## 3目标代码溯源单元

### 概述

目标溯源是对工程中一个给定目标点查找其所有终结符(外部输入源和常量)的过程。我们需要获得的是目标点到其终结符的路径信息。

此处添加几个概念描述

1. 无来源：当前节点即为终止符

2. 单来源：和当前节点有关的上层来源节点仅有一个

3. 多来源：和当前节点有关的上层来源节点有多个

4. 定义域：对于一个函数，一个文件乃至一个工程来说 ,一个变量可能会出现多次，而所有出现中属于定义点的出现组成的集合被称为定义域。

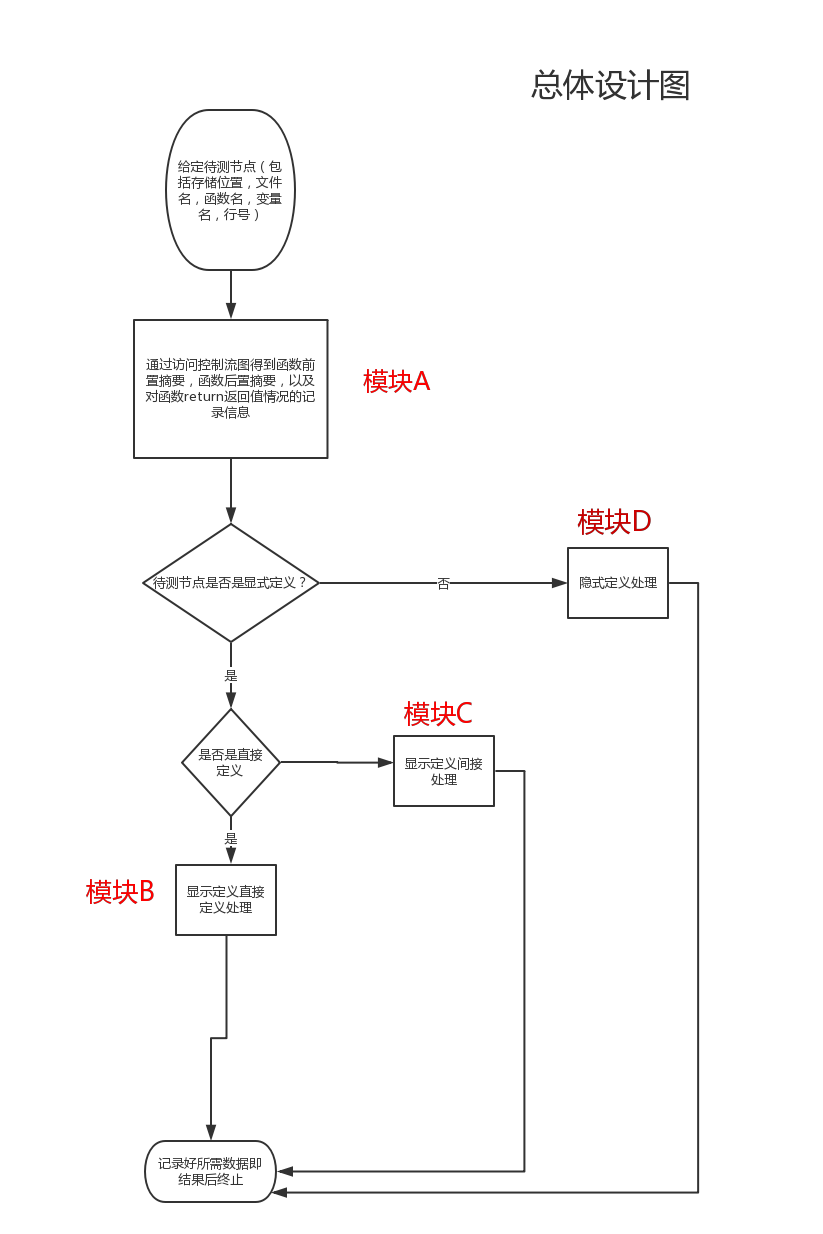
5. 定义轨迹：对于一个变量的而言，其定义可能是间接定义，即为其定义的元素不是直接定义点，因此需要递归寻找为其定义的元素的定义点，直至找到的定义点为直接定义点为止。这个寻找过程的所有定义点组成的有向路径称之为定义轨迹。

