软件分析与架构设计

收集语义和指针分析

何冬杰
重庆大学

三地址码（Three-Address）

❑Review SIMP Syntax

$$
\begin{array}{lcl@{\qquad}lcl@{\qquad}lcl@{\qquad}lcl}
S & ::= & x := a & b & ::= & \text{true} & a & ::= & x & op_b & ::= & \text{and} \mid \text{or} \\
 & \mid & \text{skip} & & \mid & \text{false} & & \mid & n & op_r & ::= & < \mid \leqslant \mid = \\
 & \mid & S_1;\ S_2 & & \mid & \text{not } b & & \mid & a_1\ op_a\ a_2 & & \mid & > \mid \geqslant \\
 & \mid & \text{if } b \text{ then } S_1 \text{ else } S_2 & & \mid & b_1\ op_b\ b_2 & & & & op_a & ::= & + \mid - \mid * \mid / \\
 & \mid & \text{while } b \text{ do } S & & \mid & a_1\ op_r\ a_2 & & & & & &
\end{array}
$$

❑三地址码：一种编译器常用中间表示（IR）形式

- ➢3-address Syntax：
- ➢解决AST难以追踪程序执行时的数据流和控制流问题

$\textbf{\textit{Inst}} ::= x \coloneqq n \mid x \coloneqq y \mid x \coloneqq y\ op\ z \mid \text{goto } n \mid \text{if } x \text{ op}_\text{r}\ 0 \text{ goto } n$

$\text{op}_\text{a} ::= + \mid - \mid * \mid / \mid \ldots$

$\text{op}_\text{r} ::= < \mid \leq \mid = \mid > \mid \geq \mid \ldots$

$\text{P} \in \textbf{\textit{Num}} \rightarrow \textbf{\textit{Inst}}$

Inst: instructions
Num: number literals
x, y ∈ ***Var***: variables

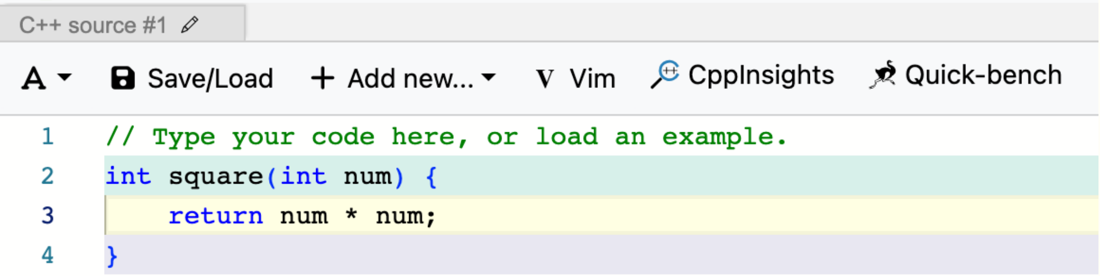
更多3地址指令，后续会用

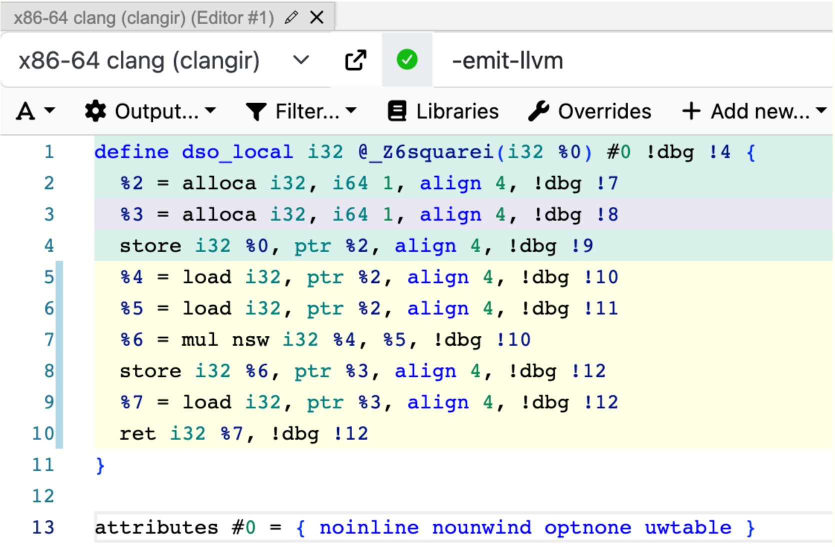
$\textbf{\textit{Inst}} ::= \cdots \mid x \coloneqq f(y) \mid \text{return } x \mid x \coloneqq y.m(z) \mid \text{read } x \mid \text{print } x \mid x \coloneqq \&p \mid x \coloneqq *p \mid *p \coloneqq x \mid \text{x} \coloneqq \text{y.f} \mid \text{x.f} \coloneqq \text{y} \mid \text{halt}$

三地址码（Three-Address）

w = x * y + z	if b then S1 else S2	while b do S
3地址码 1. t = x * y 2. w = t + z	3地址码 1. if b goto 4 2. S2 3. goto 5 4. S1 5. ...	3地址码 1. if !b then goto 3 2. S 3. ...

- ❑**AST到3地址的转换是直接的、简单的**
  - ➢Compiler Explorer：https://godbolt.org
  - ➢ (可选)作业：clang如何将AST转成LLVM-IR?





