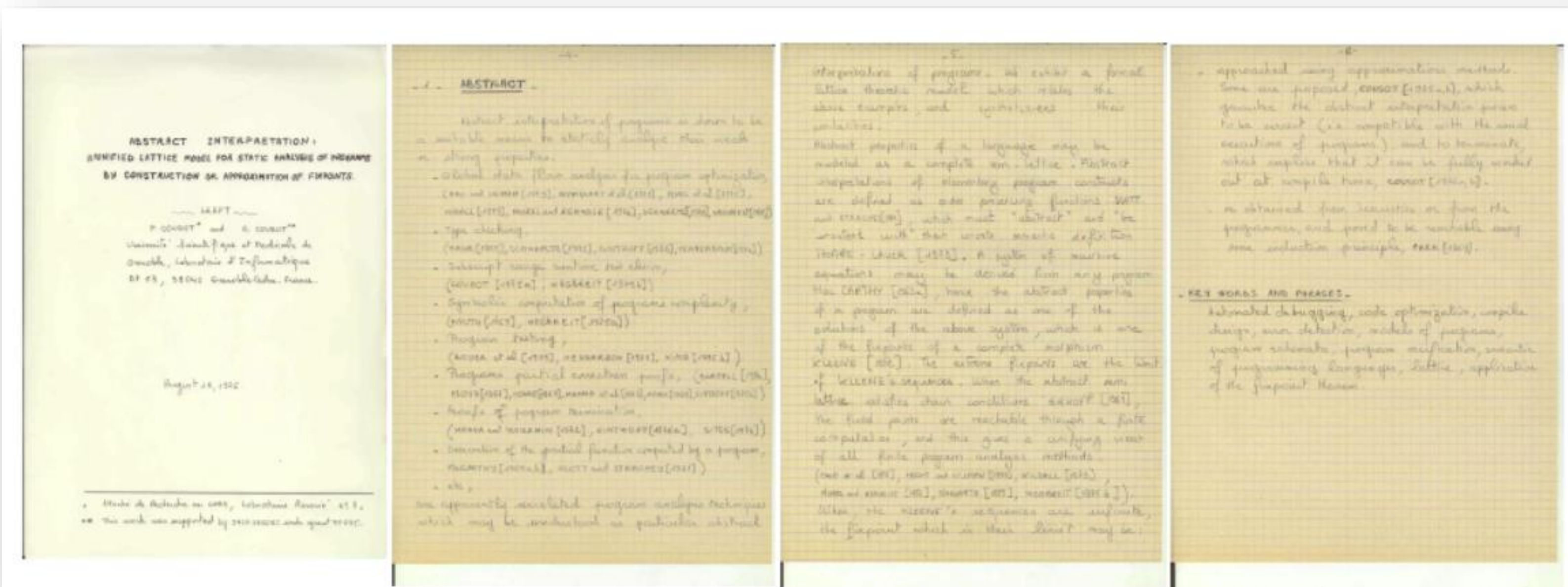
软件分析

抽象解释和分析精度

熊英飞
北京大学

抽象解释

- 最早发表于POPL’77（手写的100页论文）


ABSTRACT INTERPRETATION: A UNIFIED LATTICE MODEL FOR STATIC ANALYSIS OF PROGRAMS BY CONSTRUCTION OR APPROXIMATION OF FIXPOINTS

ABSTRACT

KEY WORDS AND PHRASES

所获荣誉

2013年ACM SIGPLAN 程序语言成就奖

Report Problem Contact Us

SIGPLAN

To explore programming language concepts and tools focusing on design, implementation and efficient use.

Home | Awards | Conferences | Resources | Membership | SIGPLAN Research Highlights | Student Information | Publications | Announcement

Programming Languages Achievement Award

Given by ACM SIGPLAN to recognize an individual or individuals who has made a significant and lasting contribution to the field of programming languages. The contribution can be a single event or a life-time of achievement. The award includes a prize of $5,000. The award is presented at SIGPLAN's PLDI conference the following June.

Nominations

- Details of the nomination and award process (pdf).
- Please use http://awards.sigplan.org/ to submit nominations.