单来源的上层来源既有可能是多来源，也有可能是单来源。通过逐步递归迭代可以找出整个定义域以及整个定义轨迹即是我们溯源的目的。针对一个给定的源代码目标点，首先判断目标点的类型（定义点，使用点，参数传递，定义后再使用或者使用后再定义）然后根据其类型进行分别处理。（1）若其为定义点，则通过控制流图寻找为其定义的使用点。若为其定义的使用点为库函数等外部输入源或者常量等终结符则停止溯源，否则分别对其所有使用点利用深度优先搜索进行溯源。（2）若为使用点，则通过结合控制流图和抽象语法树来寻找其定义点，进而对其定义点进行溯源。（3）若为参数传递，则能够通过函数调用关系（可由遍历语法树得到）得出此参数所在函数被哪些函数调用，也可以知道参数所在函数在在什么位置被调用（遍历抽象语法树可以得到），此位置我们记为P，同时也可以得到函数参数的相对位置，即索引,记为index，假设将这些函数记为列表A。然后依次跳转进入列表A中所包含的函数内，根据索引index和位置P可以定位出原参数在此函数内的直接来源，记为H。即将函数间溯源问题转换成了函数内的溯源问题，接着继续溯源H即可。（4）若目标点类型为定义后再使用（例如 i = 10; ++ i），这种情况可以向上溯源两层寻找其定义点，即此处++i为定义后再使用，寻找其两层定义点分别为 i = 10中i和 ++i中i，然后继续溯源第二层定义点即可。（5）若为使用后再定义（例如i = 2;i++），寻找其使用点的定义点可以找到i=2中的i，继续溯源即可。

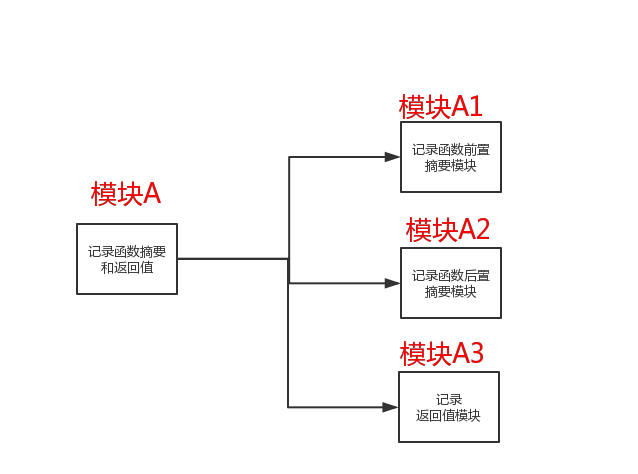
### 处理流程

溯源的过程是通过借助于抽象语法树和控制流图以及定义使用链来进行的。

总体设计思路见下图。



注：此处的模块A是遍历控制流图时的记录模块，不是处理模块（摘要处理模块包含在了B，C，D模块里）



过程间问题目前暂时分为3类：参数，函数副作用，return

例子1（参数问题）

1 f1() {  
2 int a = 1;

3 int b = 2;

4 f2(a, b);

5 }

6 f2() {int a , int b} {

7 int c = a – b;

8 }

对于这种参数问题，分析变量在被调用函数中，处理办法是找它的前置摘要

即当初遍历代码时，在f1中遍历到f2的时候要记录下前置摘要信息。可以以这样一种形式记录：HashMap<Position , D(s,c,v,p)>

其中 Position: <文件名，函数名，行号，参数index(即第几个参数))

p :<文件名，函数名，行号>

对于此例即为 f2: <文件名，f1 , 4 , 1)—> <null , 1 , null , <文件名，f1 , 2 >>

然后分析变量c时，要溯源到第7行的变量a,和b,也即函数f2的传入参数，然后在通过前置函数摘要信息找到各个参数的D(s,c,v,p).

