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

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

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.

Annotation 1.2. 为了方便说明, 规定后面提到的所有 CFG 都是以 statement 为结点, 除了 branch statement **if** $(c)\{S\}$, 我们通常将其条件判断作为一个 test node **test**(c), 而其 body S 正常展开.

Definition 1.3. 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; 若当 z 可达且值为 true 时 x 会被执行, 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 1.4. 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.

注意往常我们定义 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 1.5. 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 1.6. 给定 simple langugae 上的 program P，构造 E_d 规则如下

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

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

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

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

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

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

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 1.7. 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.

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 $\mathrm{CD}(X)$ of every node X that are control dependent on X.

begin
    for each node $X \in \mathrm{RCFG}$ **do**
        $\mathrm{CD}(X) = \emptyset$
    for each node $X \in \mathrm{RCFG}$ **do**
        for each node $Y \in \mathrm{RCFG}$ **do**
            Insert Y into $\mathrm{CD}(X)$

Definition 1.8. 给定 program P 的某个 procedure 对应 CFG 上的一条 control flow path $p = (s_0, s_1, \cdots, s_n)$，设 P 的 program dependence graph 为 $G = (V, E)$. 设关注的 data facts 为 D，s_i 上 data values along p 为 $\mathrm{in}_{s_i}, \mathrm{out}_{s_i} \subseteq D$，其中除了 $\mathrm{in}_{s_0} = I \subseteq D$ 其它 data values 都为 $\emptyset$，s_i 的 transfer function 为 tr_{s_i}. 那么 $\forall s_i \in p, i = 0, 1, \cdots, n$,

$$\mathrm{out}_{s_i} = \mathrm{tr}_{s_i}(\mathrm{in}_{s_i})$$
$$\text{and } \forall (s_i, s_j) \in E_d, \mathrm{in}_{s_j} = \mathrm{in}_{s_j} \cup \mathrm{out}_{s_i}.$$

上述分析手法称为sparse analysis. 同时设对任意 $(s_i, s_j) \in E_d$ 的 data dependence edge condition 为 $\phi(s_i, s_j)$，$\mathrm{in}_{s_i}, \mathrm{out}_{s_i} \subseteq D \times \mathcal{P}$，其中 $\mathcal{P}$ 为 data dependence paths. 初始化 $\mathrm{in}_{s_0} = (I, \prod_{s_0} = \{\langle (s_0), \phi_{(s_0)} \rangle\})$ 和 $\mathrm{in}_{s_i} = (\emptyset, \prod_{s_i} = \{\}), i = 1, 2, \cdots, n$，其中 $\phi(s_i, s_i)$ 表示对应 statement 或者变量本身的约束条件. 那么 $\forall s_i \in p, i = 0, 1, \cdots, n$,

$$\mathrm{out}_{s_i} = \left(\mathrm{tr}_{s_i}(X_{s_i}), \bigcup_{p_i = (\pi_i, \phi_{\pi_i}) \in \prod_{s_i}} \{\langle \pi_i, \phi_{\pi_i} \wedge \phi_{(i,i)} \rangle\} \right)$$
$$\text{and } \forall (s_i, s_j) \in E_d, \mathrm{in}_{s_j} = \mathrm{in}_{s_j} \cup_p \mathrm{out}_{s_i}.$$

其中 $\cup_p$ 定义为

$$\mathrm{in}_{s_j} \cup_p \mathrm{out}_{s_i} = \left(X_{\mathrm{in}_{s_j}} \cup X_{\mathrm{out}_{s_i}}, \prod_{s_j} \cup \bigcup_{p_i = \langle \pi_i = (\cdots, s_i), \phi_{\pi_i} \rangle \in \prod_i} \{\langle \pi_i' = (\cdots, s_i, s_j) \rangle, \phi_{\pi_i'}\} \right).$$

则称上述手法为path-sensitive sparse analysis.

Annotation 1.9. sparse analysis 相比于"dense" analysis 或者 conventional analysis，它并不是沿着原本的 control flow 来传递 data facts，而是沿着 data dependence 来传递的. 自然地就形成了 data dependence path，我们若要考虑 path-sensitive 就直接考虑 data dependence path condition. 这里我引入了特殊的约束条件 $\phi(s_i, s_j)$ 是有必要的，因为 statement 能否正确的执行也需要一定的约束条件，i.e. $z = x/y \Rightarrow y \neq 0$. 我这里关于 path-sensitive sparse analysis 的定义和原文有一点区别，原文一个 $\prod$ 保存了所有 data dependence paths，似乎最后把它们聚合到一起来作为整体 data dependence path conditions，我有点不理解. 因此我把关系每个结点的 data dependence paths 分开放置，这样后面对某一点进一步分析的时候，我们可以只关注相关的 data dependece path.

Definition 1.10.