Rigorous Software Development
CSCI-GA 3033-009

Instructor: Thomas Wies

Spring 2013

Lecture 12

Axiomatic Semantics

- An axiomatic semantics consists of:
  - a language for stating assertions about programs;
  - rules for establishing the truth of assertions.
- Some typical kinds of assertions:
  - This program terminates.
  - If this program terminates, the variables x and y have the same value throughout the execution of the program.
  - The array accesses are within the array bounds.
- Some typical languages of assertions
  - First-order logic
  - Other logics (temporal, linear)
  - Special-purpose specification languages (Z, Larch, JML)

Assertions for IMP

- The assertions we make about IMP programs are of the form:

  $\{A\}\ c\ \{B\}$

  with the meaning that:
  - If A holds in state q and $q \xrightarrow{c} q'$
  - then B holds in q'
- A is the pre-condition and B is the post-condition
- For example:

  $\{ y \leq x \}\ z := x;\ z := z + 1\ \{ y < z \}$

  is a valid assertion
- These are called Hoare triples or Hoare assertions

Semantics of Hoare Triples

- Now we can define formally the meaning of a partial correctness assertion:

  $\models \{A\}\ c\ \{B\}$ iff

  $\forall q \in Q.\ \forall q' \in Q.\ q \models A \wedge q \xrightarrow{c} q' \Rightarrow q' \models B$

- and the meaning of a total correctness assertion:

  $\models [A]\ c\ [B]$ iff

  $\forall q \in Q.\ q \models A \Rightarrow \exists q' \in Q.\ q \xrightarrow{c} q' \wedge q' \models B$

  or even better:

  $$\begin{array}{l} \quad \forall q \in Q.\ \forall q' \in Q.\ q \models A \wedge q \xrightarrow{c} q' \Rightarrow q' \models B \\ \wedge \\ \quad \forall q \in Q.\ q \models A \Rightarrow \exists q' \in Q.\ q \xrightarrow{c} q' \wedge q' \models B \end{array}$$

Inference Rules for Hoare Triples

- We write $\vdash \{A\}\ c\ \{B\}$ when we can derive the triple using inference rules
- There is one inference rule for each command in the language.
- Plus, the rule of consequence

$$\frac{\vdash A' \Rightarrow A \qquad \vdash \{A\}\ c\ \{B\} \qquad \vdash B \Rightarrow B'}{\vdash \{A'\}\ c\ \{B'\}}$$

Inference Rules for Hoare Logic

- One rule for each syntactic construct:

$$\vdash \{A\}\ \mathtt{skip}\ \{A\} \qquad\qquad \vdash \{A[e/x]\}\ x{:=}e\ \{A\}$$

$$\frac{\vdash \{A\}\ c_1\ \{B\} \quad \vdash \{B\}\ c_2\ \{C\}}{\vdash \{A\}\ c_1;\ c_2\ \{C\}}$$

$$\frac{\vdash \{A \wedge b\}\ c_1\ \{B\} \quad \vdash \{A \wedge \neg b\}\ c_2\ \{B\}}{\vdash \{A\}\ \mathtt{if}\ b\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2\ \{B\}}$$

$$\frac{\vdash \{I \wedge b\}\ c\ \{I\}}{\vdash \{I\}\ \mathtt{while}\ b\ \mathrm{do}\ c\ \{I \wedge \neg b\}}$$

Example: A Proof in Hoare Logic

- We want to derive that

$\{n \geq 0\}$

$p := 0;$

$x := 0;$

`while` $x < n$ do

$\quad x := x + 1;$

$\quad p := p + m$

$\{p = n * m\}$

Example: A Proof in Hoare Logic

Only applicable rule (except for rule of consequence):

$$\frac{\vdash \{A\}\ c_1 \{C\} \quad \vdash \{C\}\ c_2\ \{B\}}{\vdash \{A\}\ c_1;\ c_2\ \{B\}}$$

$$\frac{\vdash \{n \geq 0\}\ p{:=}0;\ x{:=}0\ \{C\} \quad \vdash \{C\}\ \text{while } x < n \text{ do } (x{:=}x{+}1;\ p{:=}p{+}m)\ \{p = n * m\}}{\vdash \underbrace{\{n \geq 0\}}_{A}\ \underbrace{p{:=}0;\ x{:=}0;}_{c_1}\ \underbrace{\text{while } x < n \text{ do } (x{:=}x{+}1;\ p{:=}p{+}m)}_{c_2}\ \underbrace{\{p = n * m\}}_{B}}$$

Example: A Proof in Hoare Logic

What is C? Look at the next possible matching rules for c_2!

