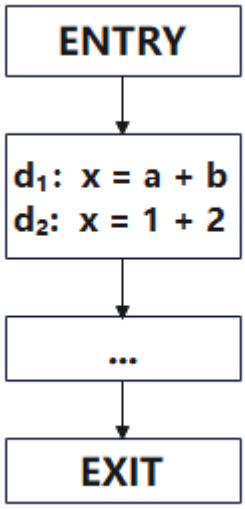
概念介绍

为了有效地优化代码，编译器需要在程序的各个节点建立并求解与信息有关的方程来收集数据流信息，并将这些信息分发给流程图的每个块，这个过程被称为数据流分析[1]。

通过检查程序的一部分(基本程序块)，编译器可以执行一些优化。如下图所示：

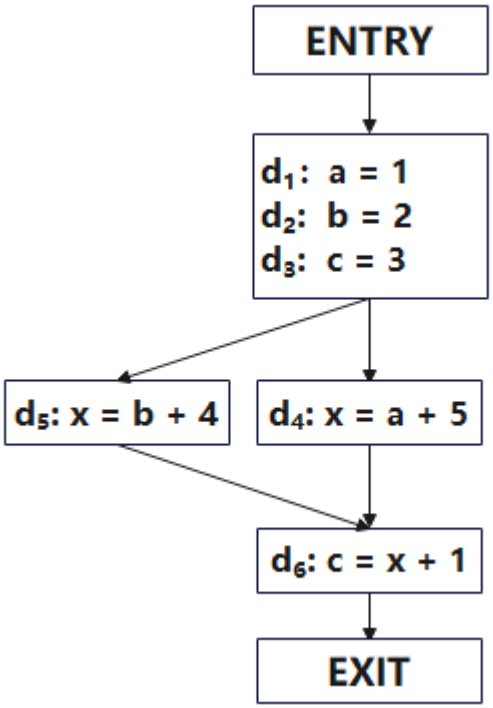


(1)在此程序中, d_1是无用的，d_1中x的值永远不会在程序中被使用；

(2)在编译时，d_2中的表达式(1 + 2)将被计算；

所以该基本程序块可以被优化为：x = 3。

但是有一些优化必须通过检查整个程序来实现。如下图所示：



从图中可以看出：d_3对c的初始化赋值是无用的，x的值恒定为6，d_6可以简化为：c = 7。

从上面这个例子可以看出，仅通过一两个连续语句，编译器无法发现这些优化信息。必须通过更加全面的数据流分析，以便编译器在程序的每个节点都了解以下内容：

(1)保证哪些变量具有恒定值

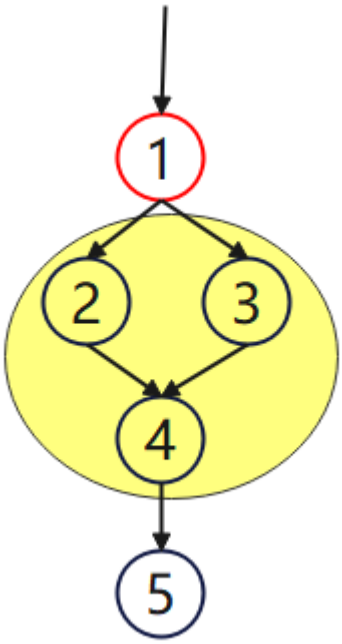
(2)重新定义之前将使用哪些变量

这篇文章将会介绍一种数据流分析方法——区域分析，在正式开始之前，我们先来了解几个概念：

(1)严格支配：如果不首先通过x就不可能到达w，则称x严格支配w。

(2)支配：如果x严格支配w或x=w，则称x支配w。

以下图为例，如果想要到达2、3或4，必须要先通过1，所以在{1,2,3,4}集合中1占支配地位，1严格支配集合{2,3,4}。