例子2（函数副作用）

这种分类主要针对全局变量以及地址传递（主要就是传入指针）

1 int a = 0;

2 f1() {

3 a = 1;

4 }

5 f2() {

6 f1();

7 b = a;

8 }

对于这种情况，待测变量（b）不在被调用函数中，处理办法是找它的后置摘要信息。

用一个映射HashMap<NameOccurrence , D(s,c,vp)>来处理

其中NameOccurrence ： <变量名 ： 行号>

例如此题 Name Occurrence就应该记录下<a : 1> ,<a: 3>

而分析变量b的时候要追溯到变量a,而7行的a并不是终结符，但是显然咱暂时已经无法继续溯源，故借助NameOccurrence , 将第3行，第1行均加入追溯轨迹。（潜在问题，NameOccurrence保存的信息太少，可能处理复杂情况会失灵）

对于全局变量而言，还有一种常见的简单隐式定义情况（即不通过调用函数定义）

1 int a = 0;

2 f2() {

3 b = a;

4 }

例如此例变量b溯源到第3行a，此时a并不是终结符，然后再借助NameOccurrence将全局变量a的其他定义点加入溯源路径即可。

例子3（针对函数返回值赋值情况）

1 int f(int x , int y) {

2 return x + y;

3 }

4 void f2() {

5 int m = 3 , n = 4;

6 int x = f(m , n);

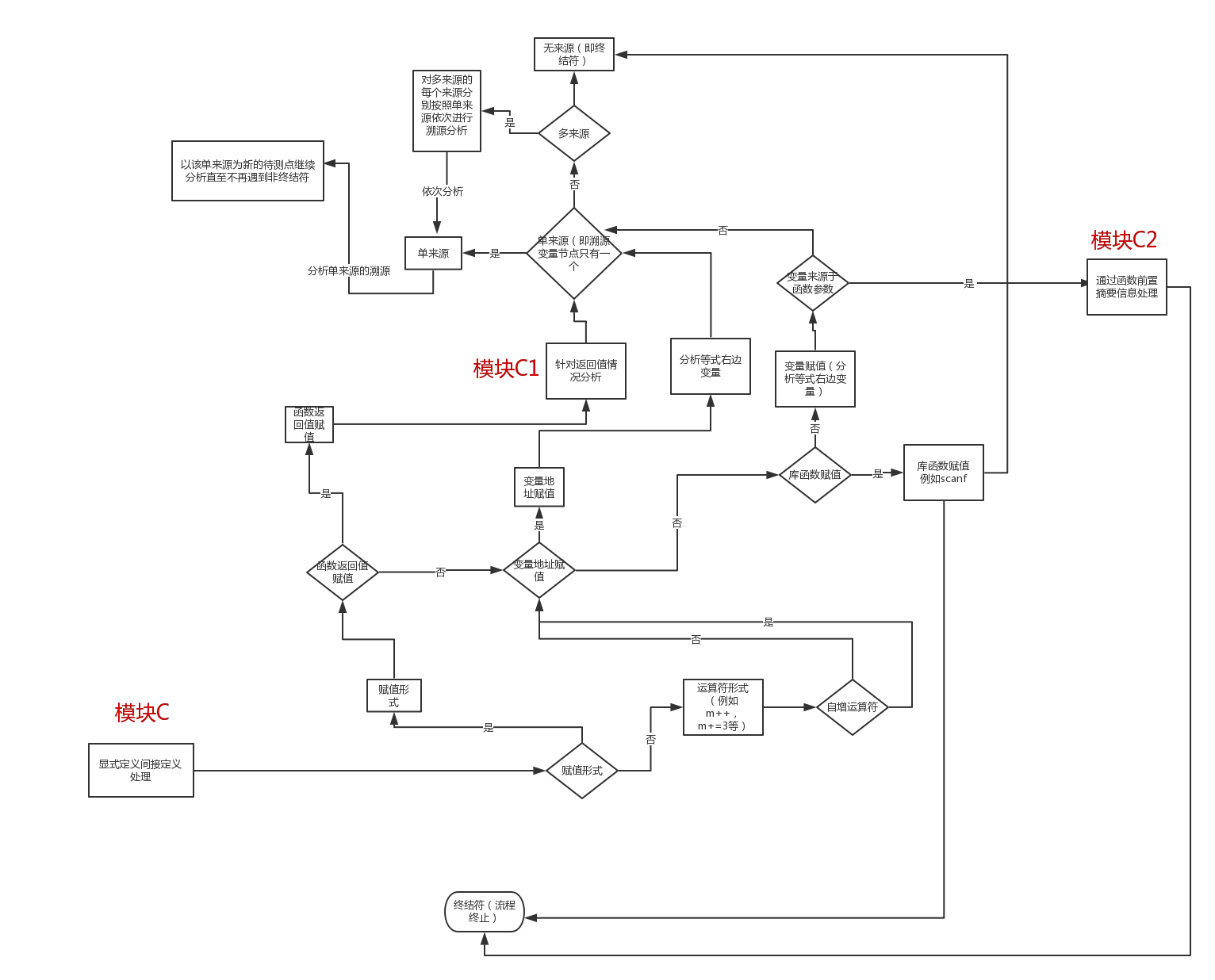
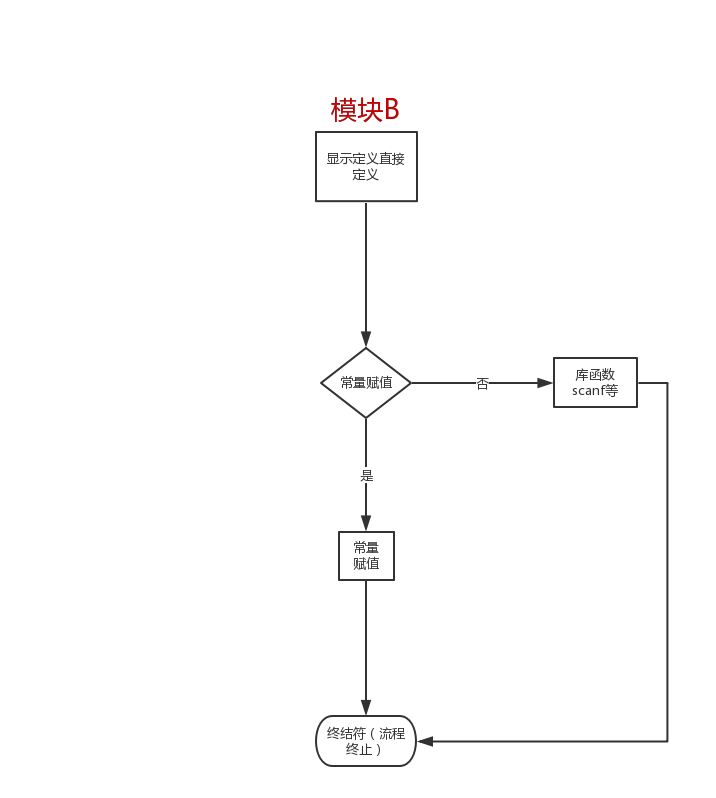
7 }

