



CSCI 740 - Programming Language Theory

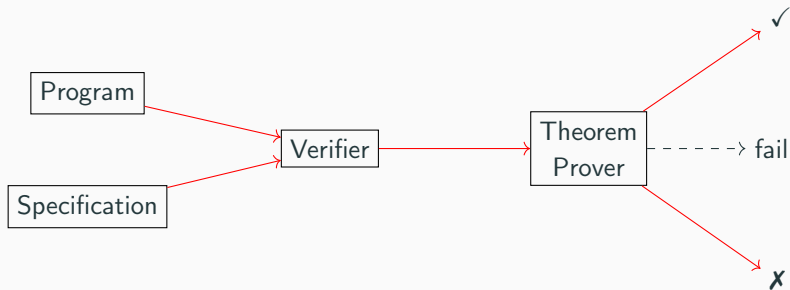
Lecture 23

Verification by Solving Horn Clauses

Instructor: Hossein Hojjat

October 27, 2017

Recap: Automated Verification



Weakest Precondition Rules: Summary

c	$\text{wp}(c, Q)$
$x := e$	$Q[x \mapsto e]$
$\text{assume}(b)$	$b \rightarrow Q$
$\text{assert}(b)$	$\text{wp}(b \wedge Q)$
$\text{havoc}(x)$	$\forall y. Q[x \mapsto y]$
$c_1; c_2$	$\text{wp}(c_1, \text{wp}(c_2, Q))$
$\text{if } b \text{ then } c_1 \text{ else } c_2$	$b \rightarrow \text{wp}(c_1, Q) \wedge \neg b \rightarrow \text{wp}(c_2, Q)$
$\text{while } b \text{ do } c$	$I \wedge \forall \vec{y}. \left((I \wedge b \rightarrow \text{wp}(c, I)) \wedge (I \wedge \neg b \rightarrow Q) \right) [\vec{x} \mapsto \vec{y}]$ (\vec{x} are variables modified in c and I is the loop invariant)

Loop Invariant

$$\frac{\vdash \{A \wedge b\} c \{A\}}{\vdash \{A\} \text{ while } b \text{ do } c \{A \wedge \neg b\}}$$

$$\text{wp}(\text{while } b \text{ do } c, Q) = I \wedge \forall \vec{y}. \left((I \wedge b \rightarrow \text{wp}(c, I)) \wedge (I \wedge \neg b \rightarrow Q) \right) [\vec{x} \mapsto \vec{y}]$$

(\vec{x} are variables modified in c and I is the loop invariant)

- Unfortunate reality: Inferring invariants automatically is undecidable
- This puts significant limits on the degree to which we can automate verification

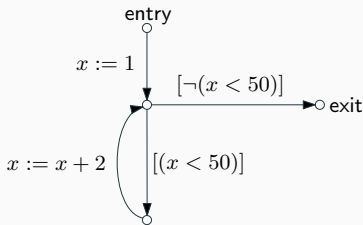
Loop Invariant

- Active research area:
how to find loop invariants efficiently and automatically
- The simplest technique: **guess-and-check!**
- Given template of invariants (e.g., $? = ?$, $? \leq ?$), instantiate the holes with program variables and constants
- Check if it's an invariant; if not, try a different instantiation
- **Abstract interpretation**: popular approach to discover invariants
 - We will discuss it later in the course
- Today we discuss an alternative approach: reducing verification to checking the satisfiability of **Horn clauses**
- First, let's discuss control-flow graphs

Control Flow Graphs

- **Control Flow Graph (CFG):** graph representation of computation and control flow in the program
- Highlights the possible flow of execution

```
x = 1
while (x < 50) {
    x = x + 2
}
```



Generating Control-Flow Graphs

- Start with graph that has one **entry** and one **exit** node
- Draw an edge from entry to exit and label it with the entire program

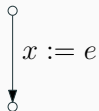


- Recursively decompose the program to have more edges with simpler labels
- When labels cannot be decomposed further, we are done

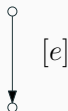
Basic Operations

- Base cases

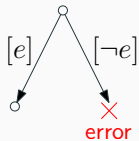
Assignment



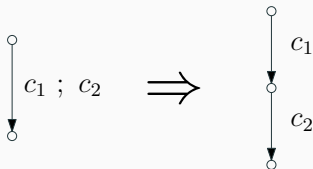
assume(e)



assert(e)

