

数据流分析 Foundation.

1. 形式化抽象

DFA 迭代算法 $\Rightarrow F: V^k \rightarrow V^k$

终止于 V^k 之间的不动点: $X_{i+1} = F(X_i)$

PS: 关于算法机制的 3 个问题:

- ① 是否一定可以收敛到一个 DFA 的不动点?
- ② 解是否唯一, 或全局最优?
- ③ 算法的代价是否可计算?

2. 数学基础

1. 偏序集

① 定义: 非空集合 P + 其上的偏序关系 \leq

{自反性: $\forall x \in P, x \leq x$

{反对称性: $\forall x, y \in P, x \leq y \wedge y \leq x \Rightarrow x = y$

{传递性: $\forall x, y, z \in P, x \leq y \wedge y \leq z \Rightarrow x \leq z$

② 上/下界

上界: 偏序集全域 P 中的某个元素与子域 S 中的所有元素都可比, 令大于则为上界; 令小于则为下界

其中，最小的上界称为上确界(lub或join)，记作 US；最大的下界称为下确界(glb或meet)，记作 NS。

Ps: 并不是所有偏序集皆有 lub 或 glb，但如果有，则 lub 或 glb 一定唯一。

② 格

① 定义：对于偏序集 (P, \leq) ，若对于 $\forall a, b \in P$ ， $a \cup b$ 及 $a \cap b$ 均存在，则 (P, \leq) 称为格。

Ps: 格的定义实际上就是说，对偏序集中所有两两可比的元素均存在上确界与下确界，称其“封闭”。

② 半格

在偏序集 (P, \leq) 中，对 $\forall a, b \in P$ ，

若只存在 $a \cup b$ ，则称其为 join 半格；

若只存在 $a \cap b$ ，则称其为 meet 半格。

③ 完全格

在偏序集 (P, \leq) 中，对 $\forall S \subseteq P$ ，如果 US 和 NS 均存在，则其称为完全格。

Ps: 完全格的条件比格更强，其不仅规定不可比元素两两自闭，甚至要求多言自闭操作，而且对于无穷集还要若考虑了算无穷时，起无穷的元总是否存在。e.g. (N, \leq) 就不是完全格。

PPS: 完全格某种程度上描述的是“整体自闭性”对于完全格，一定存在最大元与最小元：

$$T = UP$$

$$L = \sqcap P$$

因此完全格就是针对边界所描述的整体自闭性，如果一个格是有界的，则它一定是完全格。

③ 格的积 product lattice

给定 n 个格 $L_1 = (P_1, \lesssim_1), L_2 = (P_2, \lesssim_2), \dots, L_n = (P_n, \lesssim_n)$ 若存在最小上界与最大下界，则有格的积 $L^n = (P, \lesssim)$ ：

$$\textcircled{1} \quad P = P_1 \times P_2 \times \dots \times P_n$$

$$\textcircled{2} \quad (x_1, x_2, \dots, x_n) \lesssim (y_1, y_2, \dots, y_n) \iff$$

$$(x_1 \lesssim_1 y_1) \wedge (x_2 \lesssim_2 y_2) \wedge \dots \wedge (x_n \lesssim_n y_n)$$

③ \cup 与 \sqcap 的意义同理也是对应位置进行 meet 和 join.

Ps: ① 格的极依旧是格

② 如果因子皆是完全格，则极也是完全格。

3. DFA 与格论

那么 DFA 和格论究竟有什么关系呢？我们先来回顾一下 DFA。所谓 DFA，实际上就是在程序本身与 Safe-approximation 方法论所塑造的“结构”的基础上，求解各节点的描述性数据，其中数据的域空间需要结合任务的目标进行定义，其包含某种特定的信息。

结构与数据的结合点，实际上就是所谓的“信息流”，其意味着数据的求解过程在某种程度上就是信息的流动。既然是“流动”，那么必然会有一个方向性的问题，是 forward 还是 backward 主要取决于你的任务中，数据的确主要是在依赖于前面的程序点，还是后面的程序点（Reaching Definition 主要是

前面, Live Variables 主要是后面).