Recipients of the Achievement Award

2013: Patrick and Radhia Cousot

Patrick and Radhia Cousot are the co-inventors of abstract interpretation, a unifying theory of sound abstraction and approximation of structures involved in various domains of computer science, such as formal semantics, specification, proof, and verification. In particular, abstract interpretation has had a major impact on the development of the static analysis of software. In their original work, the Cousots showed how to relate a static analysis to a language's standard semantics by means of a second, abstract semantics that makes precise which features of the full language are being modeled and which are being discarded (or abstracted), providing for the first time both a formal definition of and clear methodology for designing and proving the correctness of static analyses. Subsequently, the Cousots contributed many of the building blocks of abstract interpretation in use today, including chaotic iteration, widening, narrowing, combinations of abstractions, and a number of widely used abstract domains. This work has developed a remarkable set of intellectual tools and has found its way into practice in the form of widely used libraries and frameworks. Finally, the Cousots and their collaborators have contributed to demonstrating the utility of static analysis to society. They led the development of the AstrÈe static analyzer, which is used in the medical, automotive, and aerospace industry for verifying the absence of a large class of common programming errors in low-level embedded systems code. This achievement stands as one of the most substantial successes of program verification to date.

所获荣誉

2018年 约翰·冯诺依曼奖

Patrick Cousot awarded John von Neumann Medal

Patrick Cousot is the recipient of the IEEE John von Neumann medal, given "for outstanding achievements in computer-related science and technology".

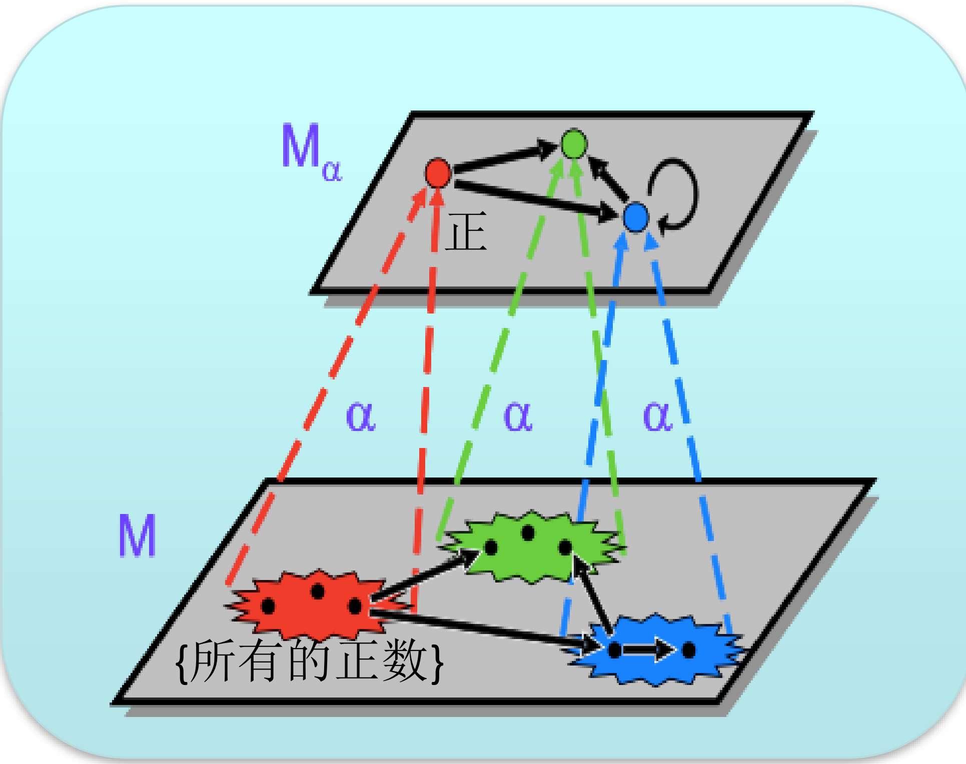
Read More

IEEE JOHN VON NEUMANN MEDAL
RECIPIENTS

2018	PATRICK COUSOT Professor, New York University, New York, New York, USA	"For introducing abstract interpretation, a powerful framework for automatically calculating program properties with broad application to verification and optimization."