Only applicable rule (except for rule of consequence):

$$\frac{\vdash \{\mathtt{I} \wedge b\}\ c\ \{\mathtt{I}\}}{\vdash \{\mathtt{I}\}\ \mathtt{while}\ b\ \mathrm{do}\ c\ \{\mathtt{I} \wedge \neg b\}}$$

We can match $\{\mathtt{I}\}$ with $\{C\}$ but we cannot match $\{\mathtt{I} \wedge \neg b\}$ and $\{p = n * m\}$ directly. Need to apply the rule of consequence first!

$$\frac{\vdash \{n \geq 0\}\ p{:=}0;\ x{:=}0\ \{C\} \qquad \vdash \{C\}\ \text{while } x < n \text{ do } (x{:=}x{+}1;\ p{:=}p{+}m)\ \{p = n * m\}}{\vdash \underbrace{\{n \geq 0\}}_{A}\ \underbrace{p{:=}0;\ x{:=}0;}_{c_1}\ \underbrace{\text{while } x < n \text{ do } (x{:=}x{+}1;\ p{:=}p{+}m)}_{c_2}\ \underbrace{\{p = n * m\}}_{B}}$$

Example: A Proof in Hoare Logic

What is C? Look at the next possible matching rules for c_2!

Only applicable rule (except for rule of consequence):

$$\frac{\vdash \{I \wedge b\}\ c\ \{I\}}{\vdash \underbrace{\{I\}}_{A}\ \underbrace{\texttt{while}\ b\ \texttt{do}\ c}_{c'}\ \underbrace{\{I \wedge \neg b\}}_{B}}$$

Rule of consequence:

$$\frac{\vdash A' \Rightarrow A \qquad \vdash \{A\}\ c'\ \{B\} \qquad \vdash B \Rightarrow B'}{\vdash \{A'\}\ c'\ \{B'\}}$$

$I = A = A' = C$

$$\frac{\vdash \{n \geq 0\}\ p{:=}0;\ x{:=}0\ \{C\} \qquad \vdash \overbrace{\{C\}}^{A'}\ \overbrace{\text{while } x < n \text{ do } (x{:=}x{+}1;\ p{:=}p{+}m)}^{c'}\ \overbrace{\{p = n * m\}}^{B'}}{\vdash \{n \geq 0\}\ p{:=}0;\ x{:=}0;\ \text{while } x < n \text{ do } (x{:=}x{+}1;\ p{:=}p{+}m)\ \{p = n * m\}}$$

Example: A Proof in Hoare Logic

What is $\mathtt{I}$? Let's keep it as a placeholder for now!

Next applicable rule:

$$\frac{\vdash \{A\}\ c_1 \{C\} \quad \vdash \{C\}\ c_2\ \{B\}}{\vdash \{A\}\ c_1;\ c_2\ \{B\}}$$

$$\dfrac{\dfrac{\vdash \underbrace{\{\mathtt{I} \wedge x<n\}}_{A}\ \underbrace{x := x+1;}_{c_1}\ \underbrace{p:=p+m}_{c_2}\ \underbrace{\{\mathtt{I}\}}_{B}}{\vdash \{\mathtt{I}\}\ \text{while } x < n \text{ do } (x:=x+1;\ p:=p+m)\ \{\mathtt{I} \wedge x \geq n\}} \quad \vdash \mathtt{I} \wedge x \geq n \Rightarrow p = n * m}{\vdash \{n \geq 0\}\ p:=0;\ x:=0\ \{\mathtt{I}\} \quad \vdash \{\mathtt{I}\}\ \text{while } x < n \text{ do } (x:=x+1;\ p:=p+m)\ \{p = n * m\}}$$

$$\frac{\vdash \{n \geq 0\}\ p:=0;\ x:=0\ \{\mathtt{I}\} \quad \vdash \{\mathtt{I}\}\ \text{while } x < n \text{ do } (x:=x+1;\ p:=p+m)\ \{p = n * m\}}{\vdash \{n \geq 0\}\ p:=0;\ x:=0;\ \text{while } x < n \text{ do } (x:=x+1;\ p:=p+m)\ \{p = n * m\}}$$

Example: A Proof in Hoare Logic

What is C? Look at the next possible matching rules for c_2!