我们发现“结构”是 DFA 中真正重要的东西, 它表征了关于程序本身的结构性信息, 外加一些我们为了 Safe - approximation 的结果所增加的“约束”, 其控制结构中对支流 merge 的处理。(may - must)。

扯了这么多, 我们发现 DFA 无非就是
F + ~~may/must~~ ^{+ init} 结构 + 约束 + 数据空间

+ 信息流 (backward / forward) + Boundary ^{init}

由于 约束和数据空间在本质上都取决于亦的
任务, 因此我们最终选择将这两者放在一
起进行抽象(加之 join / meet 某种程度上就
是数据间的某种操作, 所谓分支只不过 是其
结构式、非形式化的体现而已。其机制在本
质上取决于数据空间的结构, 程序只是调用这
个机制而已), 最终其抽象的结果实际上就
是一个特殊的代数系统 — 格。

基于程序结构
DFA = $F + L$ → 数据空间 / may / must
初始点是根据不动点
原理进行选择的

backward / forward + Boundary, init
Out[ENTRY] / In[EXIT]

即 DFA 实际上就是以格为代数系统的
某个含方向的函数族的迭代过程。

4. 格上函数的单调性与不动点

(1) 单调性.

格上的函数 $f: L \rightarrow L$ 是单调的, 若 $\forall x, y \in L$
皆有 $x \leq y \Rightarrow f(x) \leq f(y)$

不动点定理

给定完全格 (L, \leq) , 如果 $f: L \rightarrow L$ 单调且
格 L 有穷, 则:

- ① 最小不动点可由 $f(L), f(f(L)), \dots, f^k(L)$ 求得.
② 最大不动点可由 $f(L), f(f(L)), \dots, f^k(L)$ 求得。

证明: 完全格 L , $f: L \rightarrow L$ 单调, L 有穷

(1) 不动点存在 → must

由上和 $f: L \rightarrow L$ 的定义, $L \preceq f(L)$

由 f 单调, $f(L) \preceq f(f(L)) = f^2(L)$

由 L 是有界的, 一定存在 $f^{\text{Fix}} = f^k(L) = f^{k+1}(L)$

(2) 所求得不动点最小,

假设我们有另一个不动点 x , 即 $x = f(x)$.

是知 $L \preceq x$,

则有 $f(L) \preceq f(x)$,

假设 $f^i(L) \preceq f^i(x)$, 由 f 单调性易证

$f^{i+1}(L) \preceq f^{i+1}(x)$.

由数学归纳法, $f^k(L) \preceq f^k(x)$, $k \in \{0, 1, \dots\}$

故而不动点 f^{Fix} 一定小于等于 x , QED

而, 最小不动点唯一

类似偏序集中 lub 或 glb 若存在, 则唯一,

我们假设存在另一最小不动点 x ,

由上得 $f^{\text{Fix}}(x) \preceq f^{\text{Fix}}(L)$ 且 $f^{\text{Fix}}(L) \preceq f^{\text{Fix}}(x)$

由反对称性证毕。

5. DFA 与不动点定理的联系

我们实际上已经得到格上函数的不动点性质，现在我们需要用它来解释 DFA 相关算法的细节。首先我们需要进一步形式化 DFA 的场景：

$$DFA = (F + L + D)$$

$$F_i = f_i : L \rightarrow L \quad L^k \text{ (有穷} \rightarrow \text{完全格)} \checkmark$$