抽象解释

- 主要解释抽象空间和具体空间的关系



抽象空间

具体空间

抽象解释

- 具体化函数γ将抽象值映射为具体值
  - γ(正) ={所有的正数}
  - $\gamma(\bot) = \emptyset$
  - 暂时可以把具体值想像成集合
- 抽象化函数α将具体值映射为抽象值
  - α({所有的正数})=正
  - $\alpha(\{1,2\}) =$ 正
  - $\alpha(\{-1,0\}) =$ 槑
- 假设抽象域上存在偏序关系$\sqsubseteq$

伽罗瓦连接 Galois Connection

- 我们称γ和α构成抽象域$\boldsymbol{\text{虚}}$和具体域$\boldsymbol{D}$之间的一个伽罗瓦连接，记为

$$(\boldsymbol{D}, \subseteq) \leftrightarrows_{\alpha}^{\gamma} (\boldsymbol{\text{虚}}, \sqsubseteq)$$

- 当且仅当

$$\forall X \in \boldsymbol{D}, \text{甲} \in \boldsymbol{\text{虚}}: \alpha(X) \sqsubseteq \text{甲} \Leftrightarrow X \subseteq \gamma(\text{甲})$$

定理

- $(\boldsymbol{D}, \subseteq) \leftrightarrows_{\alpha}^{\gamma} (虚, \sqsubseteq)$当且仅当以下所有公式成立

- α是单调的：$\forall X, Y \in \boldsymbol{D}: X \subseteq Y \Rightarrow \alpha(X) \sqsubseteq \alpha(Y)$
- γ是单调的：$\forall 甲, 乙 \in 虚$: $甲 \sqsubseteq 乙 \Rightarrow \gamma(甲) \subseteq \gamma(乙)$
- $\gamma \circ \alpha$保持或增大输入：$\forall X \in \boldsymbol{D}: X \subseteq \gamma\big(\alpha(X)\big)$
- $\alpha \circ \gamma$保持或缩小输入：$\forall 甲 \in 虚: \alpha\big(\gamma(甲)\big) \sqsubseteq 甲$

程序静态分析中，一般直接将$\alpha \circ \gamma$定义为等价变换

证明

- $\Rightarrow$
  - $\forall X \in \boldsymbol{D}: X \subseteq \gamma\big(\alpha(X)\big)$
    - 由$\alpha(X) \sqsubseteq \alpha(X)$和伽罗瓦连接定义可得$X \subseteq \gamma\big(\alpha(X)\big)$
  - $\forall \text{甲} \in \textbf{虚}: \alpha\big(\gamma(\text{甲})\big) \sqsubseteq \text{甲}$
    - 由$\gamma(\text{甲}) \subseteq \gamma(\text{甲})$和伽罗瓦连接定义可得$\alpha\big(\gamma(\text{甲})\big) \sqsubseteq \text{甲}$
  - $\forall X, Y \in \boldsymbol{D}: X \subseteq Y \Rightarrow \alpha(X) \sqsubseteq \alpha(Y)$
    - $X \subseteq Y \subseteq \gamma\big(\alpha(Y)\big) \Rightarrow \alpha(X) \sqsubseteq \alpha(Y)$
  - $\forall \text{甲}, \text{乙} \in \textbf{虚}: \ \text{甲} \sqsubseteq \text{乙} \Rightarrow \gamma(\text{甲}) \subseteq \gamma(\text{乙})$
    - $\alpha\big(\gamma(\text{甲})\big) \sqsubseteq \text{甲} \sqsubseteq \text{乙} \Rightarrow \gamma(\text{甲}) \subseteq \gamma(\text{乙})$

证明

- $\Leftarrow$
  - $\alpha(X) \sqsubseteq$ 甲
  - $\Rightarrow \gamma(\alpha(X)) \subseteq \gamma(甲)$ 【γ的单调性】
  - $\Rightarrow X \subseteq \gamma(甲)$ 【 $X \subseteq \gamma(\alpha(X))$ 】

  - $X \subseteq \gamma(甲)$
  - $\Rightarrow \alpha(X) \sqsubseteq \alpha(\gamma(甲))$ 【α的单调性】
  - $\Rightarrow \alpha(X) \sqsubseteq$ 甲 【$\alpha(\gamma(甲)) \sqsubseteq$ 甲】