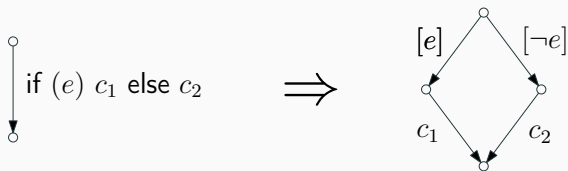


- Sequence of statements



Control Structures

- Conditional statement



- While loop

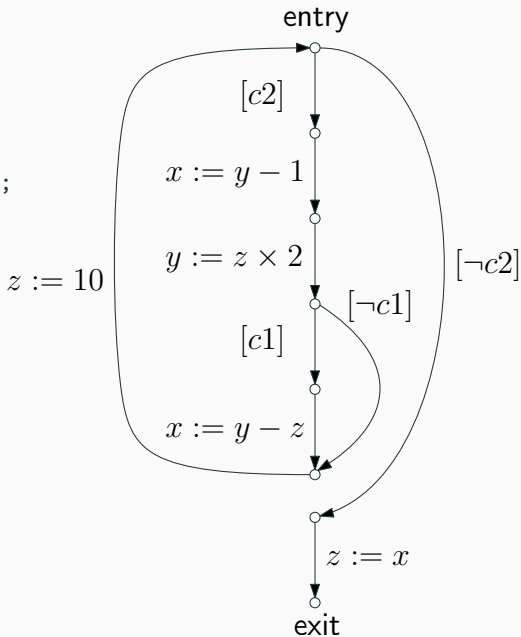


Exercise: Convert to CFG

```
while(c2) {  
    x = y - 1;  
    y = z * 2;  
    if (c1) x = y - z;  
    z = 10;  
}  
z = x;
```

Exercise: Convert to CFG

```
while(c2) {  
  x = y - 1;  
  y = z * 2;  
  if (c1) x = y - z;  
  z = 10;  
}  
z = x;
```



Example

- How to prove that the assertion does not fail in this program?

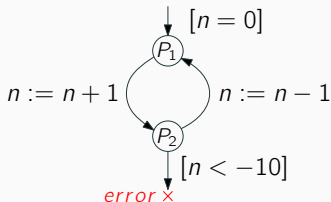
```
int n = 0;
while (true) {
    n = n + 1;
    assert(n ≥ -10);
    n = n - 1;
}
```

Example

- How to prove that the assertion does not fail in this program?

```
int n = 0;
while (true) {
  n = n + 1;
  assert(n ≥ -10);
  n = n - 1;
}
```

Control Flow Graph

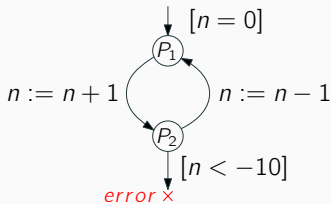


Example

- How to prove that the assertion does not fail in this program?

```
int n = 0;
while (true) {
  n = n + 1;
  assert (n ≥ -10);
  n = n - 1;
}
```

Control Flow Graph



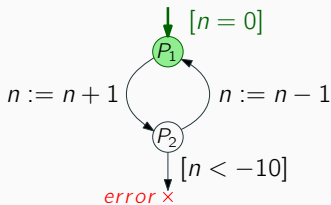
- Let $P_i(n)$ denotes the reachable values of n in state P_i
- Initially we do not know the set of reachable values of n in each state
- P_i 's can be any Boolean formula on n , for example:
 - $P_1(n) = (n \geq 0)$ and $P_2(n) = (n = -5) \vee (n > 0)$
 - $P_1(n) = (n = 0)$ and $P_2(n) = \text{true}$ (any value can reach it)
 - ...
- We can write constraints between P_i 's according to CFG

Example

- How to prove that the assertion does not fail in this program?

```
int n = 0;
while (true) {
  n = n + 1;
  assert(n ≥ -10);
  n = n - 1;
}
```

Control Flow Graph



$(n = 0)$

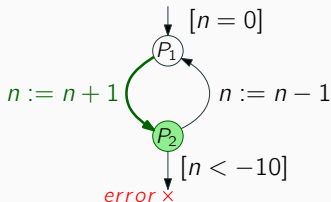
$\rightarrow P_1(n)$

Example

- How to prove that the assertion does not fail in this program?

```
int n = 0;
while (true) {
  n = n + 1;
  assert(n ≥ -10);
  n = n - 1;
}
```