An Abstract Machine of 3-Address IR

- **Configuration (state):**
  - Environment
  - Program counter

    $c \in E \times \mathbb{N}$
- **Program:**
  - Maps labels to instructions

    $\mathrm{P} \in \mathbb{N} \rightarrow \boldsymbol{Inst}$
- **Transition or Execution:**
  - Small-step Semantics

$$P \vdash \langle E, n \rangle \rightsquigarrow \langle E', n' \rangle$$

$\boldsymbol{Inst} ::= \; x \coloneqq n \mid x \coloneqq y \mid x \coloneqq y \; op \; z \mid \text{goto } n \mid \text{if } x \text{ op}_\text{r} \; 0 \text{ goto } n$

$$\frac{P(n) = x := m}{P \vdash \langle E, n \rangle \rightsquigarrow \langle E[x \mapsto m], n+1 \rangle} \; \textit{step-const}$$

$$\frac{P[n] = x := y}{P \vdash \langle E, n \rangle \rightsquigarrow \langle E[x \mapsto E(y)], n+1 \rangle} \; \textit{step-copy}$$

$$\frac{P(n) = x := y \; op \; z \quad E(y) \; \mathbf{op} \; E(z) = m}{P \vdash \langle E, n \rangle \rightsquigarrow \langle E[x \mapsto m], n+1 \rangle} \; \textit{step-arith}$$

$$\frac{P(n) = \text{goto } m}{P \vdash \langle E, n \rangle \rightsquigarrow \langle E, m \rangle} \; \textit{step-goto}$$

$$\frac{P(n) = \text{if } x \; op_r \; 0 \text{ goto } m \quad E(x) \; \mathbf{op_r} \; 0 = \textit{true}}{P \vdash \langle E, n \rangle \rightsquigarrow \langle E, m \rangle} \; \textit{step-iftrue}$$

$$\frac{P(n) = \text{if } x \; op_r \; 0 \text{ goto } m \quad E(x) \; \mathbf{op_r} \; 0 = \textit{false}}{P \vdash \langle E, n \rangle \rightsquigarrow \langle E, n+1 \rangle} \; \textit{step-iffalse}$$

收集语义（Collecting Semantics）

The **collecting semantics** is used to define a collection of all the states (defined by operational semantics) of the program at a given point.

❑零分析 (zero analysis):

- Determine whether the value of a variable is 0
- Collecting semantics: $\mathbb{E} = \{\mathrm{E} \mid \langle \mathrm{E}, \mathrm{n} \rangle \text{ is a state}\}$
- Abstract function: $\alpha_{\mathrm{ZA}}(\mathrm{x}) = \{\mathrm{i} \mid \exists \mathrm{E} \in \mathbb{E}, \text{such that } \mathrm{E}(\mathrm{x}) = \mathrm{i}\}$
- Given a variable x : if $\alpha_{\mathrm{ZA}}(\mathrm{x}) = \{0\}$, the value of x is zero

`1. x = 10`	$x \longmapsto \{10\}$
`2. y = 2 * x`	$\mathrm{x} \longmapsto \{10\}, \mathrm{y} \longmapsto \{20\}$
`3. z = 0`	$\mathrm{x} \longmapsto \{10\}, \mathrm{y} \longmapsto \{20\}, \mathrm{z} \longmapsto \{0\}$
`4. if y > 0 goto 6`	$\mathrm{x} \longmapsto \{10\}, \mathrm{y} \longmapsto \{20\}, \mathrm{z} \longmapsto \{0\}$
`5. z = 2 * x - y`	$\mathrm{x} \longmapsto \{10\}, \mathrm{y} \longmapsto \{20\}, \mathrm{z} \longmapsto \{0\}$
`6. ...`	$\mathrm{x} \longmapsto \{10\}, \mathrm{y} \longmapsto \{20\}, \mathrm{z} \longmapsto \{0\}$

收集语义（Collecting Semantics）

The **collecting semantics** is used to define a collection of all the states (defined by operational semantics) of the program at a given point.

❑达到定值 (reaching definitions)

- Extending environment E (called E_{RD}): $E_{RD} \in Var \rightarrow \mathbb{Z} \times \mathbb{N}$
- $\mathrm{x} \mapsto \mathrm{v}, \mathrm{n}$: indicating that x was last defined as v at the location n

Collecting Semantics for Reaching definitions

New rules

$$\frac{P[n] = x := m}{P \vdash E, n \rightsquigarrow E[x \mapsto m, n], n+1}\ \textit{step-const}$$

$$\frac{P[n] = x := y}{P \vdash E, n \rightsquigarrow E[x \mapsto E[y], n], n+1}\ \textit{step-copy}$$

$$\frac{P[n] = x := y\ op\ z \quad E[y]\ \mathbf{op}\ E[z] = m}{P \vdash E, n \rightsquigarrow E[x \mapsto m, n], n+1}\ \textit{step-arith}$$

$$\frac{P(n) = \text{goto } m}{P \vdash \langle E, n\rangle \rightsquigarrow \langle E, m\rangle}\ \textit{step-goto}$$

$$\frac{P(n) = \text{if } x\ op_r\ 0 \text{ goto } m \quad E(x)\ \mathbf{op_r}\ 0 = true}{P \vdash \langle E, n\rangle \rightsquigarrow \langle E, m\rangle}\ \textit{step-iftrue}$$

$$\frac{P(n) = \text{if } x\ op_r\ 0 \text{ goto } m \quad E(x)\ \mathbf{op_r}\ 0 = false}{P \vdash \langle E, n\rangle \rightsquigarrow \langle E, n+1\rangle}\ \textit{step-iffalse}$$

```
1. x = 3
2. t = x - 2
3. if t > 0 goto 5
4. x = 4
5. y = x + 5
```

$\alpha(E, 4) = \{2,4\}$

$\alpha(E, 5) = \{1,2,4,5\}$

- Abstract function: $\alpha_{RD}(E_{RD}, n) = \{m \mid \exists x \in domain(E_{RD}) \text{ such that } E_{RD}(x) = i, m\}$

m处定义的变量可到达节点n

收集语义（Collecting Semantics）

The **collecting semantics** requires us to know each execution of the program, assuming a (possibly infinite) trace for each run.

