Lecture 19

Wednesday, April 1, 2020 4:47 PM

- Hello!

- Unit 1: Hoare logic; partial correctness; termination
- Unit 2: SAT & SMT solvers
- Unit 3: Static analysis & abstract interpretation
  - Predicate abstraction
  - Counterexample-Guided Abstraction Refinement (CEGAR)
  - Concrete states $\underset{\gamma}{\overset{\alpha}{\rightleftarrows}}$ Abstract states

- Dataflow analysis
  - Ignoring pointers/heap.
  - Focussing on Imp.

Recap of Imp

Commands

$c ::= c_1; c_2 \mid \text{if (Bexp) } \{c_1\} \text{ else } \{c_2\}$

$\mid x := AExp \mid \text{while (BExp) } \{c_1\}$

Expressions

Expressions

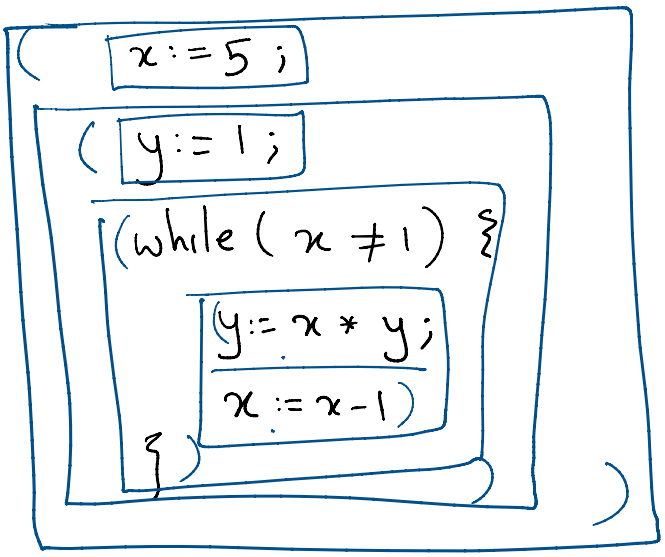
$AExp ::= n \mid x \mid AExp_1 + AExp_2$ (Arithmetic)

$\mid AExp_1 - AExp_2 \mid AExp_1 * AExp_2$

$BExp ::= AExp_1 \leq AExp_2 \mid BExp_1 \text{ and } BExp_2$

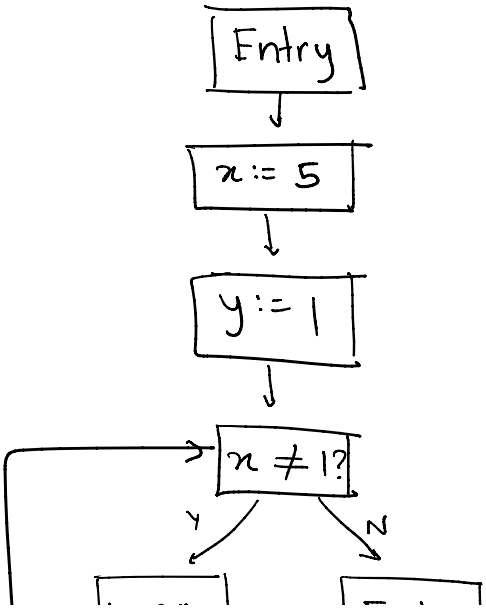
$\mid \text{not } BExp_1$

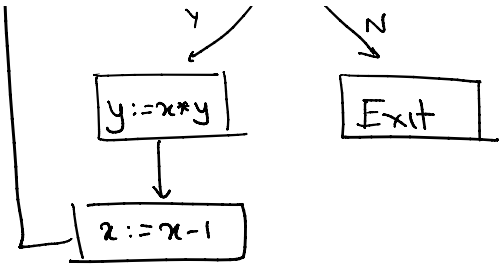
```
x := 5 ;
y := 1 ;
while ( x ≠ 1 ) {
    y := x * y ;
    x := x - 1
}
```