\sqcup/\sqcap

只需证 F_i 单调即可：

① f_i : Gen/kil 的元素是固定的，所以某元素活过一轮 Transfer，下一轮也一定会活下来，其永远不会缩小。

② \sqcup/\sqcap : $L \times L \times L \dots \rightarrow L$

即证对 $\forall x, y, z \in L$, $x \leq y$, 则 $x \sqcup z \leq y \sqcup z$

由 \sqcup 的定义, $y \leq y \sqcup z$,

由 $x \leq y$, 有 $x \leq y \sqcup z$

故 $y \sqcup z$ 构成 $x \sqcup z$ 的上界,

由定义, 显然, $x \sqcup z \leq y \sqcup z$

故 DFA 上的迭代算法应用不动点定理！

★ ① DFA 上的迭代算法（以 \sqcup / \sqcap 为初值
根据 D 方向在 L 上迭代）显然一定可以收敛到不动点。

② 不动点不唯一，但我们一定收敛到 greatest
或 least (至于为什么是 best, 之后论证)

③ 什么时候能到达不动点？也即迭代算法
的精度是多少。

最坏情况即为每次迭代只有一个 node 的值
在格中前进了一步，则 $i_{\max} = h \times k$

格的高度 \hookrightarrow node 数

\sqcup / \sqcap 上的最长路径长 \hookleftarrow

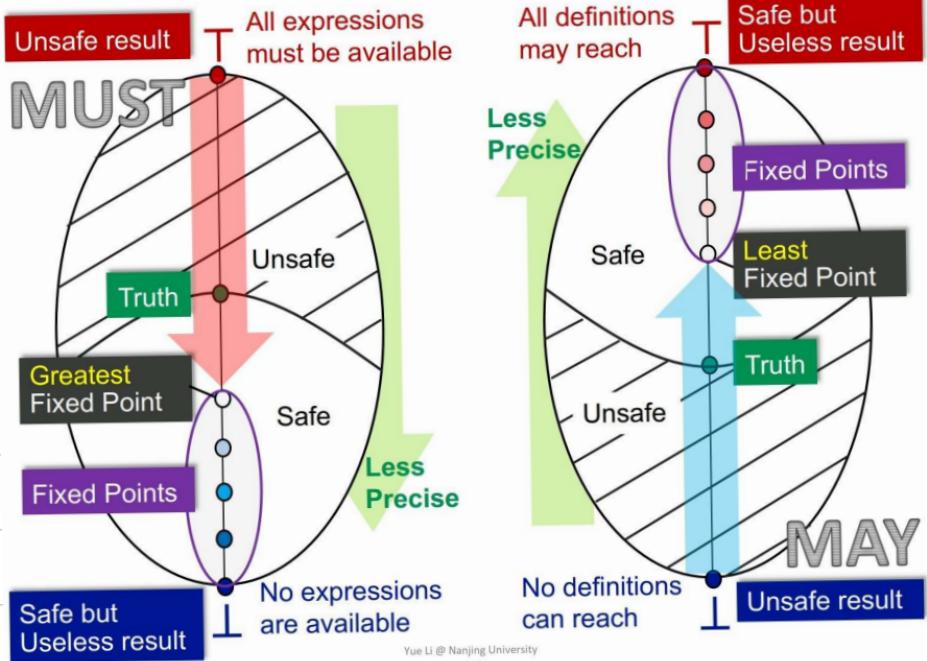
6. 格论视角下 DFA 中的 may/must 分析

初始化： $\sqcup / \sqcap \rightarrow$ unsafe 点

Merge： $\sqcup / \sqcap \rightarrow$ 保证 safe

Fixed Point：最小 / 最大 \Leftrightarrow best? ✓

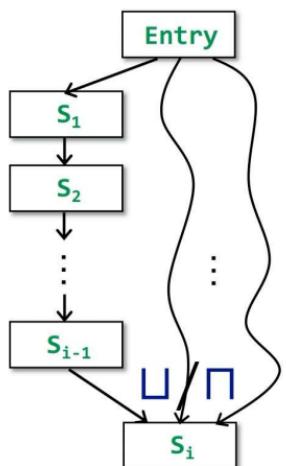
我们能做到的最好精度



Yue Li @ Nanjing University

7. 解的精度与MOP

- Meet-Over-All-Paths Solution (MOP)



$$P = \text{Entry} \rightarrow s_1 \rightarrow s_2 \rightarrow \dots \rightarrow s_i$$