❑Infinite Domain value

$$x \longmapsto \{-\infty, \cdots, -2, -1, 0, 1, 2, \cdots + \infty\}$$

```
1. Read x
2. y = 2 * x
3. z = 0
4. if y > 0 goto 6
5. z = 2 * x - y
6. ...
```

❑Infinite execution path

```
1. read x
2. t = 0
3. if x < 0 goto 7
4. x = x - 1
5. t = t + x
6. goto 3
7. print t
```

循环次数取决于输入x

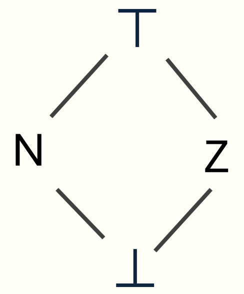
❑ 需要对数据域和控制流等进行抽象

抽象域（Abstraction）

❑ 零分析（Zero Analysis）

➢具体域：$\mathbb{Z}:\{-\infty,\cdots,-2,-1,0,1,2,\cdots+\infty\}$

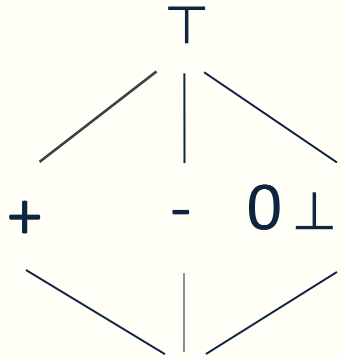
➢抽象域：$\{\perp, N, Z, \top\}$



❑符号分析 (Sign Analysis)

➢具体域：$\mathbb{Z}:\{-\infty,\cdots,-2,-1,0,1,2,\cdots+\infty\}$

➢抽象域：$\{\perp, +, -, 0, \top\}$



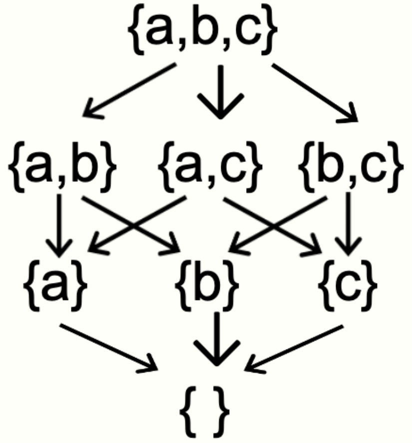
❑指针分析（Pointer Analysis）

➢具体域：运行时堆对象或栈上位置

➢抽象域：&t, alloca, 或new A对应行号表示

- Allocation site abstraction

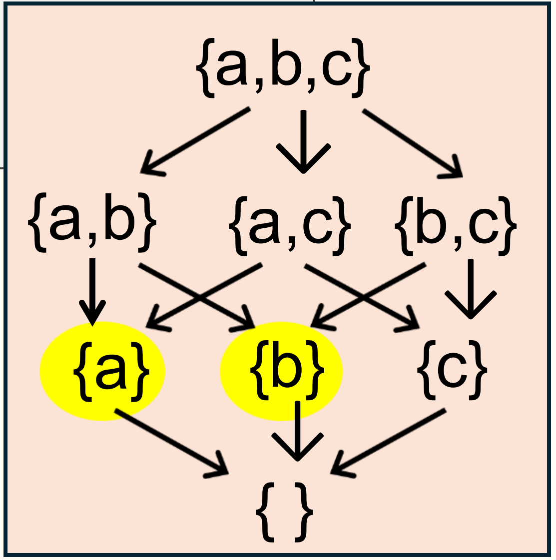
❑抽象域元素往往构成格（lattice）



偏序（Partial orders）

Given a set S, a partial order $\sqsubseteq$ is a binary relation on S that satisfies:

- reflexivity: $\forall x \in S: x \sqsubseteq x$
- transitivity: $\forall x, y, z \in S: x \sqsubseteq y \land y \sqsubseteq z \rightarrow x \sqsubseteq z$
- anti-symmetry: $\forall x, y \in S: x \sqsubseteq y \land y \sqsubseteq x \rightarrow x = y$



❑ $(S, \sqsubseteq)$称为偏序集（poset）

- ➢一个集合的幂集和⊆（subset）构成一个偏序集
- ➢偏序集中的两个元素可能无法比较,e.g., 右图中的{a}和{b}

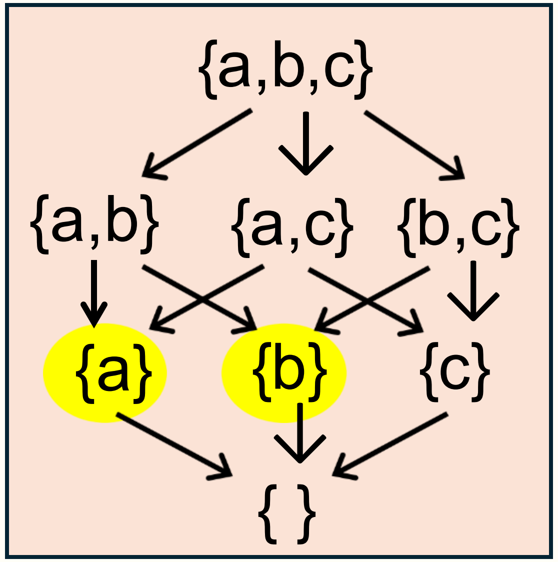
❑ Upper and Lower bounds

Let $X \subseteq S$ be a subset, we say that $y \in S$ is an **upper bound** $(X \sqsubseteq y)$ when $\forall x \in X: x \sqsubseteq y$;
We say that $y \in S$ is a **lower bound** $(y \sqsubseteq X)$ when $\forall x \in X: y \sqsubseteq x$;
A ***least upper bound*** $\sqcup X$ is defined by $X \sqsubseteq \sqcup X \ \land \ \forall y \in S: X \sqsubseteq y \Rightarrow \sqcup X \sqsubseteq y$;
A ***greatest lower bound*** $\sqcap X$ is defined by $\sqcap X \sqsubseteq X \ \land \ \forall y \in S: y \sqsubseteq X \Rightarrow y \sqsubseteq \sqcap X$.

格（Lattices）

❑格与半格

- Given a poset (S, $\sqsubseteq$), $\forall$a, b $\in$ S,
- if $a \sqcup b$ exists, then (S, $\sqsubseteq$) is called a join semilattice;
- if $a \sqcap b$ exists, then (S, $\sqsubseteq$) is called a meet semilattice;
- if both exist, then (S, $\sqsubseteq$) is called a lattice.