Only applicable rule (except for rule of consequence):

$\vdash \{A[e/x]\}\ x{:=}e\ \{A\}$

$$\dfrac{\vdash \{n \geq 0\}\ p{:=}0;\ x{:=}0\ \{I\} \qquad \dfrac{\dfrac{\dfrac{\vdash \overbrace{\{I \wedge x<n\}}^{A}\ \overbrace{x := x+1}^{c_1}\ \{C\} \qquad \vdash \{C\}\ \overbrace{p{:=}p+m}^{c_2}\ \overbrace{\{I\}}^{B}}{\vdash \{I \wedge x<n\}\ x := x+1;\ p{:=}p+m\ \{I\}}}{\vdash \{I\}\ \text{while } x < n \text{ do } (x{:=}x+1;\ p{:=}p+m)\ \{I \wedge x \geq n\}} \qquad \vdash I \wedge x \geq n \Rightarrow p = n * m}{\vdash \{I\}\ \text{while } x < n \text{ do } (x{:=}x+1;\ p{:=}p+m)\ \{p = n * m\}}}{\vdash \{n \geq 0\}\ p{:=}0;\ x{:=}0;\ \text{while } x < n \text{ do } (x{:=}x+1;\ p{:=}p+m)\ \{p = n * m\}}$$

Example: A Proof in Hoare Logic

What is C? Look at the next possible matching rules for c_2!

Only applicable rule (except for rule of consequence):

$\vdash \{A[e/x]\}\ x:=e\ \{A\}$

$$\dfrac{\vdash \{n \geq 0\}\ p:=0;\ x:=0\ \{I\} \qquad \dfrac{\dfrac{\dfrac{\vdash \{I \land x<n\}\ x:=x+1\ \{I[p+m/p]\} \quad \vdash \{I[p+m/p]\}\ p:=p+m\ \{I\}}{\vdash \{I \land x<n\}\ x:=x+1;\ p:=p+m\ \{I\}}}{\vdash \{I\}\ \text{while } x < n \text{ do } (x:=x+1;\ p:=p+m)\ \{I \land x \geq n\}} \quad \vdash I \land x \geq n \Rightarrow p = n * m}{\vdash \{I\}\ \text{while } x < n \text{ do } (x:=x+1;\ p:=p+m)\ \{p = n * m\}}}{\vdash \{n \geq 0\}\ p:=0;\ x:=0;\ \text{while } x < n \text{ do } (x:=x+1;\ p:=p+m)\ \{p = n * m\}}$$

Example: A Proof in Hoare Logic

Only applicable rule (except for rule of consequence):

$\vdash \{A[e/x]\}\ x{:=}e\ \{A\}$

Need rule of consequence to match $\{I \wedge x<n\}$ and $\{I[x+1/x,\ p+m/p]\}$

$$\dfrac{\vdash \{n \geq 0\}\ p{:=}0;\ x{:=}0\ \{I\} \qquad \dfrac{\dfrac{\dfrac{\vdash \{I \wedge x<n\}\ x{:=}x+1\ \{I[p+m/p]\} \quad \vdash \{I[p+m/p]\}\ p{:=}p+m\ \{I\}}{\vdash \{I \wedge x<n\}\ x{:=}x+1;\ p{:=}p+m\ \{I\}}}{\vdash \{I\}\ \text{while } x<n \text{ do } (x{:=}x+1;\ p{:=}p+m)\ \{I \wedge x \geq n\}} \quad \vdash I \wedge x \geq n \Rightarrow p = n * m}{\vdash \{I\}\ \text{while } x<n \text{ do } (x{:=}x+1;\ p{:=}p+m)\ \{p = n * m\}}}{\vdash \{n \geq 0\}\ p{:=}0;\ x{:=}0;\ \text{while } x<n \text{ do } (x{:=}x+1;\ p{:=}p+m)\ \{p = n * m\}}$$

Example: A Proof in Hoare Logic

Let’s just remember the open proof obligations!

