IFDS: Dataflow Analysis via Graph Reachability

枫聆

2022 年 1 月 12 日

目录

Definitions

Annotation 1.1. 在数据流分析中的“精确”一词，实际上等价于“meet over all vaild path”.

- 在过程内分析 (intraprocedural) 中，一条“vaild path”就是指从某个 procedure 的 CFG 上从 entry node 到特定的点这样一条路径，同时这样的路径也可以称为 same level vaild path.
- 在过程间分析 (interprocedural) 中，一条“vaild path”就是指当从 main function 开始，且某个 procedure 结束之后返回调用它的 procedure，直到某个特定程序点的这样一条路径.

上述东西没有什么新意，就是在图上分析时候，需要确定一类程序潜在可能的执行路径，而我们只分析它们. 这样 vaild path 如果更形式化一点，我们可以可以用 CFL 来描述.

Definition 1.2. 数据流分析中的可能会出现所有不同的数据值组成的集合 D(underlying set) 称为 dataflow facts. 对于可能分析得到的结果是 dataflow facts 的一个子集，通常我们把所有可能得到的结果记为 2^D.

Definition 1.3. 数据流的值可以表示成位向量 (bit-vectors)，其中每个 bit 可以表示一个具体的 dataflow fact，且可以每个传递函数可以用相应的位运算来表示，这样的一类数据流分析问题我们称之为locally separable problems. i.e. reaching-definitions, available expressions, live variables.

Annotation 1.4. 怎么理解”separable”? separable 对应的是逻辑位运算过程不同位 bit 是不会相互影响的，可以将 dataflow facts 划分成独立的 space，每个 space 里面的 dataflow facts 在 propogate 的过程是封闭的. 例如在 reaching-definitions 中两个不同变量定义的作用域是不会相互影响的.

Definition 1.5. 若 dataflow facts D 是一个有限 (finite) 集合，且每一个 transfer function $f: 2^D \to 2^D$ 都是可分配的 (distributive，即满足 join or meet semilattice homomorphism)，这样一类过程间 (interprocedural) 数据流分析问题称为interprocedural, finite, distributive, subset problems，或简称为IFDS problems.

Definition 1.6. 设程序 P 的每个 procedure p 对应图表示为 $G_p = (N_p, E_p)$，其中 N_p 表示 p 上所有 (atomic) statements，E_p 表示 p 的控制流. G_p 中结点种类分为

- 唯一的 start node s_p;
- 唯一的 exit node e_p;
- 过程调用结点 call node，G_p 中的所有 call nodes 构成的集合记为 Call_p;
- 过程调用的返回位置结点 return-site node，即紧跟在 call node 后面，G_p 中的所有 return-site nodes 构成的集合记为 Ret_p
- 其余的结点与通常 flowgraph 上结点保持一致.

特别地，G_p 上每一对 call node c 和 return-site node r 直接有一条 c 到 r 的有向边，称为call-to-return-side edge. 设 $G^* = (N^*, E^*)$，G^* 由所有 procedures 图表示 $G_1, G_2, \cdots, G_p, \cdots$ 和两类特殊的边构成

- call-to-start edge: 从 G_{p_1} 中 call node 到对应 G_{p_2} 的 start node s_p 的有向边.
- exit-to-return-side edge: 从 G_{p_2} 的 exit node e_{p_2} 到对应 G_{p_1} 的 return-side node.

称 G^* 为supergraph.

Annotation 1.7. 理解 supergraph 只需要了解几个特殊的点

- return-site 这类结点可能是抽象出来的，在一定程度上有利于对 call node 的细化分析.
- 放置 call-to-return-side edge 的目的是用于过程内分析，i.e. local variables.
- 一个 procedure 执行返回会抽象到 exit node 上统一返回，即 exit-to-return-side 的作用.

Definition 1.8. 设 $f\colon 2^D \to 2^D$ 某个 IFDS problem 中一个 distributive transfer function，它的一个关系表示(representation relation) $R_f \subseteq (D \cup \{0\}) \times (D \cup \{0\})$ 为

$$\begin{aligned} R_f = \quad & \{0, 0\} \\ & \cup \{\, (0, y) \mid y \in f(\emptyset) \,\} \\ & \cup \{\, (x, y) \mid y \in f(\{x\}) \text{ and } y \notin f(\emptyset) \,\}. \end{aligned}$$

其中 0 表示 $\emptyset$.

