Sparse Analysis and Path Conditions Transform

枫聆

2022 年 1 月 9 日

目录

The Definition of Sparse Analysis

Definition 1.1. 定义 simple language 如下

$$
\begin{aligned}
\text{Program } P &:= \quad F+ \\
\text{Function } F &:= \quad f(v_1, v_2, \cdots) = \{S;\} \\
&\quad\ \mid f(v_1, v_2, \cdots) = \emptyset \\
\text{Statement } S &:= \quad v_1 = \langle v_1 \rangle && :: \textbf{identity} \\
&\quad\ \mid v_1 = v_2 && :: \textbf{assignment} \\
&\quad\ \mid v_1 = v_1 \oplus v_2 && :: \textbf{binary} \\
&\quad\ \mid v_1 = ite(v_2, v_3, v_4) && :: \textbf{if-then-else} \\
&\quad\ \mid v_1 = f(v_2, v_3, \cdots) && :: \textbf{call} \\
&\quad\ \mid \textbf{return } v_1 = v_2 && :: \textbf{return} \\
&\quad\ \mid \textbf{if } (v_1 = v_2)\{S_1;\} && :: \textbf{branching} \\
&\quad\ \mid S_1; S_2 && :: \textbf{sequencing}
\end{aligned}
$$

其中 $ite(v_2, v_3, v_4)$ 是一个三元表达式. 每个 function 都只有一个 return statement 作为它的唯一 exit node; function 的开头都对应的 identity function 对每一个 function parameter 进行初始化.

Definition 1.2. 给定 program P 上的两个 statements x, y 和一个 branch condition z. 若 x 中使用到了 y 中定义的变量，则称 x 数据依赖(data-dependent) 于 y; 若 x 在运行时被执行当且仅当只有当 z 可达且值为 true，则称 x 控制依赖(control-dependent) 于 z. 特别地，关于数据依赖可进一步推广至位于 statement 上的 variables 之间.

Annotation 1.3. 控制依赖更加严格的定义应该是这样: 给定 CFG 上两个不同的结点 x, y，若满足下述条件:

- 在 CFG 存在一条从 x 到 y 的 nonempty path p 满足对任意的 $v \in p$ 且 $v \neq x$ 都有 y !pdom v，其中 pdom 表示 postdominate;
- y !pdom x.

则称 y 控制依赖于 x. 简而言之存在某个 x 的后继 $x.\text{succ}_i$ 使得 y pdom $x.\text{succ}_i$，但 y !pdom x.

这里 CFG 基于 SSA 形式 (后续提到的 CFG 亦是如此)，但是关于 dominate tree 有点不同，往常我们定义 dominate tree 都是以基本块为单位，这里直接是 CFG 上的结点，所以你要推广一下: 一个基本块上的结点根据顺序线性关系两个相邻的结点构成 immediate dominance，再把基本块之间的支配关系放到原来两个基本块的结尾和开始结点.

Definition 1.4. 给定 simple language 上的 program P，定义它的program dependence graph $G = (V, E)$ 如下

- 对任意结点 $v \in V$，v 表示 P 上的一个 statement 或者 statement 中某个变量.
- E 包括两种 egdes 组成
  - data dependence edges: 对任意两个 variables x, y，若 y 的定义数据依赖于 x，则 $(x, y) \in E$，所有这样的 edges 记为 E_d.
  - control dependence edges: 对任意 statement x 和 branch condition z，若 x 控制依赖于 z，则 $(x, z) \in E$，所有这样的 edges 记为 E_c.

Definition 1.5. 给定 simple langugae 上的 program P，构造 E_d 规则如下

$$\frac{v_1 = v_2}{(v_2, v_1) \in E_d}$$

$$\frac{v_1 = v_2 \oplus v_3}{(v_2, v_1), (v_3, v_1) \in E_d}$$

$$\frac{v_1 = ite(v_2, v_3, v_4)}{(v_2, v_1), (v_3, v_1), (v_4, v_1) \in E_d}$$

$$\frac{v_1 = f(u_1, \cdots) = \{u_1 = \langle u_1 \rangle; \cdots; \textbf{return } w_1 = w_2\}}{(v_2, u_1), (w_2, w_1), (w_1, v_1) \in E_d}$$

$$\frac{v_1 = f(v_2, \cdots) \quad f(u_1, \cdots) = \emptyset}{(v_2, v_1) \in E_d}$$

$$\frac{\textbf{if } (v_1 = v_2)\{\cdots\}}{(v_2, v_1) \in E_d}$$

即对 P 上每一个 statements 都应用上述规则. 根据算法1构造 E_c.

Annotation 1.6. 算法1用到了一个 lemma: 给定 CFG 上两个结点 x, y，若 y pdom x，则在 RCFG 上有 y dom x. 直觉上若直接求每个结点 x 的 control-dependence，可以直接在 dominator tree 上向上遍历 x 的 immediate dominator，直到存在某个 dominator y 有两个及以上的 child 结点，那么 x 就控制依赖于 y. 这个直觉算法可能有冗余，但是我们可以做一点优化若在向上遍历的过程中若遇到某个 dominator 只有一个 child 结点且有控制依赖了，那么这个控制依赖就是当且要求的某个结点的控制依赖，这个优化使用了一个 dp 性质的属性; 并且我们在 dominator tree 上做深搜来遍历每个结点.

Algorithm 1: Control Depenence

input : The reverse control flow graph RCFG and the dominance frontier RDF of every every node in RCFG

output : The set CD(X) of every node X that are control dependent on X

```
1 begin
2     for each node X ∈ RCFG do
3         CD(X) = ∅
4     for each node X ∈ RCFG do
5         for each node Y ∈ RCFG do
6             Insert Y into CD(X)
```