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 \( q \overset{c}{\rightarrow} q' \)
  – 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:

\[ \models \{ A \} \ c \ \{ B \} \ \text{iff} \]

\[ \forall q \in Q. \ \forall q' \in Q. \ q \models A \land q \xrightarrow{c} q' \implies q' \models B \]

• and the meaning of a total correctness assertion:

\[ \models [A] \ c \ [B] \ \text{iff} \]

\[ \forall q \in Q. \ q \models A \implies \exists q' \in Q. \ q \xrightarrow{c} q' \land q' \models B \]

or even better:

\[ \forall q \in Q. \ \forall q' \in Q. \ q \models A \land q \xrightarrow{c} q' \implies q' \models B \land \]

\[ \forall q \in Q. \ q \models A \implies \exists q' \in Q. \ q \xrightarrow{c} q' \land q' \models B \]
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

$$
\vdash A' \Rightarrow A \quad \vdash \{A\} c \{B\} \quad \vdash B \Rightarrow B' \\
\vdash \{A'\} c \{B'\}
$$
Inference Rules for Hoare Logic

• One rule for each syntactic construct:

\[ \{A\} \text{skip} \{A\} \]
\[ \{\{A[e/x]\} x:=e \{A\} \]

\[ \begin{array}{c}
\{A\} c_1 \{B\} \quad \{B\} c_2 \{C\} \\
\hline
\{A\} c_1; c_2 \{C\}
\end{array} \]

\[ \begin{array}{c}
\{A \land b\} c_1 \{B\} \quad \{A \land \neg b\} c_2 \{B\} \\
\hline
\{A\} \text{if } b \text{ then } c_1 \text{ else } c_2 \{B\}
\end{array} \]

\[ \begin{array}{c}
\{I \land b\} c \{I\} \\
\hline
\{I\} \text{while } b \text{ do } c \{I \land \neg b\}
\end{array} \]
Example: A Proof in Hoare Logic

• We want to derive that

\{n \geq 0\}

\(p := 0;\)

\(x := 0;\)

\textbf{while} \ x < n \ \textbf{do}

\hspace{1em} x := x + 1;

\hspace{1em} p := p + m

\{p = n \ast m\}
Example: A Proof in Hoare Logic

Only applicable rule (except for rule of consequence):

\[
\text{\{A\} } c_1 \text{\{C\} } \vdash \text{\{C\} } c_2 \text{\{B\}} \\
\vdash \text{\{A\}} c_1; c_2 \text{\{B\}}
\]

\[
\vdash \{n \geq 0\} p:=0; x:=0 \text{\{C\}} \quad \vdash \text{\{C\} while } x < n \text{ do (}x:=x+1; p:=p+m) \text{ \{p = n * m\}}
\]

\[
\vdash \{n \geq 0\} p:=0; x:=0; \text{ while } x < n \text{ do (}x:=x+1; p:=p+m) \text{ \{p = n * m\}}
\]
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):

\[
\vdash \{ I \land b \} \ c \ \{ I \} \\
\vdash \{ I \} \text{ while } b \text{ do } c \ \{ I \land \neg b \}
\]

We can match \( \{ I \} \) with \( \{ C \} \) but we cannot match \( \{ I \land \neg b \} \) and \( \{ p = n \ast m \} \) directly. Need to apply the rule of consequence first!

\[
\vdash \{ n \geq 0 \} \ p:=0; \ x:=0 \ \{ C \} \\
\vdash \{ C \} \text{ while } x < n \text{ do } (x:=x+1; \ p:=p+m) \ \{ p = n \ast m \}
\]

\[
\vdash \{ n \geq 0 \} \ p:=0; \ x:=0; \text{ while } x < n \text{ do } (x:=x+1; \ p:=p+m) \ \{ p = n \ast m \}
\]

A \quad C_1 \quad C_2 \quad 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):

\[
\begin{align*}
\vdash & \{I \land b\} \ c \ \{I\} \\
\vdash & \{I\} \text{while} \ b \ \text{do} \ c \ \{I \land \neg b\} \\
\end{align*}
\]

I = A = A' = C

\[
\begin{align*}
\vdash & \{n \geq 0\} \ p:=0; \ x:=0 \ \{C\} \\
\vdash & \{C\} \text{while} \ x < n \ \text{do} \ (x:=x+1; \ p:=p+m) \ \{p = n \ast m\} \\
\end{align*}
\]
Example: A Proof in Hoare Logic

What is \( I \)? Let’s keep it as a placeholder for now!