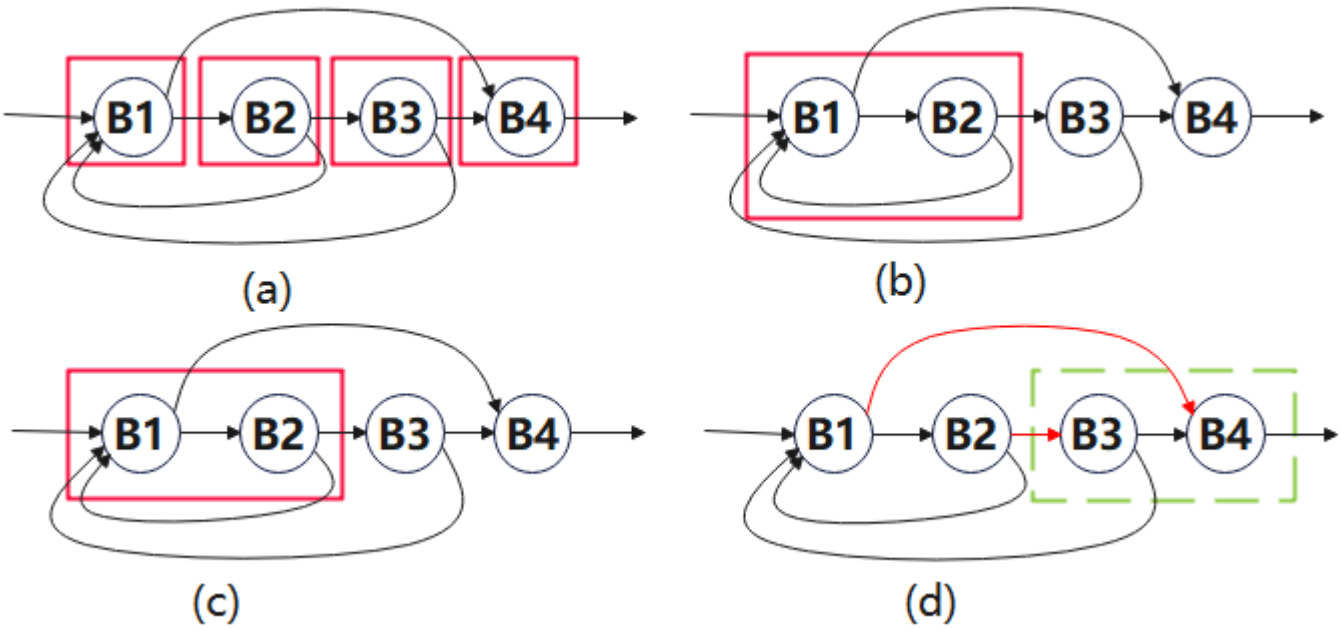
流图的区域是节点N和控制流边E的集合，必须满足以下条件：

(1)N中存在一个节点头h，它支配了N中的所有节点。

(2)如果存在节点m可以在不经过节点头h的情况下到达N中的节点n，则m也必然在N中。

(3)E是N中节点之间所有控制流边的集合，进入h的循环边可以不包含在内。

以下图为例，节点B1、B2、B3和B4都可以单独看做一个区域(图(a))；节点B1和B2以及边B1->B2和循环边B2->B1组成的循环也是一个区域(图(b))；根据区域的第三个条件，循环边可以不包含在区域内，所以节点B1和B2以及边B1->B2也可以形成一个区域(图(c))。



然而，图(d)中节点B3、B4以及边B3->B4组成的子图(虚线部分)不形成区域，因为控制流既可以通过节点B3处进入子图，也可以通过节点B4进入子图。B3和B4都无法做到完全支配另一个，不符合区域的第一个条件。即使我们选择B3作为头h，也不符合区域的第二个条件。因为B1可以不经过B3，沿着边B1->B4到达B4，根据区域的第二个条件，B1应该在该“区域”中，但这显然不正确。

区域分析的基本思想

(1)对于每个区域R，以及R中的每个子区域R'，我们计算一个传递函数，该函数总结了从区域R开始到基本块B结束执行所有可能路径的效果。

(2)如果基本块B是区域R的出口块(即区域R内的基本块B有到R外的某个块的传出边)，计算区域R的每个出口块B的传递函数就是计算从区域R的入口通向B过程中，执行所有可能路径的效果，即整个区域R的传递函数。