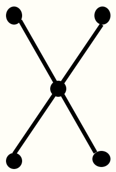


完全格

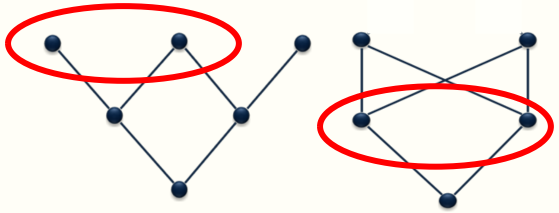
❑完全格

- If $\forall$ X $\subseteq$ S, $\sqcap$X and $\sqcup$X exist, (S, $\sqsubseteq$) is called a complete lattice.
- A complete lattice must have a unique largest element $\top = \sqcup S$ and a unique smallest element $\bot = \sqcap S$
- A finite lattice is complete if $\top$ and $\bot$ exist （一般有限格都是完全格）

❑举例:

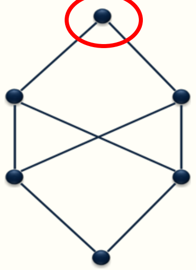


非格

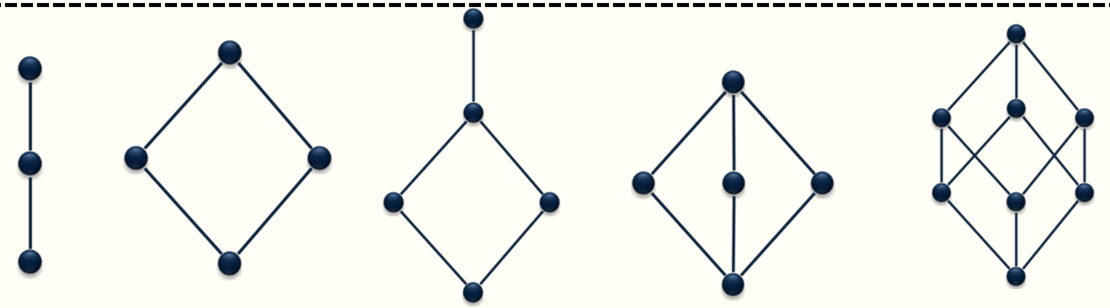


半格

不属于S



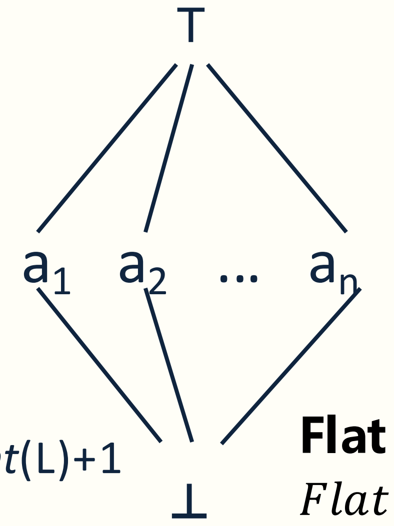
格

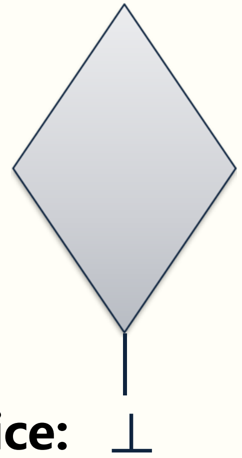


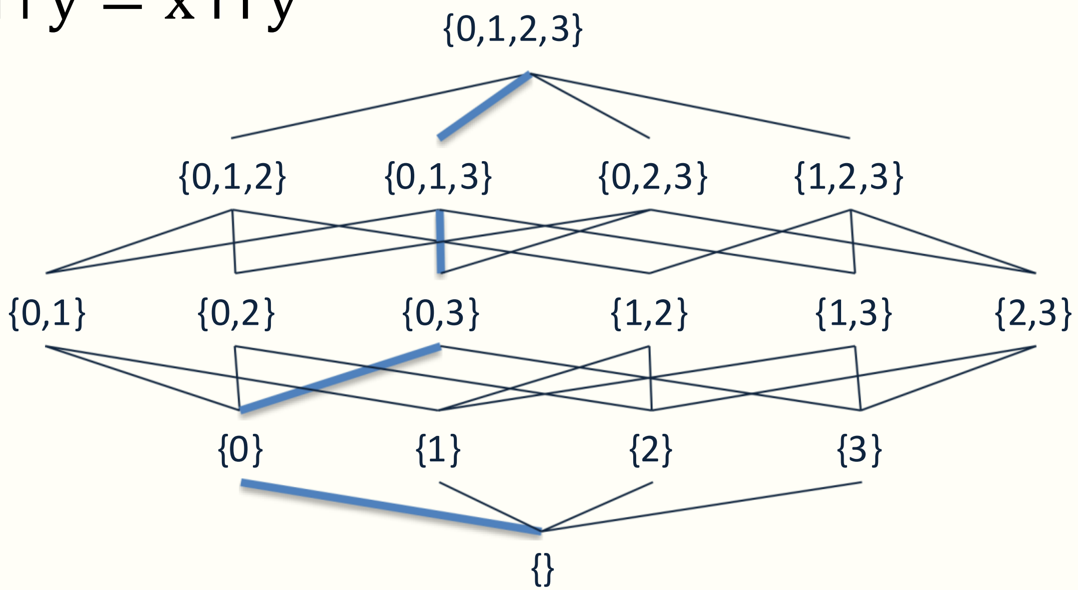
完全格

格（Lattices）

- ❑**height：the length of the longest path in the lattice**
  - ➢For complete lattice: the length from ⊥ to ⊤
- ❑**Powerset lattice:**
  - ➢Powerset of every finite set A defines a complete lattice
  - ➢$(\mathcal{P}(\mathrm{A}), \subseteq), \bot = \emptyset, \top = \mathrm{A}, \mathrm{x} \sqcup \mathrm{y} = \mathrm{x} \cup \mathrm{y}, \mathrm{x} \sqcap \mathrm{y} = \mathrm{x} \cap \mathrm{y}$
- ❑ **Flat lattice and Lift lattice**







for A = {0,1,2,3}

Lift lattice:
height(*lift*(L)) = *height*(L)+1

Flat lattice:
$Flat(A)$, where A is a set, $height\big(flat(A)\big) = 2$

格（Lattices)