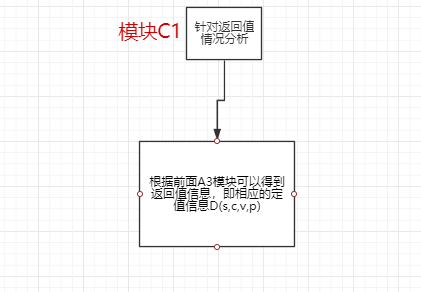
对于函数返回值赋值问题，我们可以确定需要跳转到函数，但是不能确定函数返回值是否确定与参数有关联（此例有关联， 但加入return 100 ; 就跟参数无关），

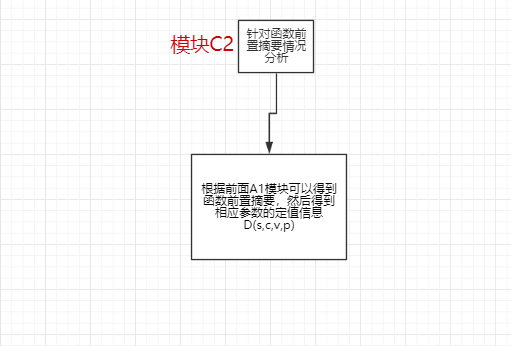
所以，我们需要记录追踪的并不是参数，而应该是返回值的D(s,c,v,p)