Definition 1.9. 定义一个 IFDS problem 的实例为 $\mathrm{IP} = (G^*, D, F, M, \sqcap)$，其中 $G^* = (N^*, E^*)$ 为对应程序 P 的 supergraph 表示，D 表示 dataflow facts，F 为 distributive transfer function set，M 为 $E^* \to F$ 的映射，$\sqcap$ 对应的 meet operator.

Annotation 1.10. 注意上述 representation 是当 f 可分配满足

$$f(X_1 \vee X_2) = f(X_1) \vee f(X_2)$$

其中 $\vee$ 为 set 上的 union 操作. 那么当 $\vee$ 为 set 上 intersection 如何定义 representation 呢? 这个问题非常有趣，此时 f 的可分配的 operator 记为 $\wedge$，文章中作者提到这样的问题可以转化为一个与当前问题等价的一个新问题，且这个新问题是 union 可分配的，i.e. 如果 “must-be-X” 是一个 intersection 可分配的问题，那么与它等价 “may-not-be-X” 问题就是一个 union 可分配的问题. 但是为什么呢? 文中没有解释，我们来做几步推导. 假设 “must-be-X” 的 f 是一个 intersection 可分配，那么 “may-not-be-X” 的 f^c 应为

$$f^c(X) = D - f(X).$$

那么

$$\begin{aligned} f^c(X \wedge Y) &= D - f(X \wedge Y) \\ &= [D - f(X)] \vee [D - f(Y)] \\ &= f^c(X) \vee f^c(X) \end{aligned}$$

真奇怪，显然没有得到想要的结果. 那么就是 $f^c(X)$ 的定义出问题了，我们仔细思考一下原问题和新问题本质没有区别，只是在每个结点的对应值用补的形式表示，两个问题同一状态相同结点的值也是互补的，于是 f^c 的正确定义应当如下:

$$f^c(X) = D - f(D - X)$$

那么

$$\begin{aligned} f^c(X \vee Y) &= D - f[D - (X \vee Y)] \\ &= D - f[(D - X) \wedge (D - Y)] \\ &= D - f(D - X) \wedge f(D - Y) \\ &= [D - f(D - X)] \vee [D - f(D - Y)] \\ &= f^c(X) \vee f^c(Y) \end{aligned}$$

这样确实可以得到一个和原问题等价的 union 可分配的问题.

Definition 1.11. 给定关系 $R \subseteq (D \cup \{0\}) \times (D \cup \{0\})$，则其唯一确定函数 $[\![R]\!]\colon 2^D \to 2^D$ 为

$$[\![R]\!](X) = \{\, y \mid \forall x \in X, (x, y) \in R \,\} \cup \{\, y \mid (0, y) \in R \,\} - \{0\}$$

Theorem 1.12. 给定 distributive transfer function f (set union)，设 f 的关系表示为 R_f，则

$$[\![R_f]\!] = f$$

证明. 上述 representation correctness 取决于 f 满足对任意的 $X = \{x_1, x_2, \cdots, x_n\} \subseteq D$ 有

$$f(X) = f(\{x_1\} \vee \{x_2\} \vee \cdots \vee \{x_n\}) = f(\{x_1\}) \vee f(\{x_2\}) \vee \cdots f(\{x_n\})$$

□

Proposition 1.13. 将 R 可以看做一个 $2(D+1)$ 结点的图 G，任意一个 pair $(x, y) \in R$ 作为 G 上一条从 x 到 y 的有向边，则 G 最多 $D^2 + D + 1$ 条边.