- **Product lattice:**
  - $L_1 \times L_2 \times \cdots \times L_n = \{(x_1, x_2, \cdots, x_n) \mid x_i \in L_i\}$
  - $L_i (1 \leq \mathrm{i} \leq \mathrm{n})$ is a complete lattice
  - $\sqsubseteq, \sqcup, and\ \ \sqcap\ \ are\ computed\ pointwise$
  - $\mathrm{Height}(L_1 \times L_2 \times \cdots \times L_n) = \sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{Height}(L_i)$
- **Map Lattice:**
  - $\boldsymbol{A} \rightarrow \boldsymbol{L} = \{[\boldsymbol{a_1} \mapsto \boldsymbol{x_1}, \boldsymbol{a_2} \mapsto \boldsymbol{x_2}, \cdots] \mid \boldsymbol{a_i} \in \boldsymbol{A} \wedge \boldsymbol{x_i} \in \boldsymbol{L}\}$
  - $\sqsubseteq, \sqcup, and\ \ \sqcap\ \ are\ computed\ pointwise$
  - $\mathrm{Height}(\mathrm{A} \rightarrow \mathrm{L}) = |\mathrm{A}| \cdot \mathrm{Height}(\mathrm{L})$
- **例子：指针分析**
  - $\boldsymbol{A}$ is the set of program variables
  - $\boldsymbol{L}$ is the powerset of allocation site abstractions (objects)

指针分析（Pointer Analysis）

- **指针分析计算指向关系：** $\boldsymbol{Pts}$: Var $\rightarrow$ $\mathcal{P}$(OBJs)
  - ➢ $Pts(x)$: 确定指针变量x运行时的可能取值($may - points - to$)
  - ➢ **Var**：程序中的指针变量；**OBJs**：allocation site抽象，以表示从同一个分配点创建的所有运行时对象
  - ➢ *Must-points-to*关系不在考虑范围内
  - ➢ 指向关系可以用来计算别名(alias)关系
    - Alias(x, y) if and only if Pts(x) $\cap$ Pts(y) $\neq \emptyset$
- **限定需要分析的3地址IR指令：**
  - ➢ 暂时只支持流不敏感（flow-insensitive）分析，不考虑goto语句
  - ➢ 暂时只涉及过程内（intra-procedural）分析，不考虑call语句

$$\boldsymbol{Inst} ::= x := y \mid x := \&p \mid x := *p \mid *p := x \mid x := y.f \mid x.f := y \mid p := q \mid x := malloc()$$

指针分析（Pointer Analysis）

❑例子：

代码	Strong Update	Weak Update
`1. q := malloc() // O1`	Pts(q) = {O1}	Pts(q) = {O1}
`2. p := malloc() // O2`	Pts(p) = {O2}	Pts(p) = {O2}
`3. p := q`	Pts(p) = {O1}	Pts(p) = {O1, O2}
`4. r := &p`	Pts(r) = {&p}	Pts(r) = {&p}
`5. s := malloc() // O3`	Pts(s) = {O3}	Pts(s) = {O3}
`6. *r := s`	Pts(p) = {O3}	Pts(p) = {O1, O2, O3}
`7. t := &s`	Pts(t) = {&s}	Pts(t) = {&s}
`8. u := *t`	Pts(u) = {O3}	Pts(u) = {O3}

流不敏感分析不区分指令执行顺序，只有弱更新，没有强更新

问题：如何设计一个指向分析算法？

Andersen's Analysis

- **Key idea: cast as a constraint-solving problem**
  - One subset constraint per instruction
  - Domain abstraction and function abstraction

$$\frac{}{[\![n\colon p := malloc()]\!] \hookrightarrow l_n \in p}\ malloc$$

$$\frac{}{[\![p := \&x]\!] \hookrightarrow l_x \in p}\ address\text{-}of$$

$$\frac{}{[\![p := q]\!] \hookrightarrow p \supseteq q}\ copy$$

$$\frac{}{[\![*p := q]\!] \hookrightarrow *p \supseteq q}\ assign$$

$$\frac{}{[\![p := *q]\!] \hookrightarrow p \supseteq *q}\ dereference$$

- The constraint solver can give the most precise solution.
  - Why? (optional)
  - Constraints are equivalent to abstract functions

$$e \in v \equiv Pts(v) = Pts(v) \cup \{e\}$$
$$\mathrm{x} \supseteq \mathrm{y} \equiv \mathrm{Pts(x)} = \mathrm{Pts(x)} \cup \mathrm{Pts(y)}$$

  - Abstract function is monotone
  - Abstract domain is finite
    - Objects are finite
  - $(\mathrm{Var} \to \mathcal{P}(\mathrm{Objects}))$ forms a map lattice
  - Abstract function work on this lattice from $\perp$ will reach the least fixed point (e.g., the most precise solution)

Kleene's fixed-point theorem

❑Monotonicity

A function f: $L \to L$ (L is a lattice) is monotonic if $\forall x, y \in L$,

$$x \sqsubseteq y \to f(x) \sqsubseteq f(y)$$

❑Fixed point

$x \in L$ is a fixed point of function $f: L \to L$ iff $f(x) = x$

❑Kleene's fixed-point theorem

In a complete lattice with finite height, every monotone function f has a *unique least fixed-point*:

$$lfp(f) = \bigsqcup_{i \geq 0} f^i(\perp)$$

Proof of existence and uniqueness

❑Existence

- Clearly, $\perp \sqsubseteq f(\perp)$
- Since f is monotone, we also have $f(\perp) \sqsubseteq f^2(\perp)$; By induction, $f^i(\perp) \sqsubseteq f^{i+1}(\perp)$
- This means that $\perp \sqsubseteq f(\perp) \sqsubseteq f^2(\perp) \sqsubseteq \ldots f^i(\perp)$ … is an increasing chain
- L has finite height, so for some k: $f^k(\perp) = f^{k+1}(\perp)$
- If $x \sqsubseteq y$ then $x \cup y = y$, So $lfp(f) = f^k(\perp)$

❑ Uniqueness

- ➢ Assume that x is another fixed-point: $x = f(x)$
- ➢ Clearly, $\perp \sqsubseteq x$
- ➢ By induction and monotonicity, $f^i(\perp) \sqsubseteq f^i(x) = x$
- ➢ In particular, $lfp(f) = f^k(\perp) \sqsubseteq x$, i.e. $lfp(f)$ is least
- ➢ Uniqueness then follows from anti-symmetry

