Scribe: Piti Irawan, Milind Kulkarni Lecturer: Radu Rugina

1 Formalizing the analysis

$$c \ := \ [\textbf{skip}]^l \mid [x := a]^l \mid c_0; c_1 \mid \textbf{if } [b]^l \textbf{ then } c_0 \textbf{ else } c_1 \mid \textbf{while } [b]^l \textbf{ do } c$$

where l stands for the label of the command.

- lab, $linit \in Lab$ (lab stands for label, and $linit$ stands for initial label),
- $flow \ : \ Com \to \rho(Lab \times Lab)$

  flow can be defined inductively.

Analysis

- complete lattice La (abstract domain which satisfies ACC,
- initial information: $ia \in La$,
- monotonic transfer function $[\![c]\!]^l \ : \ La \to La$.

Analysis Result

- $Result \ : \ Lab \to (La \times La)$,
- $Result(\cdot l) = \pi_1(Result(l))$, where $Result$ is a pair, and $\cdot l$ is the information before the statement,
- $Result(l\cdot) = \pi_2(Result(l))$, where $l\cdot$ is the information after the statement.

How do we compute this information for an arbitrary computation?

System of Equations

$$Result(l\cdot) = [\![c]\!]^l \cdot Result(\cdot l)$$

is the result after the statement, and

$$Result(\cdot l) = \bigsqcup \{Result(l'\cdot) \mid (l', l) \in flow(P)\}$$

is the result before the statement, where $l \neq linit$, and P is the entire program.

For loops $Result(l\cdot)$ would be recursive.

$$Result(\cdot linit) = ia$$

We want the most precise analysis.

We can safely replace the $=$ with $\sqsupseteq$. For example:

$$\begin{aligned} Result(\cdot linit) &\sqsupseteq ia \\ Result(\cdot l) &\sqsupseteq \bigsqcup \{Result(l'\cdot) \mid (l', l) \in flow(P)\} \\ Result(l\cdot) &\sqsupseteq [\![c]\!]^l \cdot Result(\cdot l) \end{aligned}$$

Algorithm

- initialize $\perp$ (i.e. the most precise information)
- iterate
- reach the fixed point
- lf_p (i.e. least fixed solution)

We won't show the algorithm.

Formalize the analysis.

$c ::= \ [\textbf{skip}]^l \mid [x := a]^l \mid c_0; c_1 \mid \textsf{if } [b]^l \textsf{ then } c_0 \textsf{ else } c_1 \mid \textsf{while } [b]^l \textsf{ do } c$

$l_{init} \in lab$

flow: Com $\rightarrow \mathcal{P}(lab \times lab)$

Analysis

Comple lattice L_a (abstract domain) satisfies Ascending Chain Theorem

Initial information $i_a \in L_a$

Transfer function $[\![c]\!]^l : \ L_a \rightarrow L_a$

Analysis Result

Result: $lab \rightarrow (L_a \times L_a)$

Result$(\bullet l) = \pi_1(\text{Result}(l))$

Result$(l\bullet) = \pi_2(\text{Result}(l))$

System of Equation

Result$(l\bullet) = [\![c]\!]^l \text{Result}(\bullet l)$

Result$(\bullet l) = \bigsqcup\{\text{Result}(l'\bullet) \mid (l', l) \in flow(P)\}\ l \in l_{init}$

Result$(l_{init}) = i_a$

We can savely replace "=" with "$\sqsupseteq$" but we may get less precise solution. The set of solution obtained with "$\sqsupseteq$" is larger than the one obtained with "=".

Algorithm for performing analysis

- initialize $\perp$ ($\perp$ is the most precise information)
- iterate over the system of equation above until we reach the fixed point
- this should give us the least fixed point solution

What does this least fixed point mean in Lattices?

2 Fixed Points for Lattices

Recall that we can define a complete lattice as follow: $(L, \sqsubseteq, \sqcap, \sqcup, \top, \perp)$

Where $L = \{i_a\} \times L_a^{2lab-1}$ and $\perp_L = \{i_a, \perp_{L_a}, \ldots, \perp_{L_a}\}$

lab is the number of labels in the program.

So we take a monotonic function $f : L \rightarrow L$

Starting with $\perp$, we iteratively compute $f(\perp), f^2(\perp), \ldots, f^n(\perp)$

Claim: $\bigsqcup f^n(\perp) =$ least fixed point of $f = \bigsqcup Fix(f)$ where $Fix(f) = \{x | f(x) = x\}$

Alternatively, instead of starting from $\perp$ we can start from $\top$ and iteratively compute $f^n(\top)$, then $\sqcap f^n(\top)$ = greatest fixed point of $f = \sqcap Fix(f)$

Insert figure here

When will least fixed point of $f = \bigsqcup f^n(\perp)$?

- f is continuous.
- L satisfies Ascending Chain Condition.

What happens if L doesn't satisfy Ascending Chain Condition?