函数抽象

- 给定伽罗瓦连接$(\boldsymbol{D}, \subseteq) \leftrightarrows_{\alpha}^{\gamma}$ (虚$, \sqsubseteq)$
- 给定$\boldsymbol{D}$上的函数f和虚上的函数子

- 子是 f 的安全抽象,当且仅当
  - $(\alpha \circ f \circ \gamma)$(甲) $\sqsubseteq$ 子(甲)
  - 即$(f \circ \gamma)$(甲) $\sqsubseteq$ $(\gamma \circ$ 子)(甲)
- 子是 f 的最佳抽象,当且仅当
  - $\alpha \circ f \circ \gamma =$ 子
- 子是 f 的精确抽象,当且仅当
  - $f \circ \gamma = \gamma \circ$ 子
- 最佳抽象总是存在，但精确抽象不一定存在

定义程序分析的安全性

- 执行踪迹（具体执行序列）：(语句编号,内存状态)构成的序列
- 程序分析：分析程序所有执行踪迹集合的属性
  - 符号分析：正常返回的执行踪迹对应变量的符号
  - 可达定值：执行踪迹集合的所有踪迹在某个节点的可达定值的并
  - 可用表达式：执行踪迹集合中所有踪迹在某个节点的可用表达式的交
  - 定义通常包括两部分：1. 单条踪迹的属性 2. 如何从单条的属性得到集合的属性
- 程序分析的安全性：在分析结果域存在某种偏序关系，与理想结果相同或者更大视为安全

定义程序分析的安全性

- 程序的执行踪迹集合和分析结果域构成伽罗瓦连接
  - 具体域：执行踪迹集合+集合子集关系
  - 抽象域：分析结果+分析结果上的偏序关系
  - α：踪迹集合对应的精确分析结果，定义为
    - $\alpha(X) = \sqcup_{x \in X} \beta(x)$
  - $\gamma(\text{甲}) = \{x \mid \beta(x) \sqsubseteq \text{甲}\}$

红色为特定分析需要定义的部分

- 容易证明上述元素形成伽罗瓦连接
  - $\alpha(X) \sqsubseteq \text{甲} \Rightarrow X \subseteq \gamma(\text{甲})$：由$\gamma$定义直接可得
  - $X \subseteq \gamma(\text{甲}) \Rightarrow \alpha(X) \sqsubseteq \text{甲}$：两边应用$\alpha$，得$\alpha(X) \sqsubseteq \alpha(\gamma(\text{甲})) = \sqcup_{\beta(x) \sqsubseteq \text{甲}} \beta(x) \sqsubseteq \text{甲}$
    - 最后一步用到下页最小上界定理。给定集合$\{\beta(x) \mid \beta(x) \sqsubseteq \text{甲}\}$，甲是该集合的上界，$\sqcup_{\beta(x) \sqsubseteq \text{甲}} \beta(x)$是最小上界。

集合的最小上界

- 上界：给定集合S，如果满足$\forall s \in S: s \sqsubseteq u$，则称u是S的一个上界
- 最小上界：设u是集合S的上界，给定任意上界u'，如果满足$u \sqsubseteq u'$，则称u是S的最小上界
- 引理：$\sqcup_{s\in S}\ s$是S的最小上界
  - 证明:
    - 根据幂等性、交换性和结合性，我们有$\forall v \in S$: $(\sqcup_{s\in S} s) \sqcup v = \sqcup_{s\in S}\ s$，所以$\sqcup_{s\in S}\ s$是S的上界
    - 给定另一个上界u，我们有$\forall s \in \mathrm{S}: s \sqcup u = u$，$(\sqcup_{s\in S} s) \sqcup u = (\sqcup_{s\in S}(s \sqcup u)) = \mathrm{u}$，所以$\sqcup_{s\in S}\ s$是最小上界