(3)从单个基本块组成的区域开始，逐步构造更大的区域，计算更大区域的传递函数。

(4)在构造更大的上层区域时，如果区域R的边在R的上层区域上形成一个非循环流图。我们可以继续按上层区域的拓扑顺序计算传递函数。

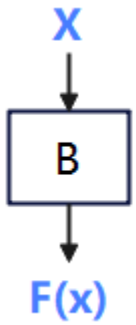
(5)如果R是一个循环区域，那么我们只需要考虑循环边对R入口节点的影响。

(6)直到整个程序组成一个区域，并计算整个程序组成的区域P的传递函数，若入口节点处的初始值为V，则：

- $out[B] = F_{P,B}(V)$
- $in[B] = \wedge_{B'} out[B']$（B'是B的前置块，即B'是有到B的边的块）

传递函数

在一个语句之前和之后的数据流值受该语句的语义约束，即语句前后的程序点的数据流值受该语句语义的约束，这种约束关系称为传递函数。基本块的传递函数表示为(以下都以Reaching Definitions 为例)：F(x) = Gen U (x - Kill)

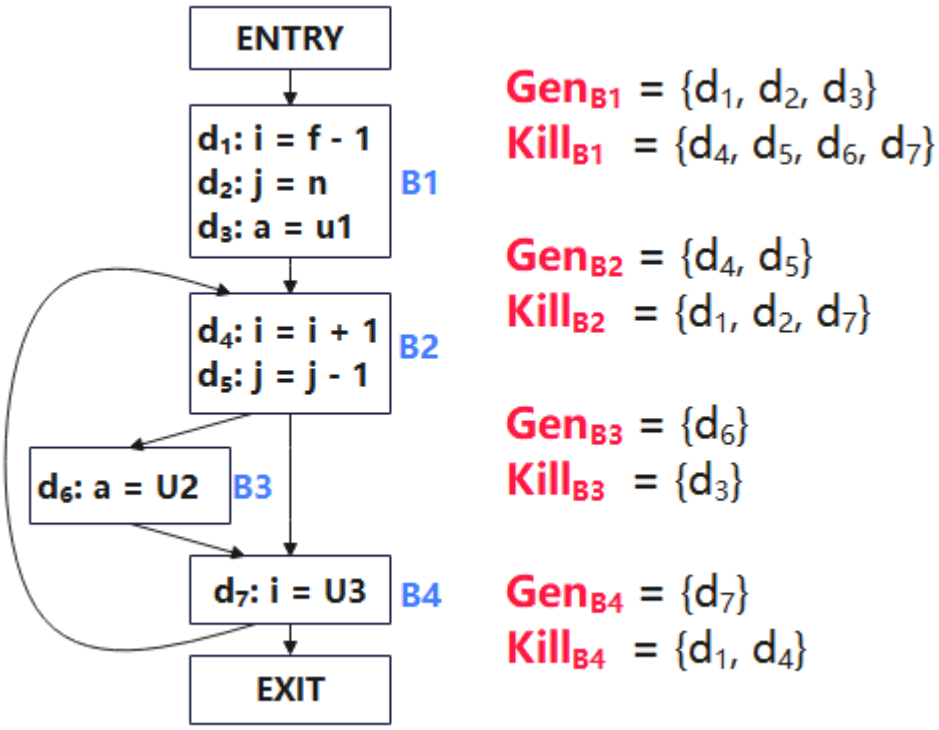


$$F(x) = Gen \cup (x - Kill)$$

$$Kill = Kill_1 \cup Kill_2 \cup \dots Kill_n$$

$$Gen = Gen_n \cup (Gen_{n-1} - kill_n) \cup \dots (Gen_1 - kill_2 - Kill_3 - \dots - Kill_n)$$