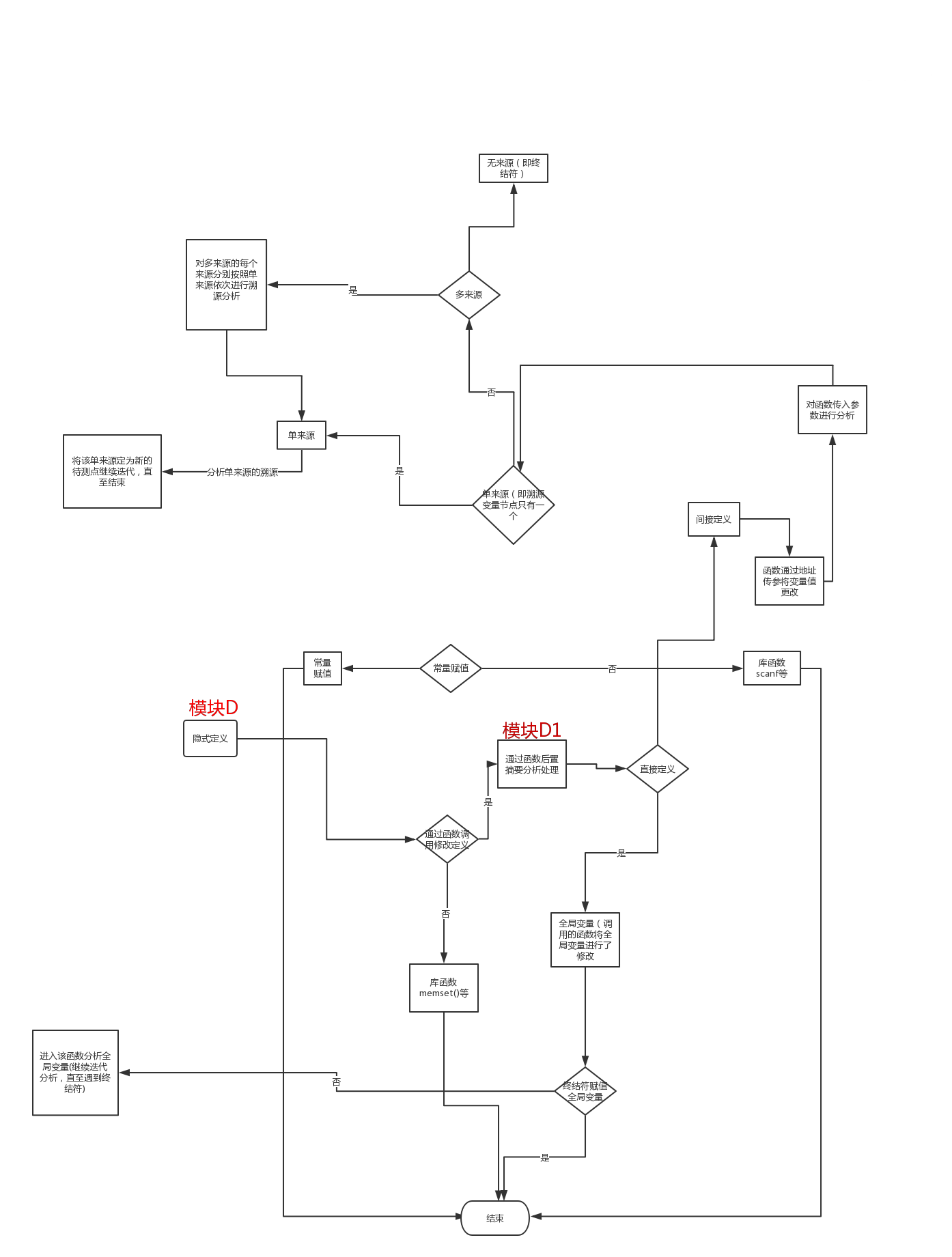
对此题而言即为D(s,c,v,p): <+ , null ,(x , y ) , <文件名，f , 2>)