证明. 注意是不存在 $(x, 0) \in R$，所以不是 $(D+1)^2$(但是原文认为是 $(D+1)^2$). □

Definition 1.14. 给定两个关系 $R_1, R_2 \subseteq S \times S$，定义它们的复合关系(composition)$R_2 \circ R_1$ 为

$$R_2 \circ R_1 = \{\, (x, y) \in S \times S \mid \text{if } z \in S, (x, z) \in R_1 \text{ and } (z, y) \in R_2 \,\}.$$

Theorem 1.15. 给定两个 distributive transfer function f, g (set union)，设 f, g 的关系表示分别为 R_f, R_g，则

$$[\![R_g \circ R_f]\!] = g \circ f.$$

证明. 对任意的 $X \subseteq D$，设 $y \in [\![R_g \circ R_f]\!](X)$，我们来证明 $y \in g \circ f(X)$. 若 $(0, y) \in R_g \circ R_f$，那么存在 $(0, z) \in R_f$ 和 $(z, y) \in R_g$，显然此时 $z = 0$，则 $y \in g(\emptyset)$，于是

$$g \circ f(X) = g[\emptyset \vee f(X)] = g(\emptyset) \vee g \circ f(X).$$

因此 $y \in g \circ f(X)$; 若 $(x, y) \in R_g \circ R_f$，其中 $x \in X$，那么存在 $(x, z) \in R_f$ 和 $(z, y) \in R_g$，显然 $z \neq 0$ 且 $z \in f(x)$ 和 $y \in g(z)$，于是

$$\begin{aligned} g \circ f(X) &= g \circ f[\{x\} \vee (X - \{x\})] \\ &= g[f(x) \vee f(X - x)] \\ &= g\{z \vee [f(x) - \{z\}] \vee (X - \{x\})\} \\ &= g(z) \vee g\{[f(x) - \{z\}] \vee (X - \{x\})\} \end{aligned}$$

因此 $y \in g \circ f(X)$. 同理可证，设 $y \in g \circ f(X)$，则有 $y \in [\![R_g \circ R_f]\!](X)$. □

Corollary 1.16. 给定一组 distributive transfer function $f_1, f_2, \cdots, f_n$，设 $f_1, f_2, \cdots, f_n$ 对应的关系表示分别为 $R_1, R_2, \cdots, R_n$ 则

$$f_n \circ f_{n-1} \circ \cdots \circ f_2 \circ f_1 = [\![R_n \circ R_{n-1} \circ \cdots \circ R_2 \circ R_1]\!].$$

Definition 1.17. 给定一个 IDFS 问题实例 $\mathrm{IP} = (G^*, D, F, M, \sqcap)$. 定义exploded supergraph $G^{\otimes} = (N^{\otimes}, E^{\otimes})$ 如下

$$\begin{aligned} N^{\otimes} &= N^* \times (D \cup \{0\}) \\ E^{\otimes} &= \{\, \langle m, d_1 \rangle \to \langle n, d_2 \rangle \mid (m, n) \in E^* \text{ and } (d_1, d_2) \in R_{(m,n)} \,\} \end{aligned}$$

Annotation 1.18. exploded supergraph 就是 supergraph 的基础上，将每个结点扩展成了 $D+1$ 个结点，同时把每条边用其对应的 transfer function relations 扩展成了 $|R_f|$ 条边.

Annotation 1.19. 将 vaild path 推广至到 exploded supergraph 上，称其为 realizable path.

Theorem 1.20. 给定一个 IDFS 问题实例 $\mathrm{IP} = (G^*, D, F, M, \cup)$ 的 exploded supergraph 表示 $G^{\otimes} = (N^{\otimes}, E^{\otimes})$，则对任意的结点 $n \in N^*$，有 $d \in \mathrm{MVP}_n$(meet-over-all-path-solution) 当且仅当存在一条从 $\langle s_m, 0 \rangle$ 到 $\langle n, d \rangle$ 的 realizable path.

Annotation 1.21. 上述定理就是我们的最终目标，将类 IFDS 问题可以转换称 graph reachability 问题. 注意这个定理的 correctness 取决于当前 IDFS 是一个 set union IFDS 问题，其他细节是比较显然的.

Algorithm and Complexity

Definition 2.1. 给定一个 IDFS 问题实例 IP $= (G^*, D, F, M, \cup)$，定义任意起始结点是 $\langle s_p, 0\rangle$ 的 same level realized path 称为path edge. 定义任意 $n \in$ CallNodes，以 $\langle n, d_1\rangle$ 为起点和以 $\langle \text{ReturnSite}(n), d_2\rangle$ 为终点且其前缀路径包含 $\langle n, d_1\rangle \to \langle s_{\text{Called}(n)}, d_3\rangle$ 的 same level realizable path 称为summary path.