其中，x表示基本块B的输入；F(x)表示基本块B的输出；kill表示被基本块B中各语句杀死的变量的集合；Gen表示基本块中没有被各语句杀死的定值的集合。



上图显示了流图中各基本块的Gen和Kill集合。以基本块B1为例，该基本块有三条语句：

(1)d_1: i = f-1，“生成”了一个对变量i的赋值d_1，并“杀死”了程序中其它对i的赋值，即d_4和d_7；

(2)d_2: j = n，“生成”了一个对变量j的赋值d_2，并“杀死”了程序中其它对j的赋值，即d_5；

(3)d_3: a = u1，“生成”了一个对变量a的赋值d_3，并“杀死”了程序中其它对a的赋值，即d_6。

所以基本块B1的Gen为{$d_1$1,d_2,d_3}，kill为{d_4,d_5,d_6,d_7}。

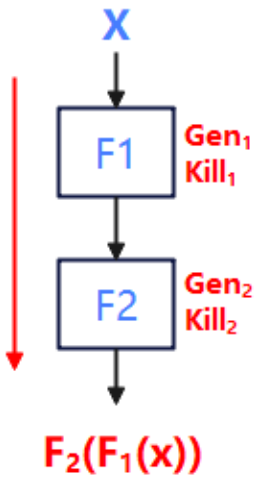
对于区域分析，构造更大的区域实际上就是更新了Gen和Kill集合的值。

关于传递函数的必要假设

为了使区域分析发挥作用，我们需要对区域中传递函数集的属性做出某些假设。具体来说，我们需要对传递函数进行三个基本操作：组合、汇聚和闭包。

1. 组合(composition)

节点序列的传递函数可以通过各个节点的传递函数的组合来导出。设F_1和F_2是两个节点的传递函数，执行F_1后执行F_2的效果用$F_2(F_1(x))$表示：

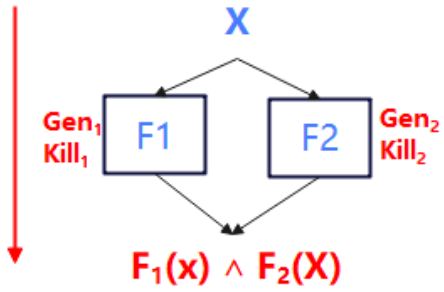


$$F_2(F_1(x)) = Gen_2 \cup (F_1(x) - Kill_2)$$
$$= Gen_2 \cup ((Gen_1 \cup (x - Kill_1)) - Kill_2)$$
$$= Gen_2 \cup (Gen_1 - Kill_2) \cup (x - (Kill_1 \cup Kill_2))$$

组合操作后的Gen值　　组合操作后的Kill值

2. 汇聚(meet)

汇聚用于导出执行路径不同，但输入端点与输出端点相同的节点。一般用$F_1(x) \wedge F_2(x)$表示：

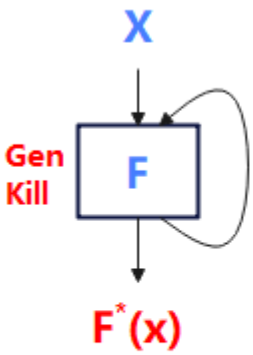


$$F_1(x) \wedge F_2(X) = Gen1 \cup (x - Kill1) \cup Gen2 \cup (x - Kill2)$$
$$= (Gen_1 \cup Gen_2) \cup (x - (Kill_1 \cap Kill_2))$$

汇合操作后的Gen值　　汇合操作后的Kill值

3. 闭包(closure)

如果F表示循环的传递函数，那么F_n表示循环n次的效果。在迭代次数未知的情况下，我们必须假设循环可以执行0次或多次。我们用F*(x)表示这样一个循环的传递函数，则F的闭包如下图所示：