$$
\dfrac{
\vdash \{n \geq 0\}\ p:=0;\ x:=0\ \{I\}
\qquad
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\vdash \{I[x+1/x,\ p+m/p]\}\ x:=x+1\ \{I[p+m/p]\}
\qquad
\vdash I \wedge x < n \Rightarrow I[x+1/x,\ p+m/p]
}{
\vdash \{I \wedge x<n\}\ x:=x+1\ \{I[p+m/p]\}
}
\quad
\vdash \{I[p+m/p\}\ p:=p+m\ \{I\}
}{
\vdash \{I \wedge x<n\}\ x:=x+1;\ p:=p+m\ \{I\}
}
}{
\vdash \{I\}\ \text{while } x < n \text{ do } (x:=x+1;\ p:=p+m)\ \{I \wedge x \geq n\}
}
\qquad
\vdash I \wedge x \geq n \Rightarrow p = n * m
}{
\vdash \{I\}\ \text{while } x < n \text{ do } (x:=x+1;\ p:=p+m)\ \{p = n * m\}
}
}{
\vdash \{n \geq 0\}\ p:=0;\ x:=0;\ \text{while } x < n \text{ do } (x:=x+1;\ p:=p+m)\ \{p = n * m\}
}
$$

Example: A Proof in Hoare Logic

Let's just remember the open proof obligations!

$$\vdash I \land x < n \Rightarrow I[x+1/x, p+m/p]$$

$$\vdash I \land x \geq n \Rightarrow p = n * m$$

Continue with the remaining part of the proof tree, as before.

$$\dfrac{\dfrac{\dfrac{\vdash n \geq 0 \Rightarrow I[0/p, 0/x] \qquad \vdash \{I[0/p, 0/x]\}\ p:=0\ \{I[0/x]\}}{\vdash \{n \geq 0\}\ p:=0\ \{I[0/x]\}} \qquad \vdash \{I[0/x]\}\ x:=0\ \{I\}}{\vdash \{n \geq 0\}\ p:=0;\ x:=0\ \{I\}} \qquad \dfrac{\vdots}{\vdash \{I\}\ \text{while } x < n \text{ do } (x:=x+1;\ p:=p+m)\ \{p = n * m\}}}{\vdash \{n \geq 0\}\ p:=0;\ x:=0;\ \text{while } x < n \text{ do } (x:=x+1;\ p:=p+m)\ \{p = n * m\}}$$

Now we only need to solve the remaining constraints!

Example: A Proof in Hoare Logic

Find $\mathtt{I}$ such that all constraints are simultaneously valid:

$\vdash n \geq 0 \Rightarrow \mathtt{I}[0/p, 0/x]$

$\vdash \mathtt{I} \wedge x < n \Rightarrow \mathtt{I}[x+1/x, p+m/p]$

$\vdash \mathtt{I} \wedge x \geq n \Rightarrow p = n * m$

$\mathtt{I} \equiv p = x * m \wedge x \leq n$

$\vdash n \geq 0 \Rightarrow 0 = 0 * m \wedge 0 \leq n$

$\vdash p = x * m \wedge x \leq n \wedge x < n \Rightarrow p+m = (x+1) * m \wedge x+1 \leq n$