Algorithm 1: Tabulation Algorithm

input : A exploded supergraph $G^{\otimes}$ of the instance of IFDS problem

output : The dataflow values X_i of every node i

begin
- Let $(N^{\otimes}, E^{\otimes}) = G^{\otimes}$
- PathEdge $= \{\langle s_{main}, 0\rangle \to \langle s_{main}, 0\rangle\}$
- WorkList $= \{\langle s_{main}, 0\rangle \to \langle s_{main}, 0\rangle\}$
- SummaryEdge $= \emptyset$
- `ForwardTabulateSLRPs()`
- **for** each $n \in N^*$ **do**
  - $X_n = \{\, d_2 \in D \mid \text{if } d_1 \in D \cup \{0\} \text{ and } \langle s_{\text{profOf}(n)}, d_1\rangle \to \langle n, d_2\rangle \in \text{PathEdge} \,\}$

Procedure Propagate(e)

begin
- **if** $e \notin$ PathEdge **then**
  - Insert e into PathEdge;
  - Insert e into WorkList;

Procedure ForwardTabulateSLRPs

```
1  begin
2      while WorkList ≠ ∅ do
3          Select and remove an edge ⟨s_p, d_1⟩ → ⟨n, d_2⟩ ;
4          if n ∈ Call_p then
5              for each d_3 such that ⟨n, d_2⟩ → ⟨s_calledProc(n), d_3⟩ do
6                  Propagate(⟨s_calledProc(n), d_3⟩ → ⟨s_calledProc(n), d_3⟩) ;
7              for each d_3 such that ⟨n, d_2⟩ → ⟨returnSide(n), d_3⟩ ∈ E^⊗ ∪ SummaryEdge do
8                  Propagate(⟨s_p, d_1⟩ → ⟨returnSide(n), d_3⟩) ;
9          else if n = e_p then
10             for each c ∈ callers(p) do
11                 for each d_4, d_5 such that ⟨c, d_4⟩ → ⟨s_p, d_1⟩ and ⟨n, d_2⟩ → ⟨returnSide, d_5⟩ do
12                     if ⟨c, d_4⟩ → ⟨returnSide, d_5⟩ ∉ SummaryEdge then
13                         Insert ⟨c, d_4⟩ → ⟨returnSide(c), d_5⟩ into SummaryEdge ;
14                         for each d_3 such that ⟨s_profOf(c), d_3⟩ → ⟨c, d_4⟩ do
15                             Propagate(⟨s_profOf(c), d_3⟩ → ⟨returnSide(c), d_5⟩)
16         else
17             for each ⟨n, d_2⟩ → ⟨m, d_3⟩) ∈ E^⊗ do
18                 Propagate(⟨s_p, d_1⟩ → ⟨m, d_3⟩));
```

Annotation 2.2. 整个算法的脉络还是比较清晰的，理解定义 path edge 和 summary edge 就是为该算法的 dp 性质设计的. 一些可能比较有趣的点

- FowardTabulateSLRPs 第 10 行这里可能有一些冗余，原文这里 calllers 函数的设计是找到所有调用该 procedure 的调用者，如果我们只需要分析特定程序潜在可能的运行的路径，那么可能存在一些 callers 根本不会被运行，这些 calllers 理论上也没必要分析，所以这里的 callers 函数可以设计成一个动态函数，用于记录已经被分析过 caller.

Theorem 2.3. Tabulation algorithm 的时间复杂度为 $O(ED^3)$.

证明. Tabulation algorithm 1 的运行时间包括 ForwardTablutateSLRPs 函数的运行时间和第 7-8 行的运行时间. 其中第 7-8 行的运行时间很容易计算为 $O(ND^2)$，其中还 $N < E$ 因此 $O(ED^2)$.

ForwardTablutateSLRPs 函数 3 的运行时间，你要仔细分析是比较麻烦的，巧妙一点就是可以看做从 $D+1$ 结点开始在 $G^{\otimes}$ 上的 $D+1$ 次广搜，一次广搜的时间复杂度是和图的规模成线性关系的即 $O(ND^2+ED^2)$，还是由于 $N < E$，即有 $O(ED^2)$，因此总的时间复杂度为 $O(ED^3)$. □

参考文献

[1] Precise InterproceduralDataflow Analysis via Graph Reachability.