Next applicable rule:

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

\[
\vdash \{I \land x < n\} x := x + 1; \quad p := p + m \quad \{I\}
\]

\[
\vdash \{I\} \text{ while } x < n \text{ do } (x := x + 1; \quad p := p + m) \quad \{I \land x \geq n\}
\]

\[
\vdash I \land x \geq n \Rightarrow p = n \ast m
\]

\[
\vdash \{n \geq 0\} p := 0; \quad x := 0 \quad \{I\}
\]

\[
\vdash \{I\} \text{ while } x < n \text{ do } (x := x + 1; \quad p := p + m) \quad \{p = n \ast m\}
\]

\[
\vdash \{n \geq 0\} p := 0; \quad x := 0; \quad \text{while } x < n \text{ do } (x := x + 1; \quad p := p + m) \quad \{p = n \ast m\}
\]
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):

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

\[
\begin{array}{c}
A \\
\{I \land x < n\} x := x + 1 \ {C}
\end{array} \quad \begin{array}{c}
c_1 \\
\{C\} p := p + m \ {I}
\end{array}
\]

\[
\begin{array}{c}
\{I \land x < n\} x := x + 1; p := p + m \ {I}
\end{array}
\]

\[
\vdash \{I\} \text{ while } x < n \text{ do } (x := x + 1; p := p + m) \ {I \land x \geq n}
\]

\[
\begin{array}{c}
\{n \geq 0\} p := 0; x := 0 \ {I}
\end{array} \quad \begin{array}{c}
\{I\} \text{ while } x < n \text{ do } (x := x + 1; p := p + m) \ {p = n \ast m}
\end{array}
\]

\[
\begin{array}{c}
\{n \geq 0\} p := 0; x := 0; \text{ while } x < n \text{ do } (x := x + 1; p := p + m) \ {p = n \ast m}
\end{array}
\]
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):

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

\[
\begin{align*}
\vdash \{I \land x \neq n\} \ x:=x+1 \ \{I[p+m/p]\} & \vdash \{I[p+m/p] \ p:=p+m \ \{I\} \\
\vdash \{I \land x \neq n\} \ x:=x+1; \ p:=p+m \ \{I\} & \vdash \{I\} \text{ while } x < n \text{ do } (x:=x+1; \ p:=p+m) \ \{I \land x \geq n\} \\
\vdash \{n \geq 0\} \ p:=0; \ x:=0 \ \{I\} & \vdash \{I\} \text{ while } x < n \text{ do } (x:=x+1; \ p:=p+m) \ \{p = n \ast m\} \\
\vdash \{n \geq 0\} \ p:=0; \ x:=0; \text{ while } x < n \text{ do } (x:=x+1; \ p:=p+m) \ \{p = n \ast m\}
\end{align*}
\]
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 \( \{I \land x < n\} \) and \( \{I[x+1/x, p+m/p]\} \)

\[ \vdash \{I \land x < n\} x:=x+1 \{I[p+m/p]\} \vdash \{I[p+m/p] \} p:=p+m \{I\} \]

\[ \vdash \{I \land x < n\} x:=x+1; p:=p+m \{I\} \]

\[ \vdash \{I\} \text{ while } x < n \text{ do } (x:=x+1; p:=p+m) \{I \land x \geq n\} \]

\[ \vdash I \land x \geq n \Rightarrow p = n \times m \]

\[ \vdash \{n \geq 0\} p:=0; x:=0 \{I\} \quad \vdash \{I\} \text{ while } x < n \text{ do } (x:=x+1; p:=p+m) \{p = n \times m\} \]

\[ \vdash \{n \geq 0\} p:=0; x:=0; \text{ while } x < n \text{ do } (x:=x+1; p:=p+m) \{p = n \times m\} \]
Example: A Proof in Hoare Logic