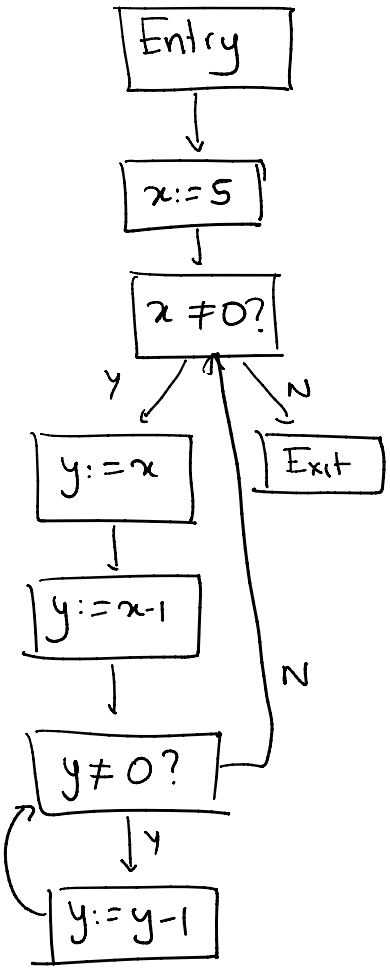


Control Flow Graph

```
x := 5 ;
y := 1 ;
while ( x ≠ 1 ) {
    y := x * y ;
    x := x - 1
}
```








```
x := 5;
while (x≠0) {
    y := x;
    y := x-1;
    while (y≠0) {
        y := y-1
    }
}
```

(Observation: This program never terminates.)

Side note:

1. Program analysis is undecidable
2. There is no algorithm which accurately

solves the program analysis problem.

3. Sacrifice either: soundness / completeness / termination

If the program has a behavior (or a bug) then the analysis concludes that the program has the behavior.

converse

If the analysis concludes that the program may exhibit a behavior, then the program definitely exhibits the behavior.

alarm

No false negatives

High recall. (1.0)

Bug ⇒ Alarm

⇔ ~~Alarm~~ ⇒ ~~Bug~~

~~Alarm~~ but Bug.

False negative

No false positives

High precision (1.0)