- if f is not continuous, then our solution may not be safe (safe means the solution is between the least fixed point of f and $\top$)
- even if f is continuous, our algorithm may not terminate

3 Range Analysis

Range analysis can be used for array bound analysis.
It allows us to check the possible values for variables.
We will define our set of ranges as follow:
Range $= \{\perp\} \cup \{[l, u] \mid l \in \mathbb{Z} \cup \{-\infty\},\ u \in \mathbb{Z} \cup \{\infty\},\ l \leq u\}$
Now we will define our complete lattice:
$(\text{Range}, \sqsubseteq, \sqcap, \sqcup, \top, \perp)$
Where $\top = [-\infty, \infty]$ and

$$r_1 \sqcup r_2 = \begin{cases} r_1 & r_2 = \perp \\ r_2 & r_1 = \perp \\ [min(l_1, l_2), max(u_1, u_2)] \text{ where } r_1 = [l_1, u_1] \wedge\ r_2 = [l_2, u_2] & \text{otherwise} \end{cases}$$

The other operations follow from the definition of $\bigsqcup$.

For example,

```
x := 0;
while (m >= 1) x := x + 1;
```

Note that as we calculate $f^n(\perp)$, we get the following chain:
$[0, 0],\ [0, 1],\ [0, 2],\ \ldots$
This is because the lattice can have infinite ascending chain.
What we want is to skip iterating and just say that after all loops are executed we will get $[0, \infty]$
To do so we will introduce the concept of widening operator.

4 Widening Operators

4.1 Upper Bound Operators

Let (L,$\sqsubseteq$) to be a complete lattice. For the definition of lattice, refer to domain construction lectures. Then, $\check{\sqcup} : L \times L \to L$ is an upper bound operator if $\forall l, l' \bullet l \sqsubseteq l \check{\sqcup} l' \sqsupseteq l'$.

Define the chain l_n recursively using operator $\Phi : L \times L \to L$:

$$\{l_n^\Phi\}_{n \geq 0} = \begin{cases} l_0 & \text{if n = 0} \\ l_{n-1}^\Phi \Phi l_n & \text{if n >0} \end{cases}$$

Notice that if Φ is an upper-bound operator then $\{l_n^\Phi\}_{n \geq 0}$ increases at least as fast as l_n.

4.2 Widening Operators

Now that we defined the upper-bound operators, we can define widening operator $\nabla : L \times L \to L$ as:
∇ is widening if:

- ∇ is an upper bound operator
- $\{l_n^{\nabla}\}_{n\geq 0}$ stabilizes for any $\{l_n\}_{n\geq 0}$

Now let's try to compute lfp(f), before we tried to compute $\sqcup f^n(\perp)$

Property
If (L,$\sqsubseteq$) is a lattice and f: L $\to$ L is monotonic, and $\nabla : L \times L \to L$ is widening:

$$\text{Let } f_{\nabla}^{n} = \begin{cases} \perp & \text{if n} = 0 \\ f_{\nabla}^{n-1} & \text{if } f(f_{\nabla}^{n-1}) \sqsubseteq f_{\nabla}^{n-1} \text{ (stabilization)} \\ f_{\nabla}^{n-1} \nabla f(f_{\nabla}^{n-1}) & \textit{otherwise} \end{cases}$$

Then,

- $\{l_n^{\nabla}\}_{n\geq 0}$ stabilizes.
- $\sqcup f_{\nabla}^{n} \sqsupseteq$ lfp(f), i.e. we are guaranteed to stabilize in the safe region.

Let's return to the example that we were doing. Define a monotonic function f on the Range domain.
Let f: Range $\to$ Range

$$f(\perp) \quad = \quad [0,0]$$

$$f([l,u]) = \begin{cases} [l, u+1] & \text{if } u \neq \infty \\ [l,\infty] & \text{if } u = \infty \end{cases}$$

Easy to see that $f^n(\perp) = [0,n]$.
Take K to be a set of constants. Define $\nabla : \textit{Range} \times \textit{Range} \to \textit{Range}$

$$\nabla([l_1,u_1][l_2,u_2]) = [min_k(l_1,l_2), max_k(u_1,u_2)]$$

The idea is to never use l_2. So define max$k$ *and min*k functions as such:

$$min_k(l_1,l_2) = \begin{cases} -\infty & if l_2 \langle l_1 \wedge l_2 \langle min(K) \\ max\{n | n \in \{K \cup \{l_1\}\} \wedge n \leq min(l_1,l_2)\} & otherwise \end{cases}$$

Max can be defined similarly. So this min function never chooses l_2, instead it either chooses l_1 or a constant from the set K given that the element chosen is less than both l_1 and l_2. We want the maximum of such elements because we want to be as precise as possible. To see how this works:
Set K = {-1,5,10}

$$\begin{array}{ll} f_{\nabla}^{0} = \perp & f(f_{\nabla}^{0}) = \quad [0,0] \\ f_{\nabla}^{1} = [-1,5] & f(f_{\nabla}^{1}) = \quad [-1,6] \\ f_{\nabla}^{2} = [-1,10] & f(f_{\nabla}^{2}) = \quad [-1,11] \\ f_{\nabla}^{3} = [-1,\infty] & \end{array}$$

To see how we get the first column, consider applying ∇ operator to [-1,5] and [-1,6] gives [-1,10]. $min_k(-1,-1)$ gives -1 since -1 is less than -1 and -1 (second condition). $max_k(5,6)$ gives 10 since 6 is greater than 5 but we dont want to select l_2 so we choose something from set K and the only constant that is greater than both 5 and 6 is 10. In the last iteration, we cannot find such a constant so we just return ∞. Notice that this range is less precise than the correct one [0,∞] but nevertheless it is safe.