然后我们再分析返回值中涉及的变量的溯源即可



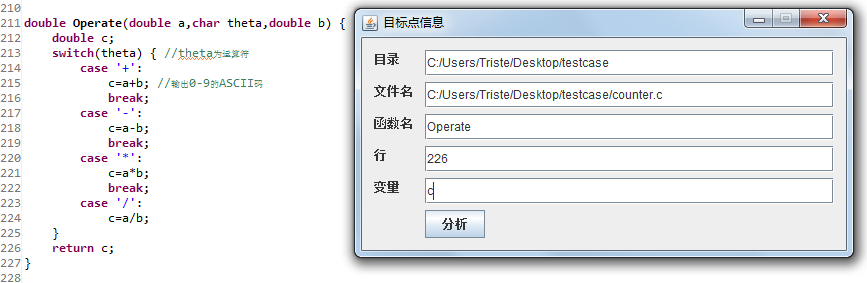




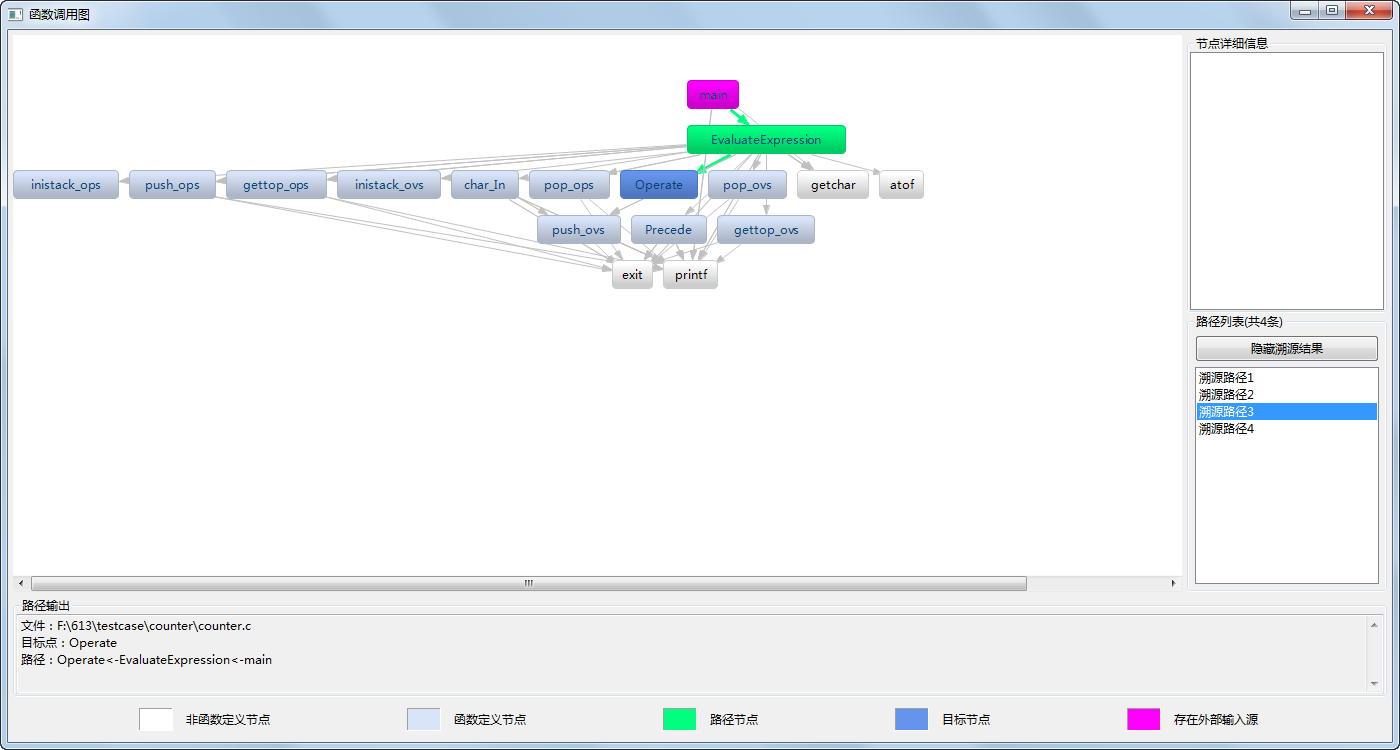


#### 溯源分析界面

设置目标点



2.点击分析进入如下页面



窗口中间最大的部分是函数调用图的显示区域，右侧自上而下依次是“节点详细信息”、“溯源路径”，窗口下边是“路径输出”。用户可以在右侧“溯源路径”中点选相应的溯源路径，在图中就会用绿色标记显示该路径将会执行的路径，下方的“输出”则会输出该路径的详细信息。

任意点击一条溯源路径，紫色表示其有外部输入源信息