Control Flow Graph



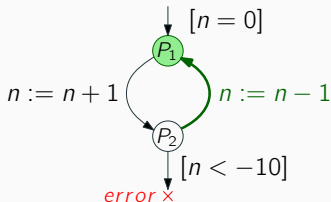
$$\begin{aligned}(n = 0) &\rightarrow P_1(n) \\ P_1(n) \wedge (n' = n + 1) &\rightarrow P_2(n')\end{aligned}$$

Example

- How to prove that the assertion does not fail in this program?

```
int n = 0;
while (true) {
  n = n + 1;
  assert(n ≥ -10);
  n = n - 1;
}
```

Control Flow Graph



$$(n = 0) \rightarrow P_1(n)$$

$$P_1(n) \wedge (n' = n + 1) \rightarrow P_2(n')$$

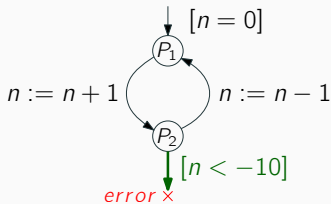
$$P_2(n) \wedge (n' = n - 1) \rightarrow P_1(n')$$

Example

- How to prove that the assertion does not fail in this program?

```
int n = 0;
while (true) {
  n = n + 1;
  assert(n ≥ -10);
  n = n - 1;
}
```

Control Flow Graph



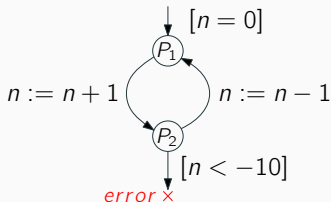
$(n = 0)$	$\rightarrow P_1(n)$
$P_1(n) \wedge (n' = n + 1)$	$\rightarrow P_2(n')$
$P_2(n) \wedge (n' = n - 1)$	$\rightarrow P_1(n')$
$P_2(n) \wedge (n < -10)$	$\rightarrow false$

Example

- How to prove that the assertion does not fail in this program?

```
int n = 0;
while (true) {
  n = n + 1;
  assert (n ≥ -10);
  n = n - 1;
}
```

Control Flow Graph



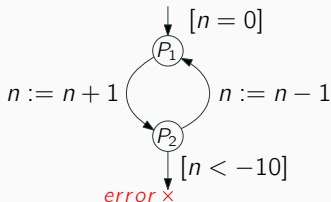
$\forall n. \quad (n = 0)$	$\rightarrow P_1(n)$
$\forall n, n'. \quad P_1(n) \wedge (n' = n + 1)$	$\rightarrow P_2(n')$
$\forall n, n'. \quad P_2(n) \wedge (n' = n - 1)$	$\rightarrow P_1(n')$
$\forall n. \quad P_2(n) \wedge (n < -10)$	$\rightarrow \text{false}$

Example

- How to prove that the assertion does not fail in this program?

```
int n = 0;
while (true) {
  n = n + 1;
  assert (n ≥ -10);
  n = n - 1;
}
```

Control Flow Graph



$$\forall n. \quad (n = 0) \quad \rightarrow \quad P_1(n)$$

$$\forall n, n'. \quad P_1(n) \wedge (n' = n + 1) \quad \rightarrow \quad P_2(n')$$

$$\forall n, n'. \quad P_2(n) \wedge (n' = n - 1) \quad \rightarrow \quad P_1(n')$$

$$\forall n. \quad P_2(n) \wedge (n < -10) \quad \rightarrow \quad \text{false}$$

Solvable: $P_1(n) \equiv (n \geq 0)$ and $P_2(n) \equiv (n \geq 1)$

- Try <https://rise4fun.com/>

$$\begin{aligned}\forall n. \quad (n = 0) & \rightarrow P_1(n) \\ \forall n, n'. \quad P_1(n) \wedge (n' = n + 1) & \rightarrow P_2(n') \\ \forall n, n'. \quad P_2(n) \wedge (n' = n - 1) & \rightarrow P_1(n') \\ \forall n. \quad P_2(n) \wedge (n < -10) & \rightarrow \text{false}\end{aligned}$$

```
(set-logic HORN)
(declare-fun P1 (Int) Bool)
(declare-fun P2 (Int) Bool)
(assert (forall ((n Int)) (=> (= n 0) (P1 n) )))
(assert (forall ((n Int) (np Int)) (=> (and (P1 n) (= np (+ n 1)))
                                         (P2 np))))
(assert (forall ((n Int) (np Int)) (=> (and (P2 n) (= np (- n 1)))
                                         (P1 np))))
(assert (forall ((n Int)) (=> (and (P2 n) (< n (- 10))) false)))
(check-sat)
(get-model)
```