$$F^*(x) = \wedge_{(n\geq 0)} F^n(x)$$

$$= x \wedge F(X) \wedge F(F(x)) \wedge \ldots$$

$$= x \cup (Gen \cup (x - Kill)) \cup Gen2 \cup (x - Kill2)$$

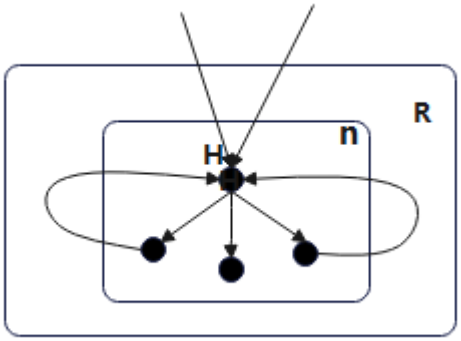
$$= Gen \cup (x - \varnothing)$$

闭包操作后的Gen值　　闭包操作后的Kill值

处理可简化流图

可简化流图是指那些可以通过以下两个规则转换简化为单个节点的流图：

(1)T1：删除循环。如果n是具有循环的节点，即边n->n，删除该边(n的所有此类边)。

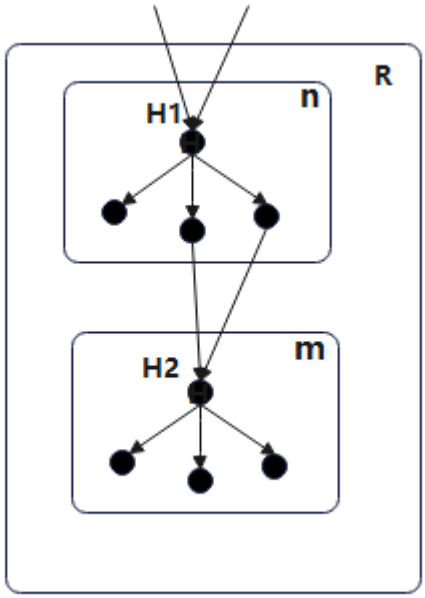


如上图所示：R是执行T1规则后形成的新区域，原区域n的头结点H也是新区域R的头结点，n中H到每一个基本块B的传递函数为$F_{n,B}$，则在新区域R中的传递函数为：

- $F_{R,in(H)} = (\wedge_P F_{n,P})^*$，P是区域R的头节点H的前置节点；
- $F_{R,B} = F_{n,B}(F_{R,in(H)})$

(2)T2：删除顶点

如果有一个节点n具有唯一的前置节点m，则将m和n合并[2]。



如上图所示：R是执行T2规则后形成的新区域，对于n中的基本块B，传递函数未改变(F_{R},B = F_{n},B)；对于m中的基本块：

- $F_{R,in(H2)} = \wedge_P F_{R,P}$，P是基本块H2的前置节点；
- $F_{R,B} = F_{m,B}(F_{R,in(H2)})$

为了构建区域的层次结构，我们需要识别循环。在可简化流图中(这里，我们假设流程图都是可简化的)，任何两个循环要么是不相交的，要么一个嵌套在另一个循环中。可简化流图解析到循环层次结构时，先将每个基本块本身作为一个区域。我们将这些区域称为叶区域。然后，从里到外排序循环，即从最里面的循环开始。处理循环时，我们通过两个步骤将整个循环替换为一个节点：