$\vdash p = x * m \wedge x \leq n \wedge x \geq n \Rightarrow p = n * m$

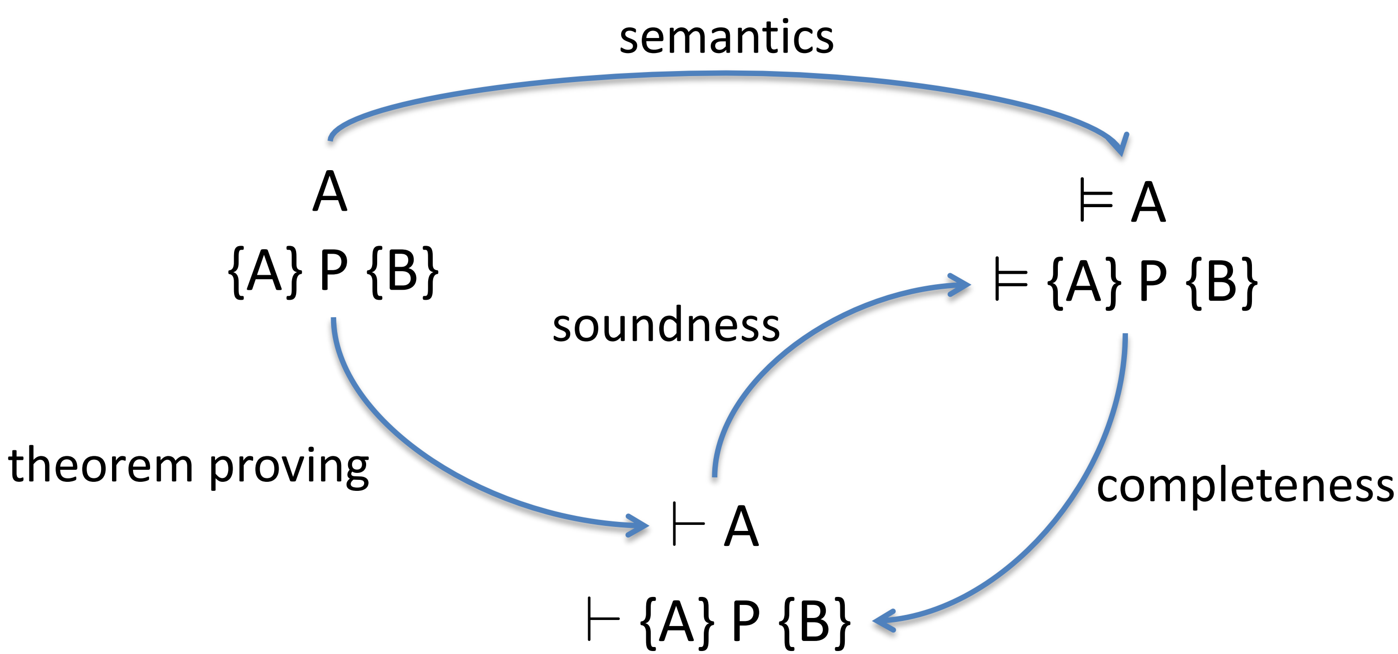
All constraints are valid!

Using Hoare Rules

- Hoare rules are mostly syntax directed
- There are three obstacles to automation of Hoare logic proofs:
  - When to apply the rule of consequence?
  - What invariant to use for `while`?
  - How do you prove the implications involved in the rule of consequence?
- The last one is how theorem proving gets in the picture
  - This turns out to be doable!
  - The loop invariants turn out to be the hardest problem!
  - Should the programmer give them?

Hoare Logic: Summary

- We have a language for asserting properties of programs.
- We know when such an assertion is true.
- We also have a symbolic method for deriving assertions.



Verification Conditions

- **Goal**: given a Hoare triple {A} P {B}, derive a single assertion *VC*(A,P,B) such that $\models$ *VC*(A,P,B) iff $\models$ {A} P {B}
- *VC*(A,P,B) is called verification condition.
- Verification condition generation factors out the hard work
  - Finding loop invariants
  - Finding function specifications
- Assume programs are annotated with such specifications
  - We will assume that the new form of the `while` construct includes an invariant:
    {`I`} `while` *b* do c
  - The invariant formula `I` must hold every time before *b* is evaluated.

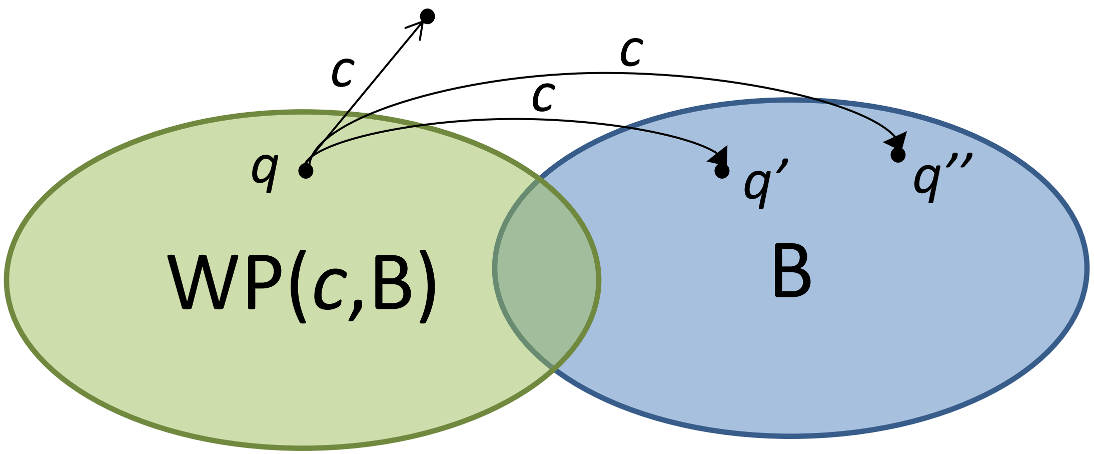
Verification Condition Generation

- Idea for VC generation: propagate the post-condition backwards through the program:
  - From {A} P {B}
  - generate $A \Rightarrow F(P, B)$
- This backwards propagation F(P, B) can be formalized in terms of weakest preconditions.