- Horn clause is an implication of the form:

$$\forall \bar{v}. \underbrace{\Phi(\bar{v}) \wedge R_1(\bar{v}) \wedge \cdots \wedge R_n(\bar{v})}_{\text{body}} \longrightarrow \underbrace{R_0(\bar{v})}_{\text{head}}$$

- $\Phi(\bar{v})$ is an arithmetic formula (e.g. $x + 2y \leq z$)
- $R_i(\bar{v})$ is a relation symbol
- Head of the clause is either a relation symbol or *false*
- A solution for the Horn clause is an assignment of formulae to relation symbols for which the implication is valid

Verification by Solving Horn Clauses

```
// ...
// ...
// ...
if (newUserName != _userName
    || newPassword != _password)
    return false;
return true;
// ...
```

Code



Safety Description

$$\forall \bar{v}. \Phi^0(\bar{v}) \wedge R_1^0(\bar{v}) \wedge \dots \wedge R_n^0(\bar{v}) \rightarrow R_0^0(\bar{v})$$

$$\forall \bar{v}. \Phi^1(\bar{v}) \wedge R_1^1(\bar{v}) \wedge \dots \wedge R_n^1(\bar{v}) \rightarrow R_0^1(\bar{v})$$

⋮

$$\forall \bar{v}. \Phi^m(\bar{v}) \wedge R_1^m(\bar{v}) \wedge \dots \wedge R_n^m(\bar{v}) \rightarrow R_0^m(\bar{v})$$

$$\forall \bar{v}. \Phi^i(\bar{v}) \wedge R_1^i(\bar{v}) \wedge \dots \wedge R_n^i(\bar{v}) \rightarrow \text{false}$$



Horn Clause Solver

Exercise

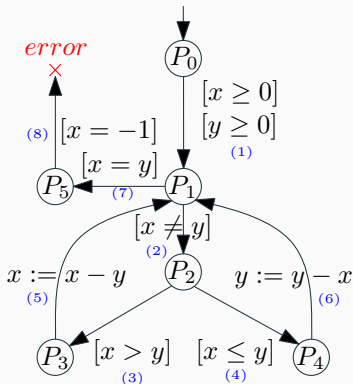
- Convert to Horn clauses

```
int x,y;  
assume(x ≥ 0 ∧ y ≥ 0);  
while(x ≠ y) {  
  if (x > y) then x := x - y;  
  else y := y - x;  
}  
assert(x ≠ -1);
```


Exercise

- Convert to Horn clauses

```
int x,y;  
assume (x ≥ 0 ∧ y ≥ 0);  
while (x ≠ y) {  
  if (x > y) then x := x - y;  
  else y := y - x;  
}  
assert (x ≠ -1);
```



Exercise

- Convert to Horn clauses

true

$$1) \quad P_0(x, y) \wedge (x \geq 0) \wedge (y \geq 0)$$

$$2) \quad P_1(x, y) \wedge (x \neq y)$$

$$3) \quad P_2(x, y) \wedge (x > y)$$

$$4) \quad P_2(x, y) \wedge (x \leq y)$$

$$5) \quad P_3(x, y) \wedge (x' = x - y)$$

$$6) \quad P_4(x, y) \wedge (y' = y - x)$$

$$7) \quad P_1(x, y) \wedge (x = y)$$

$$8) \quad P_5(x, y) \wedge (x = -1)$$

$$\rightarrow P_0(x, y)$$

$$\rightarrow P_1(x, y)$$

$$\rightarrow P_2(x, y)$$

$$\rightarrow P_3(x, y)$$

$$\rightarrow P_4(x, y)$$

$$\rightarrow P_1(x', y)$$

$$\rightarrow P_1(x, y')$$

$$\rightarrow P_5(x, y)$$

$$\rightarrow \text{false}$$