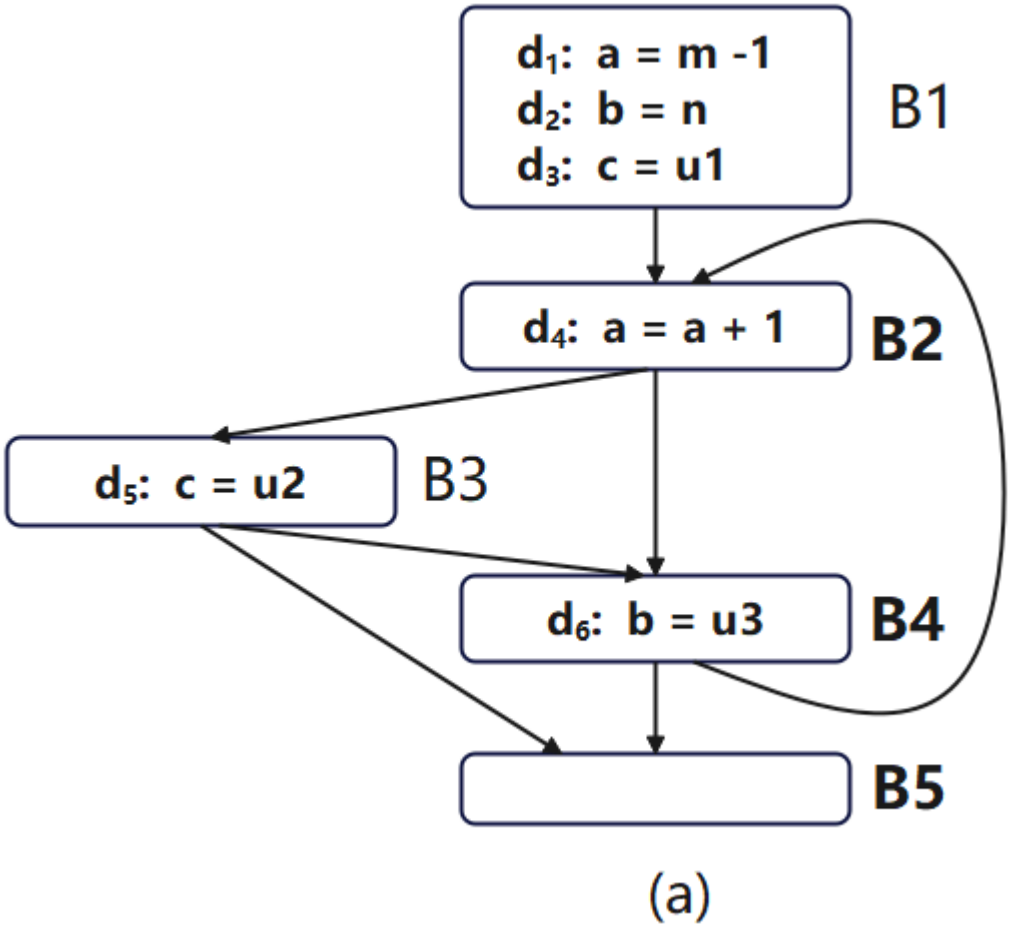
(1)将循环L的主体(除报头的循环边外的所有节点和边)替换为代表区域R的节点。L报头的边现在并入R的节点。循环L的任何出口的边被R到同一目的节点的边替换。但是，如果控制流边是循环边，那么它就会成为R上的循环边。我们称R为主体区域。

(2)构造一个代表整个循环L的区域R'。我们可以把R'称作循环区域。R和R'之间唯一的区别是，后者包括循环L的循环边。换句话说，当R'替换流图中的R时，我们所要做的就是将R的循环边删除[3]。

通过重复上述操作，我们可以逐渐将大的循环减少到单个节点。由于可简化流图的循环是嵌套的或不相交的，因此循环区域的节点可以表示在此简化过程中构建的一系列流图中循环的所有节点。

最终，所有循环都被简化为单个节点。此时，流程图有两种情况：一是简化为单个节点；二是有几个节点剩余，具有HO循环(即，简化的流图是多个节点的非循环图)。在前一种情况下，我们完成了区域层次结构的构建，而在后一种情况下，我们为整个流图再次构建出一个主体区域。

以下图为例，图(a)为控制流图。此流程图中有一个循环边(B4->B2)。区域的层次结构如图(b)所示，共有8个区域：



(1)区域R1-R5分别代表块B1-B5的叶区域。每个块也是其区域中的出口块。

(2)区域R6表示流图中唯一循环的主体；它包含区域R2、R3和R4以及三个区域间控制流边R2->R3、R2->R4和R3->R4。它有R3和R4两个出口块，因为它们都有不包含在区域中的输出边。图(c)显示了R6简化为单个节点的流程图。请注意，边R3->R5和R4->R5都被边R6->R5取代。因为从R3和R4两个区域的输出都将到达到达R5的输入，因此简化后用一条边代表之前两条边。