Let’s just remember the open proof obligations!

\[
\begin{align*}
\vdash & \{I[x+1/x, p+m/p]\} \ x:=x+1 \ \{I[p+m/p]\} \\
\vdash & \ I \land x < n \ \Rightarrow \ I[x+1/x, p+m/p] \\
\vdash & \{I \land x<n\} \ x:=x+1 \ \{I[p+m/p]\} \vdash \{I[p+m/p]\} \ p:=p+m \ \{I\} \\
\vdash & \{I \land x<n\} \ x:=x+1; \ p:=p+m \ \{I\} \\
\vdash & \{I\} \text{ while } x < n \text{ do } (x:=x+1; \ p:=p+m) \ \{I \land x \geq n\} \\
\vdash & \ I \land x \geq n \ \Rightarrow \ p = n \ast m \\
\vdash & \{n \geq 0\} \ p:=0; \ x:=0 \ \{I\} \\
\vdash & \{I\} \text{ while } x < n \text{ do } (x:=x+1; \ p:=p+m) \ \{p = n \ast m\} \\
\vdash & \{n \geq 0\} \ p:=0; \ x:=0; \text{ while } x < n \text{ do } (x:=x+1; \ p:=p+m) \ \{p = n \ast m\}
\end{align*}
\]
Example: A Proof in Hoare Logic

Let’s just remember the open proof obligations!

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

\[ \vdash I \land x \geq n \Rightarrow p = n \times m \]

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

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

Now we only need to solve the remaining constraints!

\[ \vdash \{n \geq 0\} p:=0; x:=0; \text{ while } x < n \text{ do (x:=x+1; p:=p+m) } \{p = n \times m\} \]
Example: A Proof in Hoare Logic

Find \( I \) such that all constraints are simultaneously valid:

\( \vdash n \geq 0 \Rightarrow I[0/p, 0/x] \)

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

\( \vdash I \land x \geq n \Rightarrow p = n \ast m \)

\( I \equiv p = x \ast m \land x \leq n \)

\( \vdash n \geq 0 \Rightarrow 0 = 0 \ast m \land 0 \leq n \)

\( \vdash p = x \ast m \land x \leq n \land x < n \Rightarrow p+m = (x+1) \ast m \land x+1 \leq n \)

\( \vdash p = x \ast m \land x \leq n \land x \geq n \Rightarrow p = n \ast m \)

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:
    \(\{I\}\text{while } b \text{ do } c\)
  - 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 $B$:

$$ q \models WP(c,B) \iff \forall q' \in Q. \ q \xrightarrow{c} q' \implies q' \models B $$

• 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

• $c ::= \text{assume } b$
  $\quad|\text{assert } b$
  $\quad|\text{havoc } x$
  $\quad|c_1 ; c_2$
  $\quad|c_1 \parallel c_2$
From Programs to Guarded Commands

- \( \text{GC}(\text{skip}) = \) 
  \[ \text{assume true} \]

- \( \text{GC}(x := e) = \) 
  \[ \text{assume } tmp = x; \text{ havoc } x; \text{ assume } (x = e[tmp/x]) \]

- \( \text{GC}(c_1 ; c_2) = \) 
  \[ \text{GC}(c_1); \text{GC}(c_2) \]

- \( \text{GC}(\text{if } b \text{ then } c_1 \text{ else } c_2) = \) 
  \[ \text{where } tmp \text{ is fresh} \]

- \( \text{GC}([I] \text{ while } b \text{ do } c) = ? \)
From Programs to Guarded Commands

- \( GC(\text{skip}) = \)
  
  assume true

- \( GC(x := e) = \)
  
  assume \( tmp = x \); havoc \( x \); assume \( (x = e[tmp/x]) \)

- \( GC(c_1 ; c_2) = \)
  
  \( GC(c_1) ; GC(c_2) \)  
  
  where \( tmp \) is fresh

- \( GC(\text{if} \ b \ \text{then} \ c_1 \ \text{else} \ c_2) = \)
  
  (assume \( b \); \( GC(c_1) \)) \( \sqcap \) (assume \( \neg b \); \( GC(c_2) \))