Transfer function F_P for a path P (from Entry to S_i) is a composition of transfer functions for all statements on that path: $f_{S1}, f_{S2}, \dots, f_{Si-1}$

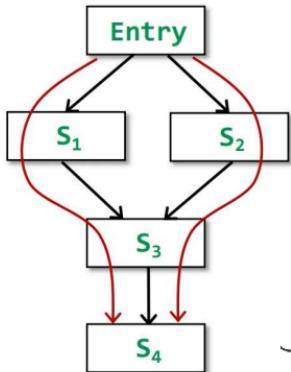
$$\text{MOP}[S_i] = \sqcup / \sqcap F_P(\text{OUT}[Entry])$$

A path P from Entry to S_i

MOP computes the data-flow values at the end of each path and apply join / meet operator to these values to find their lub / glb

Some paths may be not executable \rightarrow not fully precise
Unbounded, and not enumerable \rightarrow impractical

Ours (Iterative Algorithm) vs. MOP



$$\text{Ours} = F(x \sqcup y)$$
$$\text{MOP} = F(x) \sqcup F(y)$$

下面的问题在于 MOP 与我们的迭代算法有什么联系？

$$\text{IN}[S_4] = f_{S_3}(f_{S_1}(\text{OUT}[\text{Entry}]) \sqcup f_{S_2}(\text{OUT}[\text{Entry}]))$$

$$\text{MOP}[S_4] = f_{S_3}(f_{S_1}(\text{OUT}[\text{Entry}])) \sqcup f_{S_3}(f_{S_2}(\text{OUT}[\text{Entry}]))$$

Ps: 由于 $x \leq x \sqcup y$, $y \leq x \sqcup y$ 和 F 单调性，
易知 $\bar{F}(x) \leq \bar{F}(x \sqcup y)$, $\bar{F}(y) \leq \bar{F}(x \sqcup y)$.

若 $\bar{F}(x \sqcup y)$ 是 $F(x)$ 和 $\bar{F}(y)$ 的上界
则 $F(x) \sqcup \bar{F}(y) \leq \bar{F}(x \sqcup y)$

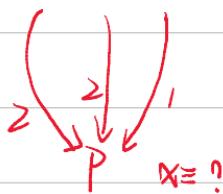
$\text{MOP} \leq \text{Ours}$

当 F 具有分配律, 若 $F(x \sqcup y) = \bar{F}(x) \sqcup \bar{F}(y)$
两者等价 $\text{MOP} = \text{Ours}$

PPS: 前面讲的 Reaching Definition, Live Variables, Available Expression 本质上都是

Gen/Kill Problem, 也即格中的 join/meet 操作都可以描述为集合的并/交，其本质上取决于由各目标所决定的数据空间的形态。

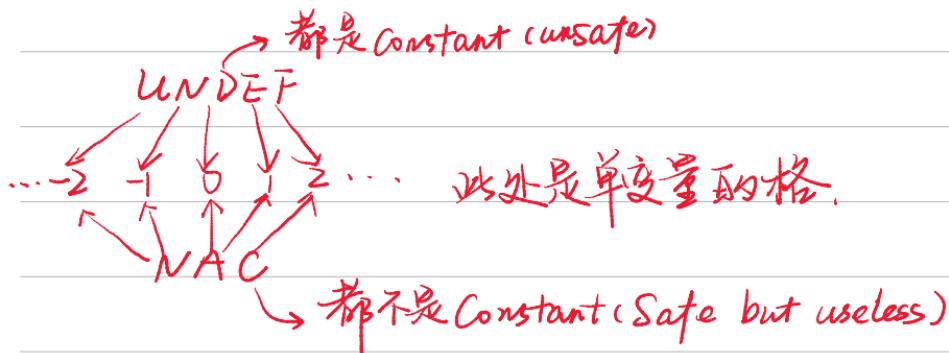
Ps: 不属于 Gen/Kill Problem 的静态分析 — 常量传播 Constant Propagation
⇒ 确定变量 x 在程序点 P 是否一定是一个常量，如果是的话，此常量是多少？