Weakest Preconditions

- The weakest precondition WP(c,B) holds for any state q whose c-successor states all satisfy B:

$$q \vDash \mathrm{WP}(c,\mathrm{B}) \quad \text{iff} \quad \forall q' \in Q.\ q \xrightarrow{c} q' \Rightarrow q' \vDash \mathrm{B}$$



- Compute WP(P,B) recursively according to the structure of the program P.

Loop-Free Guarded Commands

- Introduce loop-free guarded commands as an intermediate representation of the verification condition
- c ::= assume b
  | assert b
  | havoc x
  | c_1 ; c_2
  | $c_1 \square c_2$

From Programs to Guarded Commands

- GC(`skip`) =

  assume true

- GC(x := e) =

  assume $tmp = x$; havoc x; assume ($x = e[tmp/x]$)

  where tmp is fresh

- GC(c_1 ; c_2) =

  GC(c_1) ; GC(c_2)

- GC(`if` b `then` c_1 `else` c_2) =

- GC({`I`} `while` b do c) = ?

From Programs to Guarded Commands

- GC(`skip`) =

  assume true

- GC(x := e) =

  assume $tmp = x$; havoc x; assume ($x = e[tmp/x]$)

  where tmp is fresh

- GC(c_1 ; c_2) =

  GC(c_1) ; GC(c_2)

- GC(`if` b `then` c_1 `else` c_2) =

  (assume b; GC(c_1)) $\square$ (assume $\neg b$; GC(c_2))

- GC({`I`} `while` b do c) = ?

Guarded Commands for Loops

- GC({`I`} `while` *b* do *c*) =

  assert `I`;

  havoc x_1; …; havoc x_n;

  assume `I`;

  (assume *b*; GC(*c*); assert `I`; assume false) □

  assume $\neg b$

  where x_1, …, x_n are the variables modified in *c*

Computing Weakest Preconditions

- WP(assume b, B) =
- WP(assert b, B) =
- WP(havoc x, B) =
- WP(c_1;c_2, B) =
- WP($c_1 \square c_2$,B) =

Computing Weakest Preconditions

- WP(assume b, B) = $b \Rightarrow$ B
- WP(assert b, B) = $b \wedge$ B
- WP(havoc x, B) = B[a/x] (*a* fresh in B)
- WP(c_1;c_2, B) = WP(c_1, WP(c_2, B))
- WP(c_1 $\square$ c_2,B) = WP(c_1, B) $\wedge$ WP(c_2, B)

Putting Everything Together

- Given a Hoare triple $H \equiv \{A\}\ P\ \{B\}$
- Compute c_H = assume A; GC(P); assert B
- Compute $VC_H = WP(c_H, \text{true})$
- Infer $\vdash VC_H$ using a theorem prover.

Example: VC Generation

$\{n \geq 0\}$

$p := 0;$

$x := 0;$

$\{p = x * m \wedge x \leq n\}$

```
while x < n do
    x := x + 1;
    p := p + m
```

$\{p = n * m\}$

Example: VC Generation

- Computing the guarded command

assume n $\geq$ 0;

assume $p_0 = p$; havoc p; assume $p = 0$;

assume $x_0 = x$; havoc x; assume $x = 0$;

assert $p = x * m \wedge x \leq n$;

havoc x; havoc p; assume $p = x * m \wedge x \leq n$;

(assume x < n;

assume $x_1 = x$; havoc x; assume $x = x_1 + 1$;
assume $p_1 = p$; havoc p; assume $p = p_1 + m$;
assert $p = x * m \wedge x \leq n$; assume false)

 assume x $\geq$ n;

assert $p = n * m$

Example: VC Generation

- Computing the weakest precondition

WP (assume $n \geq 0$;

assume $p_0 = p$; havoc p; assume $p = 0$;

assume $x_0 = x$; havoc x; assume $x = 0$;

assert $p = x * m \wedge x \leq n$;

havoc x; havoc p; assume $p = x * m \wedge x \leq n$;