定义数据流分析的安全性

- 数据流分析在每个程序点产生一个结果，可以看做是下面多个伽罗瓦连接的复合

$$\{1 \to \{x = 正\},\ 2 \to \{x = 正\}, \cdots\}$$

$\bar{\gamma}$ ↓ ↑ $\bar{\alpha}$　　γ_2 ↓ ↑ α_2

$$\left\{1 \to \begin{Bmatrix}(1, x = 1)\\(1, x = 2)\\(1, x = 5)\\\cdots\cdots\end{Bmatrix},\ 2 \to \begin{Bmatrix}(1, x = 1), (2, x = 1)\\(1, x = 2), (2, x = 2)\\\cdots\cdots\end{Bmatrix}, \cdots\right\}$$

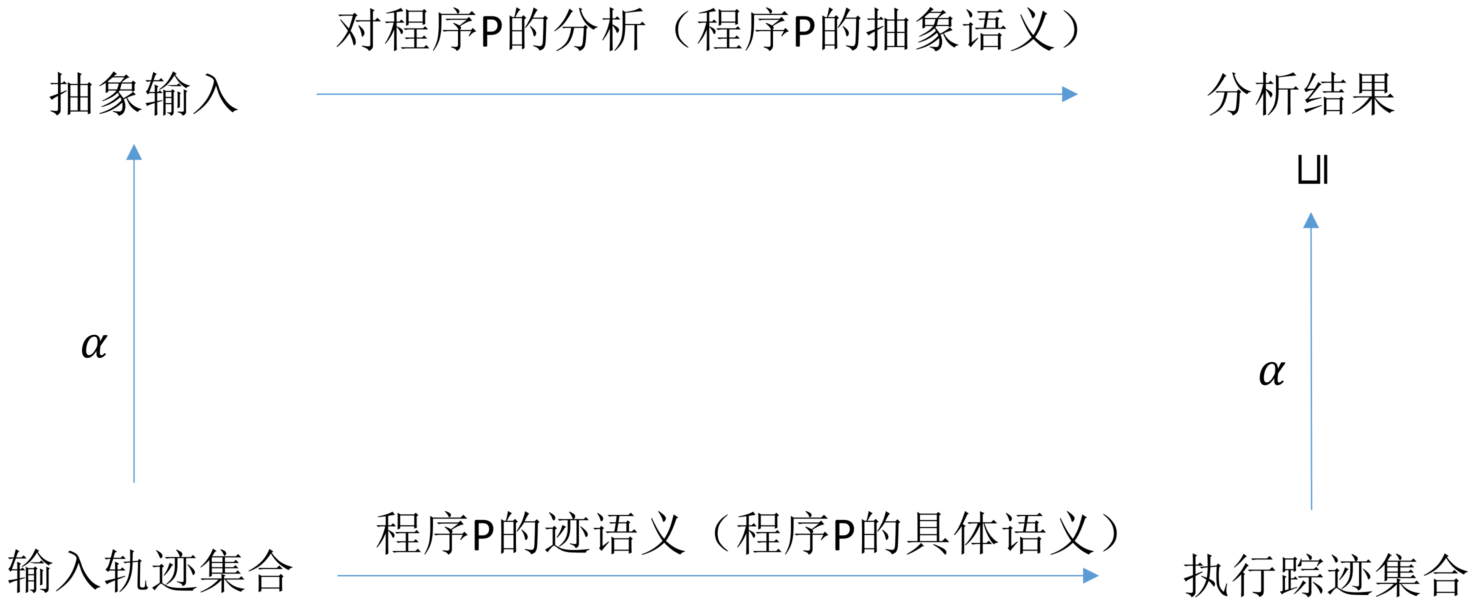
γ_1 ↓ ↑ α_1

$$\begin{Bmatrix}(1, x = 1), (2, x = 1), (5, x = 2), (6, x = 2)\\(1, x = 2), (2, x = 2), (5, x = 3), (6, x = 3)\\(1, x = 5), (3, x = 5), (4, x = 6), (5, x = 15), (6, x = 15)\\\cdots\cdots\end{Bmatrix}$$

红色为需要定义的部分

迹语义（Trace Semantics）

- 迹语义是一个函数，将程序和输入集合映射为执行踪迹的集合



也就是说，程序分析是对具体语义的一个安全函数抽象

控制流图的具体语义

- 假设每个控制流节点v对应两个函数
  - 具体（内存状态）转换函数：
    - $trans_v: M \to 2^M$
  - 控制转移函数：
    - $next_v: M \to 2^{succ(v)}$
  - 其中M为所有内存状态的集合
- 以上函数的返回值都是集合，对应不确定的执行，比如返回随机数
- 考虑反向分析的话，需要反向执行程序，程序的反向执行通常也是不确定的