Alarm ⇒ Bug

⇔ ~~Bug~~ ⇒ ~~Alarm~~

Alarm but ~~Bug~~

False positive

Types of dataflow analysis

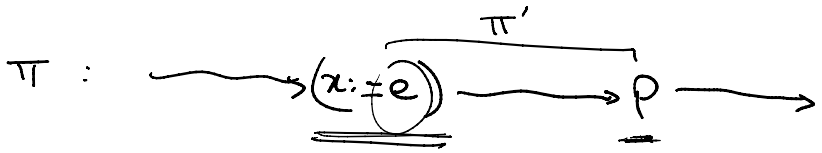
1. Reaching Definitions Analysis

   Pick some assignment statement

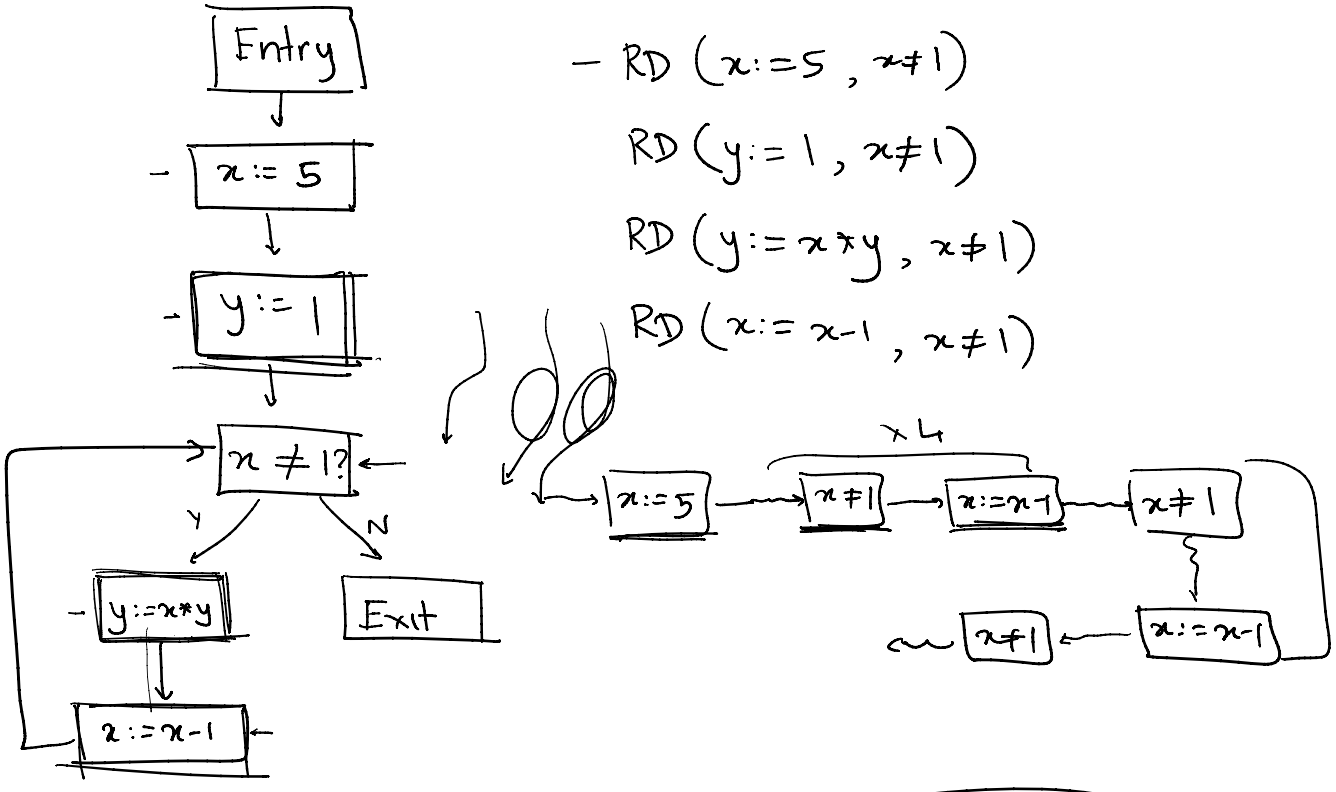
$x := e$

from the program. Pick another node p from the control flow graph.

"$x := e$" can reach p if <u>there is some program execution</u> π which passes through both p & $x := e$



s.t. x is never reassigned in the region π'.



- RD ($x := 5$, $x \neq 1$)

RD ($y := 1$, $x \neq 1$)

RD ($y := x * y$, $x \neq 1$)

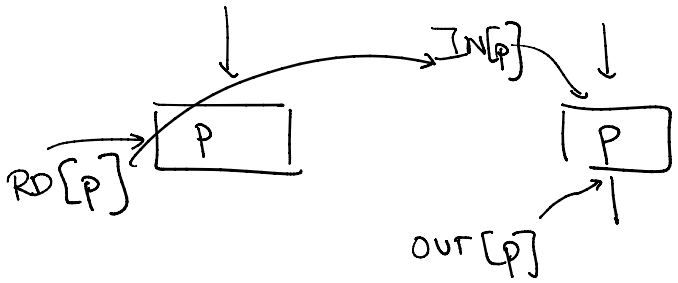
RD ($x := x-1$, $x \neq 1$)

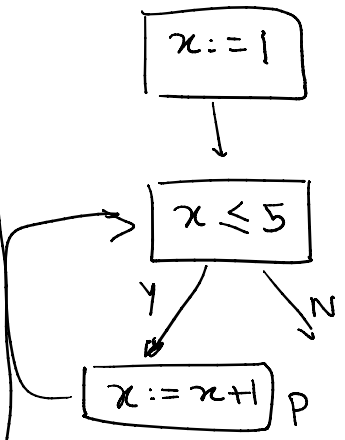
On the other hand, ~~RD ($y := 1$, $x := x-1$)~~

On the other hand, RD(y:=1, x:=x-1)

$$RD[p] = \{ x := e \mid RD(x := e, p) \}$$

How to compute RD [p] ?





IN[p] = all variable definitions which can possibly reach the beginning of statement p.

OUT[p] = all variable definitions which can possibly reach the end of p.