- \( GC(\{I\} \ \text{while} \ b \ \text{do} \ c) = ? \)
Guaranteed Commands for Loops

- \( \text{GC}(\{I\} \text{ while } b \text{ do } c) = \)
  
  \begin{align*}
  & \text{assert } I; \\
  & \text{havoc } x_1; \ldots; \text{havoc } x_n; \\
  & \text{assume } I; \\
  & (\text{assume } b; \ \text{GC}(c); \ \text{assert } I; \ \text{assume false}) \Box \\
  & \text{assume } \neg b
  \end{align*}

where \( x_1, \ldots, x_n \) are the variables modified in \( c \)
Computing Weakest Preconditions

- $\text{WP}(\text{assume } b, B) =$
- $\text{WP}(\text{assert } b, B) =$
- $\text{WP}(\text{havoc } x, B) =$
- $\text{WP}(c_1; c_2, B) =$
- $\text{WP}(c_1 \sqcap c_2, B) =$
Computing Weakest Preconditions

• \( WP(\text{assume } b, B) = b \Rightarrow B \)
• \( WP(\text{assert } b, B) = b \land B \)
• \( WP(\text{havoc } x, B) = B[a/x] \quad (a \text{ fresh in } B) \)
• \( WP(c_1 ; c_2, B) = WP(c_1, WP(c_2, B)) \)
• \( WP(c_1 \llcorner c_2, B) = WP(c_1, B) \land WP(c_2, B) \)
Putting Everything Together

• Given a Hoare triple $H \equiv \{A\} P \{B\}$

• Compute $c_H = \text{assume } A; \text{GC}(P); \text{assert } B$

• Compute $VC_H = WP(c_H, \text{true})$

• Infer $\vdash VC_H$ using a theorem prover.
Example: VC Generation

\{n \geq 0\}
\begin{align*}
p &:= 0; \\
x &:= 0; \\
\{p = x \cdot m \land x \leq n\}
\end{align*}

while \(x < n\) do
\begin{align*}
x &:= x + 1; \\
p &:= p + m
\end{align*}
\{p = n \cdot m\}
Example: VC Generation

• Computing the guarded command

\[
\begin{align*}
\text{assume } n &\geq 0; \\
\text{assume } p_0 &= p; \ \text{havoc } p; \ \text{assume } p = 0; \\
\text{assume } x_0 &= x; \ \text{havoc } x; \ \text{assume } x = 0; \\
\text{assert } p &= x \cdot m \land x \leq n; \\
\text{havoc } x; \ \text{havoc } p; \ \text{assume } p &= x \cdot m \land x \leq n; \\
\quad (\text{assume } x < n; \\
\quad \quad \text{assume } x_1 &= x; \ \text{havoc } x; \ \text{assume } x = x_1 + 1; \\
\quad \quad \text{assume } p_1 &= p; \ \text{havoc } p; \ \text{assume } p = p_1 + m; \\
\quad \quad \text{assert } p &= x \cdot m \land x \leq n; \ \text{assume false}) \\
\text{assume } x &\geq n; \\
\text{assert } p &= n \cdot m
\end{align*}
\]
Example: VC Generation

- Computing the weakest precondition

```plaintext
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 \land x \leq n;
    havoc x; havoc p; assume p = x * m \land 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 \land x \leq n; assert false)
  )
  assume x \geq n,
  assert p = n * m, true)
```
Example: VC Generation

- Computing the weakest precondition

\[ n \geq 0 \land p_0 = p \land pa_3 = 0 \land x_0 = x \land xa_3 = 0 \Rightarrow \]

\[ \begin{align*}
    pa_3 &= xa_3 \cdot m \land xa_3 \leq n \land \\
    (pa_2 &= xa_2 \cdot m \land xa_2 \leq n \Rightarrow \\
    ((xa_2 < n \land x_1 = xa_2 \land xa_1 = x_1 + 1 \land \\
    p_1 &= pa_2 \land pa_1 = p_1 + m) \Rightarrow pa_1 = xa_1 \cdot m \land xa_1 \leq n) \land \\
    (xa_2 \geq n \Rightarrow pa_2 = n \cdot m))
\end{align*} \]
Example: VC Generation