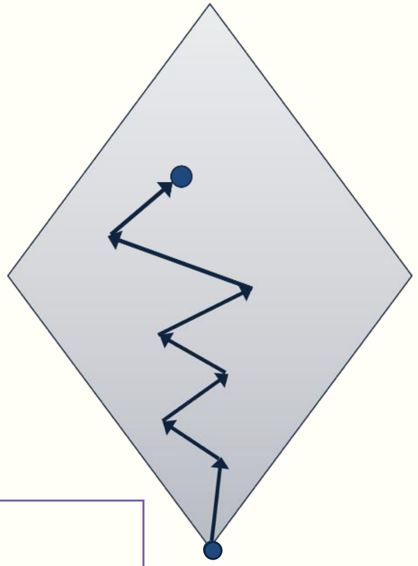
Andersen's Analysis

Let $f_1, f_2\colon L \to L$ (L is a lattice) be two monotonic functions, then their **composition** $f_1 \circ f_2$ is also **monotonic**.

Let $f\colon L \to L$ ($L = (\mathrm{Var} \to \mathcal{P}(\mathrm{OBJs}))$ is a map lattice) be the constraint solver of Andersen-style pointer analysis, obviously, f is the composition of many constraint-equivalent abstract functions, by Kleene's fixed-point theorem, the solution of Ansersen's analysis is $\mathrm{lfp}(f)$.

- **The time complexity of *lfp*(f) depends on:**
  - the height of the lattice
  - the cost of computing f
  - the cost of testing equality

```
x = ⊥;
do {
    t = x;
    x = f(x);
} while (x≠t);
```



Implementation: TIP/src/tip/solvers/FixpointSolvers.scala

A more efficient implementation

- **The Cubic Framework ($O(n^3)$)**
  - A set of tokens $\mathrm{T} = \{\mathrm{t_1}, \mathrm{t_2}, \cdots, \mathrm{t_k}\}$,
  - A collection of constraint variables $\mathrm{V} = \{\mathrm{x_1}, \cdots, \mathrm{x_n}\}$
  - A collection of constraints of these forms:

    - $t \in x$;
    - $x \subseteq y$; inclusion constraints
    - $t \in x \Longrightarrow \mathrm{y} \subseteq \mathrm{z}$; conditional constraints

- **Solution: reachability on a constraint graph**
  - Each variable is mapped to a node
  - Each node has a bitvector in $\{0,1\}^k$, initially set to all 0's
  - Each bit has a list of pairs of variables, modeling conditional constraints
  - The edges model inclusion constraints

求解过程：传播+约束维护

- ❑ $x.sol \subseteq T$:
  - the set of tokens for x (the 1's in its bitvectors)
- ❑ $x.succ \subseteq V$:
  - the successors of x (the edges)
- ❑ $x.cond(t) \subseteq V \times V$:
  - the conditional constraints for x and t
- ❑ $W \subseteq T \times V$:
  - a worklist (initially empty)

$O(\mathrm{n}^3)$的时间复杂度，n是变量数量

- $t \in x$

```
addToken(t, x)
propagate()
```

- $x \subseteq y$

```
addEdge(x, y)
propagate()
```

- $t \in x \rightarrow y \subseteq z$

```
if t ∈ x.sol
  addEdge(y, z)
  propagate()
else
  add (y, z) to x.cond(t)
```

```
addToken(t, x):
  if t ∉ x.sol
    add t to x.sol
    add (t, x) to W

addEdge(x, y):
  if x ≠ y ∧ y ∉ x.succ
    add y to x.succ
    for each t in x.sol
      addToken(t, y)

propagate():
  while W ≠ ∅
    pick and remove (t, x) from W
    for each (y, z) in x.cond(t)
      addEdge(y, z)
    for each y in x.succ
      addToken(t, y)
```

Andersen's Analysis

❑使用Cubic框架求解

$$\frac{}{[\![n\colon p := malloc()]\!] \hookrightarrow l_n \in p}\ malloc$$

$$\frac{}{[\![p := \&x]\!] \hookrightarrow l_x \in p}\ address\text{-}of$$

第一种形式：$t \in x$

$$\frac{}{[\![p := q]\!] \hookrightarrow p \supseteq q}\ copy$$

第二种形式：$x \subseteq y$

$$\frac{}{[\![*p := q]\!] \hookrightarrow *p \supseteq q}\ assign$$

$$\frac{}{[\![p := *q]\!] \hookrightarrow p \supseteq *q}\ dereference$$

第三种形式：

$\&x \in p \Rightarrow q \subseteq x$

$\&y \in q \Rightarrow y \subseteq p$

- ➢改进：SCC缩点、类型过滤（eliminate type-incompatible Objects）

❑理论复杂度$O(n^3)$, 实际复杂度$O(n^2)$ (Sridharan et al. [SAS,09])

Andersen's Analysis

❑一些扩展

➢域敏感（field sensitivity)

- 域不敏感： treats fields `.f` as dereferences `*`
- 区分不同的域（field），在Java指针分析中很重要

$$\frac{}{[\![p := q.f]\!] \hookrightarrow p \supseteq q.f}\ \textit{field-read}$$

$$\frac{}{[\![p.f := q]\!] \hookrightarrow p.f \supseteq q}\ \textit{field-assign}$$

c/c++转化成第二种形式：

$$x \subseteq y$$

Java转化成第三种形式(p.f相当于c/c++中的(*p).f)：

$$\mathrm{O} \in q \Rightarrow \mathrm{O.f} \subseteq p$$

$$\mathrm{O} \in p \Rightarrow q \subseteq \mathrm{O.f}$$

➢支持函数调用

- 参数和返回值的传递建模成copy语句

```
r = m(a1, ..., an)
```

```
T m(T p1, ... T pn) {
    ...
    return ret;
}
```