(assume $x < n$;

assume $x_1 = x$; *havoc* x; *assume* $x = x_1 + 1$;

assume $p_1 = p$; havoc p; assume $p = p_1 + m$;

assert $p = x * m \wedge x \leq n$; assert false)

$\square$ assume $x \geq n$,

assert $p = n * m$, true)

Example: VC Generation

- Computing the weakest precondition

$$n \geq 0 \wedge p_0 = p \wedge pa_3 = 0 \wedge x_0 = x \wedge xa_3 = 0 \Rightarrow$$

$$pa_3 = xa_3 * m \wedge xa_3 \leq n \wedge$$

$$(pa_2 = xa_2 * m \wedge xa_2 \leq n \Rightarrow$$

$$((xa_2 < n \wedge x_1 = xa_2 \wedge xa_1 = x_1 + 1 \wedge$$

$$p_1 = pa_2 \wedge pa_1 = p_1 + m) \Rightarrow pa_1 = xa_1 * m \wedge xa_1 \leq n)$$

$$\wedge (xa_2 \geq n \Rightarrow pa_2 = n * m))$$

Example: VC Generation

- The resulting VC is equivalent to the conjunction of the following implications

$$n \geq 0 \wedge p_0 = p \wedge pa_3 = 0 \wedge x_0 = x \wedge xa_3 = 0 \Rightarrow$$
$$pa_3 = xa_3 * m \wedge xa_3 \leq n$$

$$n \geq 0 \wedge p_0 = p \wedge pa_3 = 0 \wedge x_0 = x \wedge xa_3 = 0 \wedge pa_2 = xa_2 * m \wedge$$
$$xa_2 \leq n \Rightarrow$$
$$xa_2 \geq n \Rightarrow pa_2 = n * m$$

$$n \geq 0 \wedge p_0 = p \wedge pa_3 = 0 \wedge x_0 = x \wedge xa_3 = 0 \wedge pa_2 = xa_2 * m \wedge$$
$$xa_2 < n \wedge x_1 = xa_2 \wedge xa_1 = x_1 + 1 \wedge p_1 = pa_2 \wedge pa_1 = p_1 + m \Rightarrow$$
$$pa_1 = xa_1 * m \wedge xa_1 \leq n$$

Example: VC Generation

- simplifying the constraints yields

$$n \geq 0 \Rightarrow 0 = 0 * m \wedge 0 \leq n$$

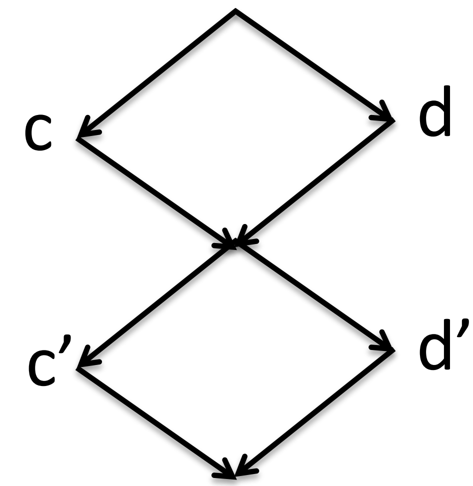
$$xa_2 \leq n \wedge xa_2 \geq n \Rightarrow xa_2 * m = n * m$$

$$xa_2 < n \Rightarrow xa_2 * m + m = (xa_2 + 1) * m \wedge xa_2 + 1 \leq n$$

- all of these implications are valid, which proves that the original Hoare triple was valid, too.

The Diamond Problem

assume A;
c □ *d*;
c' □ *d';*
assert B



$$A \Rightarrow \mathrm{WP}\,(c, \mathrm{WP}(c', B) \wedge \mathrm{WP}(d', B)) \wedge$$
$$\mathrm{WP}\,(d, \mathrm{WP}(c', B) \wedge \mathrm{WP}(d', B))$$

- Number of paths through the program can be exponential in the size of the program.
- Size of weakest precondition can be exponential in the size of the program.

