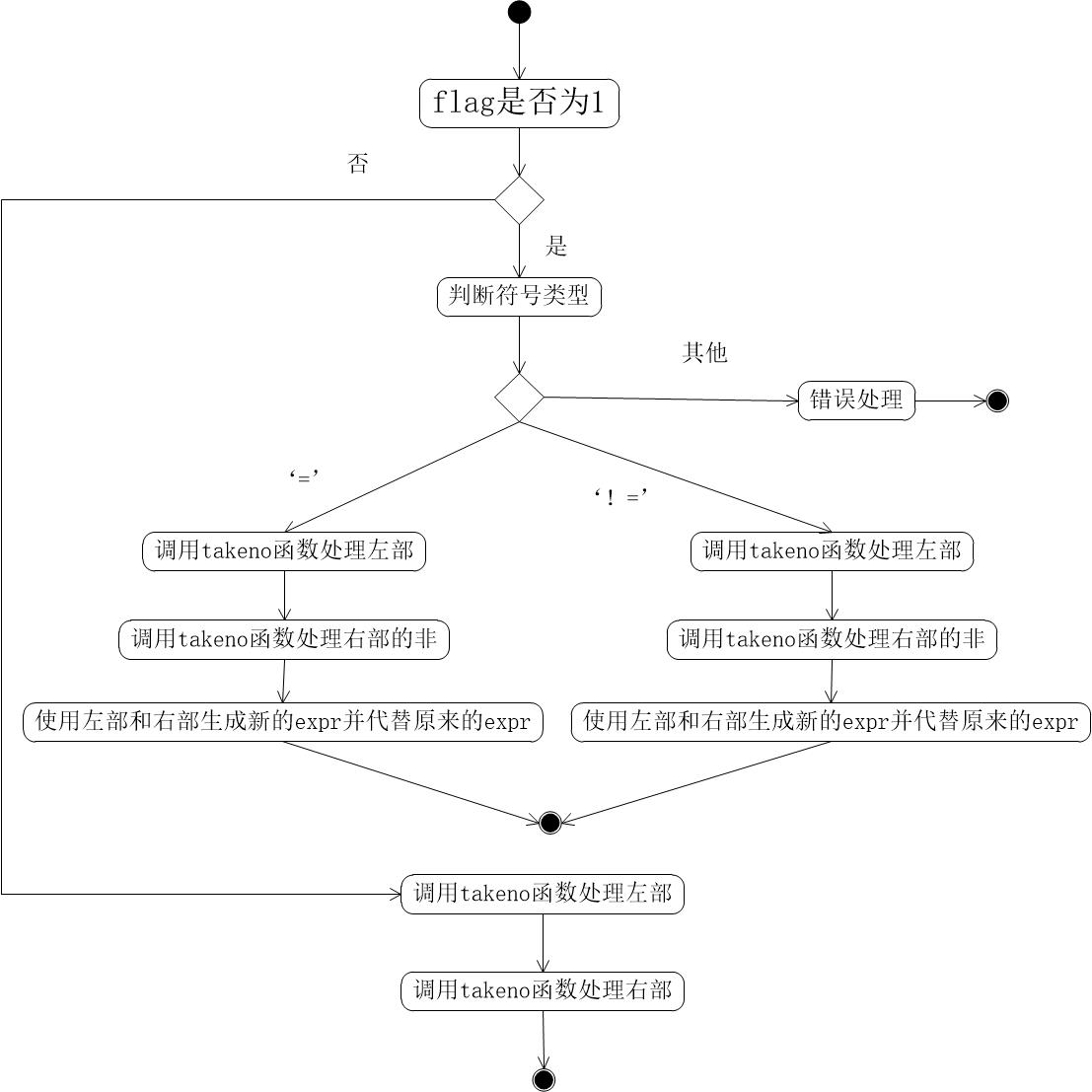
2. 下图为函数void notexpr::takeno(expr \*\*exprn,int flag) cpp\_code.cpp:4705-4717



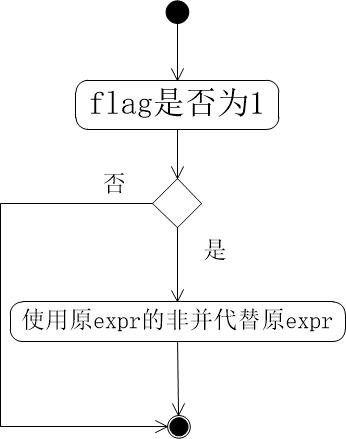
3. 下图为函数void compexpr::takeno(expr \*\*exprn,int flag) cpp\_code.cpp:4718-4754



4. 下图为函数void equalexpr::takeno(expr \*\*exprn,int flag) cpp\_code.cpp:4756-4787



4. 下图为函数void designator::takeno(expr \*\*exprn,int flag) cpp\_code.cpp:4629-4639和void quantexpr::takeno(expr \*\*exprn,int flag) cpp\_code.cpp:4819-4829



## 3.3 cpp\_code.cpp中新加的变量说明

|  |  |
| --- | --- |
| 名称 | 用途 |
| int forallnum | 用于记录forall的个数 |
| int rnum | 用于记录rule的个数 |
| int formark | 用于记录是否正在生成for结构。  formark = 0 表示不在生成for结构的过程中  formark = 1 表示正在生成for结构的过程中 |
| int ruleflag | 用于记录是否正在生成ruler结构。  ruleflag = 0 表示不在生成rule结构的过程中  ruleflag = 1 表示正在生成rule结构的过程中 |
| int condflag | 用于记录是否正在生成卫式结构。  condflag = 0 表示不在生成卫式结构的过程中  condflag = 1 表示正在生成卫式结构的过程中 |
| simplerule \*startrule | 用于记录startstate的指针 |
| simplerule \*invrule | 用于记录不变式的指针 |
| int cflag | 用于记录是否为const类型。  cflag = 0 表示不是  cflag = 1 表示是 |
| int tflag | 用于记录是否为type类型。  tflag = 0 表示不是  tflag = 1 表示是 |
| int vflag | 用于记录是否为var类型。  vflag = 0 表示不是  vflag = 1 表示是 |
| int lflag | 用于重置“是否声明”标记 |
| int blanknum | 用于打印缩进 |
| int andflag | 用于标记是否打印‘&’后的缩进  andflag = 0 表示不打印  andflag = 1 表示打印 |
| int isforall2 | 用于标记是否是forall或exist，用于打印缩进 |
| int invflag | 用于标记是否正在打印不变式，  invflag = 0 表示不是  invflag = 1 表示是 |