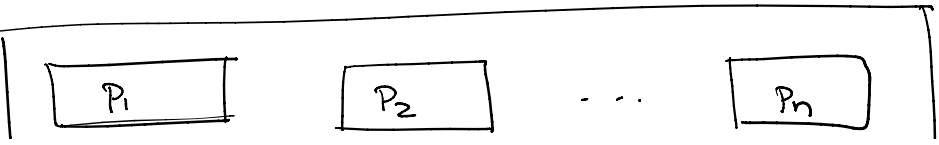
$RD[p] = \{ x := 1, x := x+1 \}$

$IN[p] = \{ x := 1, x := x+1 \}$

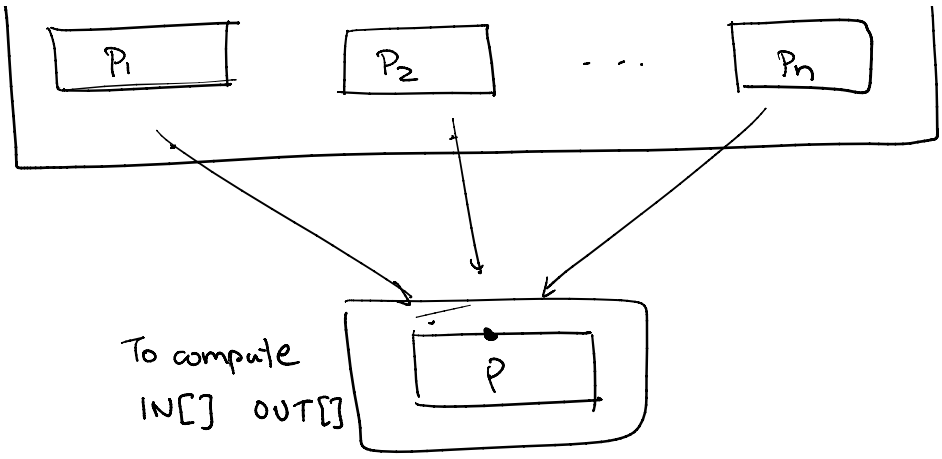
$OUT[p] = \{ x := x+1 \}$

Observation #1

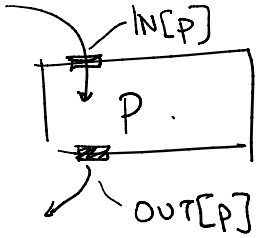
Given :

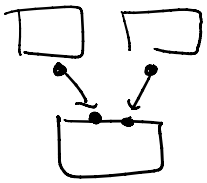


Given :
IN[], OUT[]

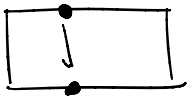


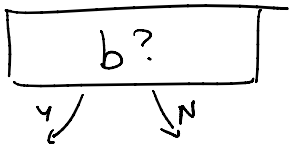
Claim : $IN[p] = \bigcup_i OUT[p_i]$ (all predecessors)

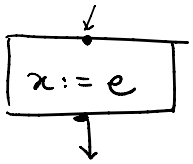


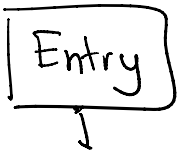


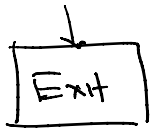
Observation # 2











Claim : $OUT[b?] = IN[b?]$

$OUT[x := e] = \left(IN[x := e] \setminus \{x := e' \mid \forall e'\}\right) \cup \{x := e\}$

$OUT[Entry] = \{(x := ?) \mid \forall x\}$

Both observations made in the path-insensitive setting. We assume every path in the CFG is feasible.

Claim: $IN[p] = \bigcup_i OUT[p_i]$ (all predecessors)

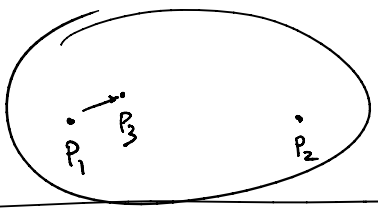
Claim: $OUT[b?] = IN[b?]$

$$OUT[x := e] = \left(IN[x := e] \setminus \{x := e' \mid \forall e'\}\right) \cup \{x := e\}$$

$$OUT[Entry] = \{(x := ?) \mid \forall x\}$$

Algorithm to Construct IN[p] & OUT[p]

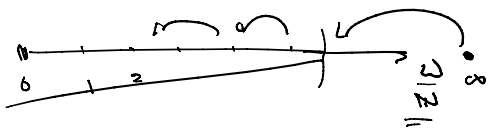
Ex: Computing shortest paths in a graph.



$\forall P_3$: $sp(P_1\ P_2) \le w(P_1\ P_3) + sp(P_3\ P_2)$

$sp(P_1\ P_2) \le w(P_1\ P_2)$.

$sp(P_1\ P_2) := \infty$, $\forall P_1\ P_2$



$sp(p_1\ p_2) := \infty$, $\forall\ p_1\ p_2$

(Until fixpoint :) (until it converges)/

Doesn't change from one iteration to the next.

$\forall\ p_1\ p_2\ p_3$:

$$sp(p_1\ p_2) = \min\left(sp(p_1\ p_2),\ w(p_1\ p_3) + sp(p_3\ p_2)\right)$$

$\forall\ p_1\ p_2$:

$$sp(p_1\ p_2) = \min\left(sp(p_1\ p_3),\ w(p_1\ p_2)\right)$$

Q1. Does this procedure terminate?

Q2. Does it compute the shortest path?