控制流图的具体语义

- 定义如下单步执行函数
  - $Step(T) = \{t + (v', m') \mid$
    $t \in T$
    $v' \in next_{last(t).node}(last(t).mem),$
    $m' \in trans_{v'}(last(t).mem)\} \cup T$
  - 即每次把序列加长一步
- 那么一个程序的迹语义就是$Step^{\infty}$，定义为
  - $Step^{\infty}(T) = \lim_{n \to \infty} Step^{n}(T)$

控制流图的抽象语义

- 复习：之前定义过轮询函数
  - $\mathrm{F}\left(\mathrm{OUT}_{v_1}, \mathrm{OUT}_{v_2}, \ldots, \mathrm{OUT}_{v_n}\right) =$
    $\left(\mathrm{f}_{\mathrm{v}_1}\left(\sqcup_{\mathrm{w} \in \mathrm{pred}(\mathrm{v}_1)} \mathrm{OUT}_\mathrm{w}\right),\right.$
    $\mathrm{f}_{\mathrm{v}_2}\left(\sqcup_{\mathrm{w} \in \mathrm{pred}(\mathrm{v}_2)} \mathrm{OUT}_\mathrm{w}\right),$
    $\ldots,$
    $\left.\mathrm{f}_{\mathrm{v}_n}\left(\sqcup_{\mathrm{w} \in \mathrm{pred}(\mathrm{v}_n)} \mathrm{OUT}_\mathrm{w}\right)\right)$
- 数据流分析的结果为$F^{\infty}(I)$
- 我们现在要证明轮询函数F是Step的安全抽象

标准数据流分析的安全性

- 如果任意节点v上的抽象域转换函数f_v满足如下条件
  - $t \in Step(\bar{\gamma}(甲)) \wedge last(t).node = v$
    $$\Rightarrow t \in \bar{\gamma}\big(f_v(甲)\big)$$
- 那么F是Step的安全函数抽象，即
  - $Step\big(\gamma(甲)\big) \subseteq \gamma\big(F(甲)\big)$

- 证明：和之前证明类似，考虑顺序、分支、合并等多种不同情况，路径都仍然在转换之后的抽象值中。略

小结：
基于抽象解释设计程序分析

- 程序的具体语义：
  - 反复应用某种具体单步执行函数得到程序的踪迹集合
- 程序的抽象语义：
  - 针对问题设计抽象域
  - 设计抽象单步执行函数，是具体执行函数的安全抽象
  - 证明抽象单步执行函数收敛
    - 通常基于单调性+半格高度有限

程序分析的分类-敏感性

- 一般而言，抽象过程中考虑的信息越多，程序分析的精度就越高，但分析的速度就越慢
- 程序分析中考虑的信息通常用敏感性来表示
  - 流敏感性flow-sensitivity
  - 路径敏感性path-sensitivity
  - 上下文敏感性context-sensitivity
  - 字段敏感性field-sensitivity
- 注意区别：
  - 敏感性 vs 分析结果的形式
  - 抽象域的值可以进一步映射为想要的分析结果

术语-流敏感(flow-sensitivity)

- 流非敏感分析（flow-insensitive analysis）：如果把程序中语句随意交换位置（即：改变控制流），如果分析结果始终不变，则该分析为流非敏感分析。
- 流敏感分析（flow-sensitive analysis）：其他情况

- 数据流分析通常为流敏感的

流非敏感区间分析举例

```
If (…)
  x = 0;
  y = x;
else
  x = 1;
  y = x - 1;
```

- 求程序执行过程中x和y所有可能取值的范围
- 流敏感分析：x:[0, 1]，y:[0, 0]
- 流非敏感分析：x:[0, 1]，y:[-1, 1]