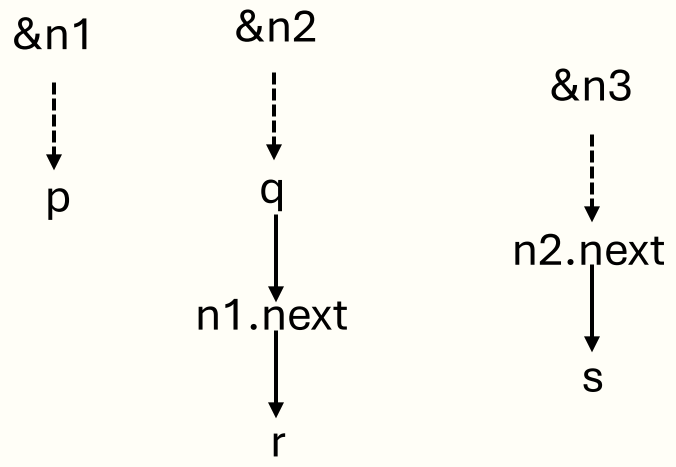
$$\frac{}{[\![r = m(a1, \cdots, an)]\!] \hookrightarrow p1 \supseteq a1, \ldots, pn \supseteq an, r \supseteq ret}\ \mathit{call}$$

Example Program

```c
#include <stdio.h>
struct Node { struct Node *next; };
int main() {
    struct Node n1, n2, n3;
    struct Node *p = &n1;
    struct Node *q = &n2;
    n1.next = q;
    struct Node *r = p->next;
    n2.next = &n3;
    struct Node *s = r->next;
    return 0;
}
```

留了个更复杂的例子作为可选课后作业

Constraint Graph (so-called Pointer Assignment Graph)



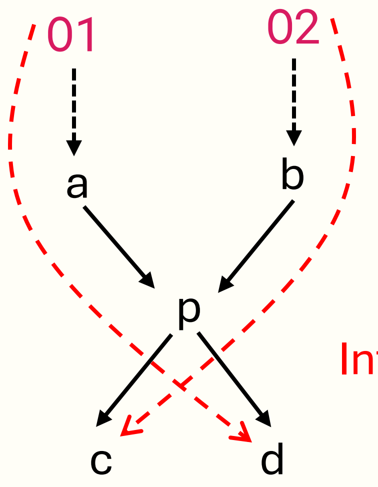
Vars	PTS	Vars	PTS
p	{&n1}	n2.next	{&n3}
q	{&n2}	s	{&n3}
n1.next	{&n2}	r	{&n2}

Context Sensitivity

```c
#include <stdlib.h>
int *id(int *p) {
    return p;
}
int main() {
    int *a, *b, *c, *d;
    a = (int *)malloc(sizeof(int)); // O1
    c = id(a);
    b = (int *)malloc(sizeof(int)); // O2
    d = id(b);
    return 0;
}
```

- Andersen's analysis is context-insensitive
  - Joins information across callsits to same function
  - Loses precision due to modeling infeasible paths
  - Can we "remember" where to return?

Constraint Graph (so-called PAG)



Infeasible value flow paths

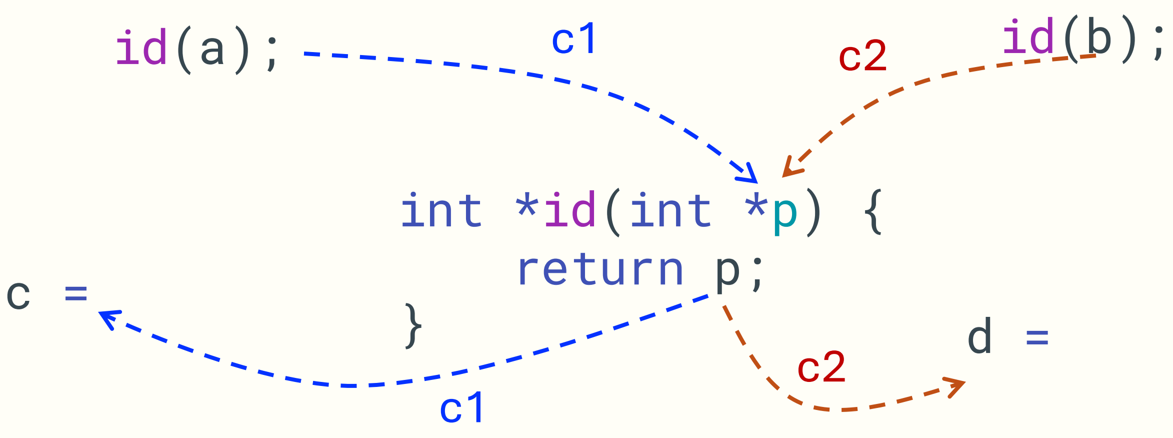
Vars	PTS	Vars	PTS
a	{O1}	b	{O2}
p	{O1, O2}		
c	{O1, O2}	d	{O1, O2}

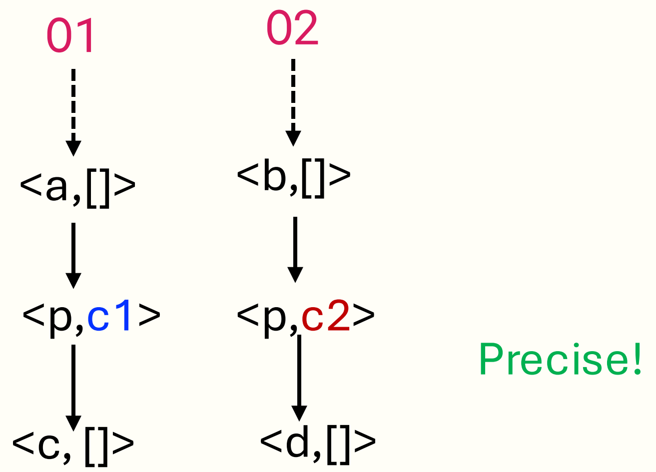
Spurious Points-to Objects

Context Sensitivity

❑ **Key idea:**

- Separate analyses for functions called in different "contexts"
- "context": an abstract concept that is statically definable; used to differentiate different calls to a function
- Contexts are used to decorate graph nodes





Vars	PTS	Vars	PTS
<a, []>	{O1}	<b, []>	{O2}
<p, c1>	{O1}	<p, c2>	{O2}
<c, []>	{O1}	<d, []>	{O2}