Must - Analysis

(1) L

① 数据空间 V



② Meet 操作 □

$NAC \sqcap v = NAC$

$UNDEF \sqcap v = v$

$c \sqcap v = NAC \quad (c \neq v)$

(2) F

对于语句 $s: x = \dots$, Transfer Function F 定义为

$F: OUT[s] = gen \cup (IN[s] - \{x, -\})$

① 如果出现对 x 的值分配, x 的原有值一定失效。

② 至于 x 进行值分配后的新值 gen,

• $s: x = c; // c is a constant$ $gen = \{(x, c)\}$

• $s: x = y;$ $gen = \{(x, val(y))\}$

• $s: x = y op z;$ $gen = \{(x, f(y,z))\}$

$f(y,z) = \begin{cases} val(y) op val(z) & // if val(y) and val(z) are constants \\ NAC & // if val(y) or val(z) is NAC \\ UNDEF & // otherwise \end{cases}$

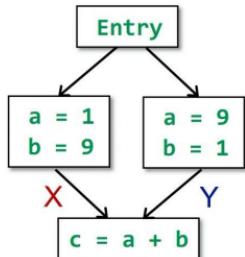
Ps: 若 s 语句不是值分配式, 则 OUT[s] 相比于 IN[s] 不发生任何变化。

(3) D

显然要进行正向分析

Ps: 关于增量传播中 F 不具有分配律的证明

Constant Propagation – Nondistributivity



$$\begin{aligned} F(X \sqcap Y) &= \{(a, \text{NAC}), (b, \text{NAC}), (\mathbf{c}, \text{NAC})\} \\ F(X) \sqcap F(Y) &= \{(a, \text{NAC}), (b, \text{NAC}), (\mathbf{c}, 10)\} \\ F(X \sqcap Y) &\neq F(X) \sqcap F(Y) \\ F(X \sqcap Y) &\sqsubseteq F(X) \sqcap F(Y) \end{aligned}$$

8. Worklist 算法

迭代算法的停止条件是所有 node 的 DUT 值都不发生改变。事实是，我们依旧遍历节点会域即使只有少数几个 node 的 DUT 值发生变化，大量的计算是无用的。

Worklist Algorithm 相当于 Iterative 算法的进化版：

Review Iterative Algorithm for May & Forward Analysis

INPUT: CFG ($kill_B$ and gen_B computed for each basic block B)

OUTPUT: $\text{IN}[B]$ and $\text{OUT}[B]$ for each basic block B

METHOD:

```
OUT[entry] = ∅;  
for (each basic block  $B \setminus \text{entry}$ )  
    OUT[B] = ∅;  
    while (changes to any OUT occur)  
        for (each basic block  $B \setminus \text{entry}$ ) {  
            IN[B] = ⋃_{P \text{ a predecessor of } B} OUT[P];  
            OUT[B] = gen_B ∪ (IN[B] - kill_B);  
        }
```

Worklist Algorithm

还需要进行处理的节点

```
OUT[entry] =  $\emptyset$ ;
for (each basic block  $B \setminus entry$ )
    OUT[B] =  $\emptyset$ ;
Worklist  $\leftarrow$  all basic blocks
while (Worklist is not empty)
    Pick a basic block  $B$  from Worklist
    old_OUT = OUT[B]
    IN[B] =  $\bigcup_{P \text{ a predecessor of } B} OUT[P]$ ;
    OUT[B] =  $gen_B \cup (IN[B] - kill_B)$ ;
    if (old_OUT  $\neq$  OUT[B])
        Add all successors of  $B$  to Worklist
```

Forward Analysis