流非敏感分析

- 不区分不同节点上的OUT值，我们就得到了流非敏感分析
  - $F_{fi}(OUT) = \sqcup_{v \in V} f_v(OUT)$
- 对比流敏感分析
  - $F(\mathrm{OUT}_{v_1}, \mathrm{OUT}_{v_2}, \ldots, \mathrm{OUT}_{v_n}) =$
  $$(\mathrm{f}_{\mathrm{v}_1}(\sqcup_{\mathrm{w} \in \mathrm{pred}(\mathrm{v}_1)} \mathrm{OUT}_\mathrm{w}),$$
  $$\mathrm{f}_{\mathrm{v}_2}(\sqcup_{\mathrm{w} \in \mathrm{pred}(\mathrm{v}_2)} \mathrm{OUT}_\mathrm{w}),$$
  $$\ldots,$$
  $$\mathrm{f}_{\mathrm{v}_n}(\sqcup_{\mathrm{w} \in \mathrm{pred}(\mathrm{v}_n)} \mathrm{OUT}_\mathrm{w}))$$
- 可以定义流非敏感结果和流敏感结果之间的伽罗瓦连接。容易看出，F_{fi}是F的安全抽象。

流敏感vs流非敏感

流敏感

$\{1 \to \{x = 正\},\ 2 \to \{x = 正\}, \cdots\}$

$\bar{\gamma}$ $\bar{\alpha}$ γ_2 α_2

$$\left\{ 1 \to \begin{Bmatrix} (1, x = 1) \\ (1, x = 2) \\ (1, x = 5) \\ \cdots\cdots \end{Bmatrix},\ 2 \to \begin{Bmatrix} (1, x = 1), (2, x = 1) \\ (1, x = 2), (2, x = 2) \\ \cdots\cdots \end{Bmatrix}, \cdots \right\}$$

γ_1 α_1

$$\begin{Bmatrix} (1, x = 1), (2, x = 1), (5, x = 2), (6, x = 2) \\ (1, x = 2), (2, x = 2), (5, x = 3), (6, x = 3) \\ (1, x = 5), (3, x = 5), (4, x = 6), (5, x = 15), (6, x = 15) \\ \cdots\cdots \end{Bmatrix}$$

流非敏感

$\{x = 正\}$

$\bar{\gamma}$ $\bar{\alpha}$

流非敏感分析

- 实际中的流非敏感分析通常针对分析进行适当化简

```
a=100;
if(a>0)
  a=a+1;
b=a+1;
```

流非敏感符号分析

$$F(a,b) = (a \sqcup 正 \sqcup a + 正, b \sqcup a + 正)$$

按变量组织转换函数

流非敏感活跃变量分析

$$OUT = OUT \cup \{a\}$$

如果某节点的KILL中的变量在任意节点的GEN中，则该变量永远不会被删除，如果不在任意节点的GEN中，则该变量永远不会被添加。所以可以直接忽略KILL。

时间空间复杂度

- 活跃变量分析：语句数为n，程序中变量个数为m，使用bitvector表示集合
- 流非敏感的活跃变量分析：每条语句对应一个并集操作，时间为O(m)，迭代一轮即收敛，因此时间复杂度上界为O(nm)，空间复杂度上界为O(m)
- 流敏感的活跃变量分析：格的高度为O(m)，即每个结点的值最多变化O(m)次。每个结点有最多O(n)个后继节点，即每个结点的值最多被更新O(mn)次。每次有后继结点变化可以只合并变化的结点，因此单个均摊之后结点总更新复杂度$O(nm^2)$，总时间复杂度上界$O(n^2m^2)$，空间复杂度上界为O(nm)

- 对于特定分析，流非敏感分析能到达很快的处理速度和可接受的精度（如基于SSA的指针分析）

路径敏感性

- 路径非敏感分析：假设所有分支都可达，忽略分支循环语句中的条件
- 路径敏感分析：考虑程序中的路径可行性，尽量只分析可能的路径

- 带条件压缩函数的分析就是路径敏感分析

参考资料

- 《编译原理》第9章
- Lecture Notes on Static Analysis
  - https://cs.au.dk/~amoeller/spa/
- A Gentle Introduction to Abstract Interpretation
  - Patrick Cousot
  - TASE 2015 Keynote speech
- 抽象解释及其在静态分析中的应用
  - 陈立前
  - SWU-RISE Computer Science Tutorial