Types of Context Sensitivity

- ❑ No context sensitivity, []
- ❑ Callsite sensitivity
  - ➢ Call strings, $[l_1, \cdots, l_n]$
  - ➢ l_i: the line label of the callsite
- ❑ Object sensitivity, $[o_1, \cdots, o_n]$
  - ➢ o_i is an allocation site
- ❑ Type sensitivity, $[t_1, \cdots, t_n]$
  - ➢ t_i is the type of an allocation object
- ❑ Value contexts
  - ➢ States (environment) at the given callsite
- ❑ k – limiting technique
- ❑ Unscalable when $k \geq 3$ in practice

```
1.  int *id(int *p) {
2.    return p;
3.  }
4.  int *wid_1(int *p_1) {
5.    return id(p_1);
6.  }
7.  ...
8.  int *wid_n(int *p_n) {
9.    return wid_(n-1)(p_n);
10. }
11. c = wid_n(a);
12. d = wid_n(b);
```

Two contexts for differentiate id():

[l5, …, l9, l11]
[l5, …, l9, l12]

Context length: n+1

```
int factorial(int* n) {
  if (*n == 0) {
    return 1;
  } else {
    int t = *n;
    *n = t - 1;
    return t * factorial(n)
  }
}
int x = 100;
int y = factorial(&x);
```

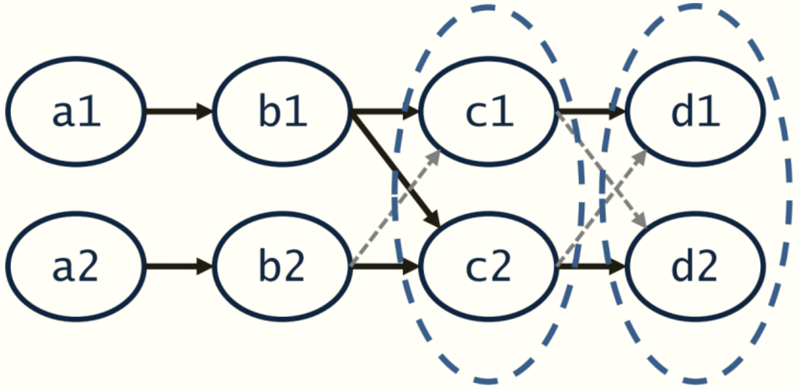
Context length could be infinite.

Steensgaard's Analysis

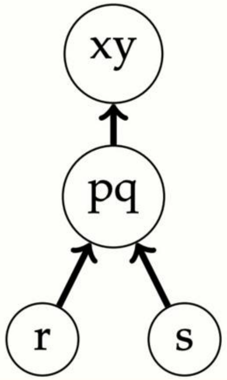
- **Problem:** Quadratic-in-practice is still not ultra-scalable
  - **Want a faster algorithm! Need ~LINEAR. How?**
- **Challenge:**
  - Solution space of pointer analysis (e.g. points-to sets) is $O(n^2)$
- **Key idea:**
  - Use constant-space per pointer.
  - Merge aliases and alternates into the same equivalence class.
  - p can point to q or r? Let's treat q and r as the same pseudo-var and merge everything we know about q and r.
  - Points-to "sets" are basically singletons
- **Algorithm runs in $\mathrm{O}(\mathrm{n} * \alpha(\mathrm{n}))$ using union-find structure**
  - Almost linear, very scalable in practice (Millions of LoC)
  - Yes, very imprecise!

Steensgaard's Analysis-Examples

```
a1 = &b1;
b1 = &c1;
c1 = &d1;
a2 = &b2;
b2 = &c2;
c2 = &d2;
b1 = &c2;
```



$$
\begin{aligned}
1:&\quad p := \&x\\
2:&\quad r := \&p\\
3:&\quad q := \&y\\
4:&\quad s := \&q\\
5:&\quad r := s
\end{aligned}
$$



Steensgaard's Analysis

$$\frac{}{[\![p := q]\!] \hookrightarrow join(*p, *q)}\ copy$$

$$\frac{}{[\![p := \&x]\!] \hookrightarrow join(*p, x)}\ address\text{-}of$$

$$\frac{}{[\![p := *q]\!] \hookrightarrow join(*p, **q)}\ dereference$$

$$\frac{}{[\![*p := q]\!] \hookrightarrow join(**p, *q)}\ assign$$

```
join(ℓ1,ℓ2)
    if (find(ℓ1) == find(ℓ2))
        return
    n1 ← *ℓ1
    n2 ← *ℓ2
    union(ℓ1,ℓ2)
    join(n1,n2)
```

- **Points-to "sets" are basically singletons**

时间复杂度$O(n * \alpha(n))$ (almost linear)，空间复杂度$O(n)$.

(Optional）作业：指针分析应用

❑用Andersen算法分析如下代码

```c
#include <stdio.h>
struct Node { struct Node *next; int *data; };
int main() {
  int a, b, c;  struct Node n1, n2, n3;
  int *pa = &a; int *pb = &b; int *pc = &c;
  struct Node *p = &n1;
  struct Node *q = &n2;
  struct Node *r = &n3;
  n1.next = &n2; q->next = r;
  p->data = pa; n2.data = &b
  struct Node *x = p->next;
  struct Node *y = x->next;
  int *d1 = p->data; int *d2 = q->data;
  r->data = pc;
  y = p->next;
  return 0;
}
```

❑用Steensgaard算法分析

```c
#include <cstdlib>
struct Node { struct Node *next; };
int main() {
  struct Node *a, *b, *c, *tmp;
  a = (struct Node *)malloc(sizeof(struct
Node));
  b = (struct Node *)malloc(sizeof(struct
Node));
  c = (struct Node *)malloc(sizeof(struct
Node));
  tmp = a;
  tmp = b;
  a = c;
  a->next = b;
  b->next = c;
  return 0;
}
```

（Optional）作业：熟悉指针分析实现

- **Qilin指针分析框架**
  - https://github.com/QilinPTA/Qilin
  - 支持多种指针分析优化技术
- **任务：**
  - 选一组Java程序（3-4个即可），尝试用Qilin框架的技术进行指针分析
  - 阅读源码，探究Qilin是怎么支持多种上下文敏感性的?
  - 指出Qilin框架设计和实现上可以改进和优化的地方?