(3)循环区域R7代表整个循环。它包括一个子区域R6和一个循环边R4->R2。它还有两个出口节点，也就是R3和R4。图(d)显示了整个循环简化到R7后的流程图。

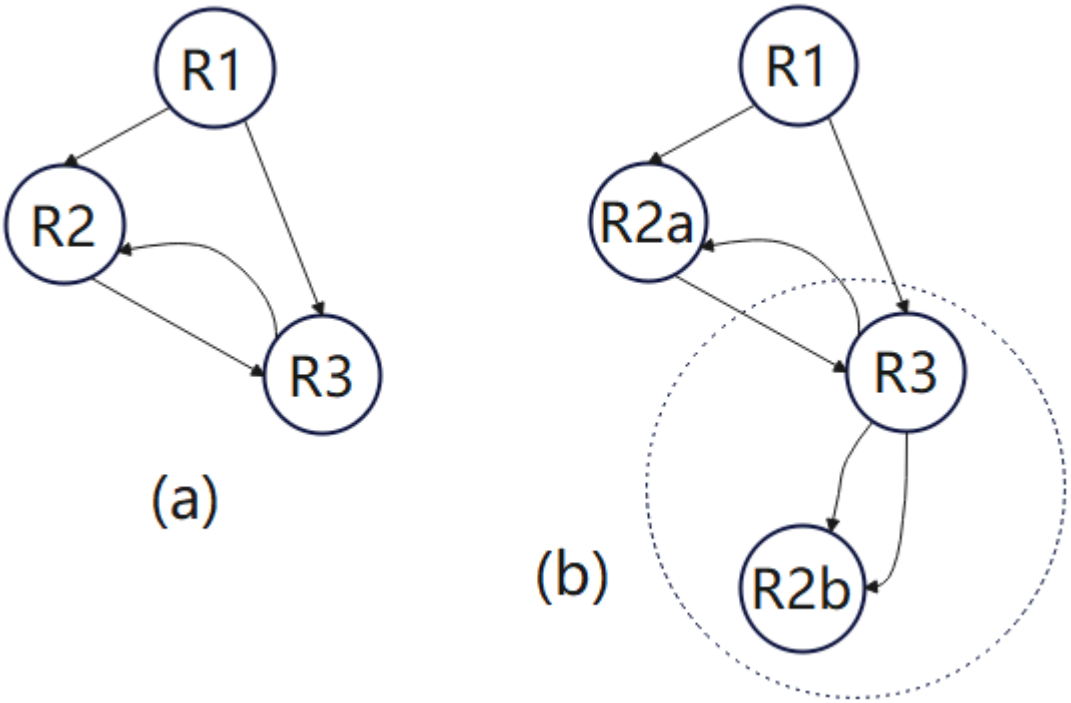
(4)最后，主体区域R8是顶部区域。它包括三个区域，R1、R7、R5和三个区域间控制流边R1->R2、R3->R5和R4->R5。当我们将流程图缩小到R8时，它将成为一个单一节点。由于其头B1没有循环边，因此不需要将此主体区域减少为循环区域。

处理不可简化流图

对于不可简化流图，我们建议使用迭代数据流分析算法来进行数据流分析。但是如果只是偶尔处理不可简化流图的话，可以使用节点分裂方法进行分析。

下面图(a)就是一个典型的不可简化流图。R2和R3之间存在循环，但R2和R3又都不占支配地位，导致我们无法进一步解析该图。我们选择一些区域R(如R2)，该区域R具有多个前置节点(R2的前置节点为R1和R3)，并且不是整个流图的头。如果有k个前置节点，则制作流图R的k个副本，并将每个前置节点连接到R的不同副本。这里需要注意，只有区域的头才可能有该区域之外的前置节点。在识别新的循环边并构建其区域后，这种节点分裂使得区域数量减少。由此产生的流图可能仍然不可简化，但通过分裂阶与新的循环被识别并折叠到区域的阶段交替使用，我们最终只剩下一个区域，即流图已经被约化。

图(b)中所示的分裂将边R2b->R3变成了循环边，此时R3支配R2b,这两个区域可以合并为一个新区域。由此产生的三个区域(R1、R2a和新区域)形成一个非循环图。此时，我们可以将整个流程图简化为单个区域。一般来说，可能需要额外的拆分，在最坏的情况下，基本块的总数可能会成为原始流图中块数的指数。



总结

本文简单介绍了一种解决数据流问题的方法——区域分析。在区域分析过程中，我们首先为基本块创建传递函数，然后通过组合、汇聚和闭包等操作总结加大区域的传递函数，最终构造出整个数据流图的传递函数。通过区域分析等数据分析的方法，可以发现更多的优化机会，如将代码从循环内部移动到循环外部、冗余的代码删除等。

参考

1. https://www.srcmini.com/9866.html

2. https://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15745-s19/www/lectures/L18-Region-Analysis.pdf

3. https://www.brainkart.com/article/Region-Based-Analysis_8197/