Avoiding the Exponential Explosion

Defer the work of exploring all paths to the theorem prover:

- WP'(assume b, B, C) = ($b \Rightarrow$ B, C)
- WP'(assert b, B, C) = ($b \wedge$ B, C)
- WP'(havoc x, B, C) = (B[a/x], C)   (a fresh in B)
- WP'(c_1;c_2, B, C) =
  let F_2, C_2 = WP'(c_2, B, C) in WP'(c_1, F_2, C_2)
- WP'($c_1 \square c_2$,B, C) =
  let X = *fresh propositional variable* in
  let F_1, C_1 = WP'(c_1, X, true) and F_2, C_2 = WP'(c_2, X, true) in
  ($F_1 \wedge F_2$, $C \wedge C_1 \wedge C_2 \wedge$ (X $\Leftrightarrow$ B))

- WP(P, B) = let F, C = WP'(P, B, true) in $C \Rightarrow$ F

Translating Method Calls to GCs

```
/*@ requires P;
  @ assignable x₁, ..., xₙ;
  @ ensures Q; @*/
  T m (T₁ p₁, ..., Tₖ pₖ) { ... }
```

A method call

```
  y = x.m(y₁, ..., yₖ);
```

is desugared into the guarded command

assert P[x/this, y_1/p_1, ..., y_k/p_k];

havoc x_1; ..., havoc x_n; havoc y;

assume Q[x/this, y/\result]

Handling More Complex Program State

- When is the following Hoare triple valid?

  $$\{A\}\ x.f = 5\ \{x.f + y.f = 10\}$$

- A ought to imply "$y.f = 5 \vee x = y$"
- The IMP Hoare rule for assignment would give us:

  $$(x.f + y.f = 10)\ [5/x.f]$$
  $$\equiv 5 + y.f = 10$$
  $$\equiv y.f = 5 \text{ (we lost one case)}$$

- How come the rule does not work?

Modeling the Heap

- We cannot have side-effects in assertions
  - While generating the VC we must remove side-effects!
  - But how to do that when lacking precise aliasing information?
- Important technique: postpone alias analysis to the theorem prover
- Model the state of the heap as a symbolic mapping from addresses to values:
  - If e denotes an address and h a heap state then:
  - sel(h,e) denotes the contents of the memory cell
  - upd(h,e,v) denotes a new heap state obtained from h by writing v at address e

Heap Models

- We allow variables to range over heap states
  - So we can quantify over all possible heap states.
- Model 1
  - One “heap” for each object
  - One index constant for each field.
    We postulate f1 ≠ f2.
  - r.f1 is sel(r,f1) and r.f1 = e is r := upd(r,f1,e)
- Model 2 (Burnstall-Bornat)
  - One “heap” for each field
  - The object address is the index
  - r.f1 is sel(f1,r) and r.f1 = e is f1 := upd(f1,r,e)

Hoare Rule for Field Writes

- To model writes correctly, we use heap expressions
  - A field write changes the heap of that field

$$\{\ B[\mathrm{upd}(f, e_1, e_2)/f]\ \}\ e_1.f = e_2\ \{B\}$$

- Important technique:
  - model heap as a semantic object
  - defer reasoning about heap expressions to the theorem prover with inference rules such as (McCarthy):

$$\mathrm{sel}(\mathrm{upd}(h, e_1, e_2), e_3) = \begin{cases} e_2 \text{ if } e_1 = e_3 \\ \mathrm{sel}(h, e_3) \text{ if } e_1 \neq e_3 \end{cases}$$

Example: Hoare Rule for Field Writes

- Consider again: { A } x.f = 5 { x.f + y.f = 10 }
- We obtain:
  A $\equiv$ (x.f + y.f = 10)[upd(f, x, 5)/f]
  $\equiv$ (sel(f, x) + sel(f, y) = 10)[upd(f, x, 5)/f]
  $\equiv$ sel(upd(f x 5) x) + sel(upd(f x 5) y) = 10
  $\equiv$ 5 + sel(upd(f, x, 5), y) = 10
  $\equiv$ if x = y then 5 + 5 = 10 else 5 + sel(f, y) = 10
  $\equiv$ x = y $\vee$ y.f = 5
- Theorem generation.
- Theorem proving.

Modeling new Statements

- Introduce
  - a new predicate isAllocated(e, t) denoting that object e is allocated at allocation time t
  - and a new variable allocTime denoting the current allocation time.
- Add background axioms:

  $\forall$x t. isAllocated(x, t) $\Rightarrow$ isAllocated(x, t+1)
- Translate `new x.T()` to

  havoc x;
  assume $\neg$isAllocated(x, allocTime);
  assume Type(x) = T;
  assume x $\neq$ null;
  assume isAllocated(x, allocTime + 1);
  allocTime := allocTime + 1;
  Translation of call to constructor x.T()