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

\[ n \geq 0 \land p_0 = p \land pa_3 = 0 \land x_0 = x \land xa_3 = 0 \implies pa_3 = xa_3 \cdot m \land xa_3 \leq n \]

\[ n \geq 0 \land p_0 = p \land pa_3 = 0 \land x_0 = x \land xa_3 = 0 \land pa_2 = xa_2 \cdot m \land xa_2 \leq n \implies xa_2 \geq n \implies pa_2 = n \cdot m \]

\[ n \geq 0 \land p_0 = p \land pa_3 = 0 \land x_0 = x \land xa_3 = 0 \land pa_2 = xa_2 \cdot m \land xa_2 < n \land x_1 = xa_2 \land xa_1 = x_1 + 1 \land p_1 = pa_2 \land pa_1 = p_1 + m \implies pa_1 = xa_1 \cdot m \land xa_1 \leq n \]
Example: VC Generation

• simplifying the constraints yields

\[ n \geq 0 \Rightarrow 0 = 0 \cdot m \land 0 \leq n \]

\[ xa_2 \leq n \land xa_2 \geq n \Rightarrow xa_2 \cdot m = n \cdot m \]

\[ xa_2 < n \Rightarrow xa_2 \cdot m + m = (xa_2 + 1) \cdot m \land xa_2 + 1 \leq n \]

• 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 \implies WP (c, WP(c’, B) \land WP(d’, B)) \land
WP (d, WP(c’, B) \land 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:

- \( WP'(\text{assume } b, B, C) = (b \Rightarrow B, C) \)
- \( WP'(\text{assert } b, B, C) = (b \land B, C) \)
- \( WP'(\text{havoc } x, B, C) = (B[a/x], C) \quad (a \text{ fresh in } B) \)
- \( WP'(c_1;c_2, B, C) = \) let \( F_2, C_2 = WP'(c_2, B, C) \) in \( WP'(c_1, F_2, C_2) \)
- \( WP'(c_1 \parallel c_2, B, C) = \) let \( X = \text{fresh propositional variable} \) in let \( F_1, C_1 = WP'(c_1, X, \text{true}) \) and \( F_2, C_2 = WP'(c_2, X, \text{true}) \) in \( (F_1 \land F_2, C \land C_1 \land C_2 \land (X \Leftrightarrow B)) \)
- \( WP(P, B) = \) let \( F, C = WP'(P, B, \text{true}) \) 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, y₁/p₁, ..., yₖ/pₖ];
havoc x₁; ..., havoc xₙ; 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 “\(y.\!f = 5 \lor x = y\)”

• 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 \( f_1 \neq f_2 \).
  – \( r.f_1 \) is \( \text{sel}(r,f_1) \) and \( r.f_1 = e \) is \( r := \text{upd}(r,f_1,e) \)
• Model 2 (Burnstall-Bornat)
  – One “heap” for each field
  – The object address is the index
  – \( r.f_1 \) is \( \text{sel}(f_1,r) \) and \( r.f_1 = e \) is \( f_1 := \text{upd}(f_1,r,e) \)
Hoare Rule for Field Writes

• To model writes correctly, we use heap expressions
  – A field write changes the heap of that field
    \[
    \{ B[\text{upd}(f, e_1, e_2)/f] \} \ e_1.f = e_2 \ {B}
    \]

• Important technique:
  – model heap as a semantic object
  – defer reasoning about heap expressions to the theorem prover with inference rules such as (McCarthy):
    \[
    \text{sel}(\text{upd}(h, e_1, e_2), e_3) =
    \begin{cases} 
    e_2 & \text{if } e_1 = e_3 \\
    \text{sel}(h, e_3) & \text{if } e_1 \neq e_3
    \end{cases}
    \]
Example: Hoare Rule for Field Writes

• Consider again: \{ A \} x.f = 5 \{ x.f + y.f = 10 \}

• We obtain:
\[
A \equiv (x.f + y.f = 10)[\text{upd}(f, x, 5)/f]
\equiv (\text{sel}(f, x) + \text{sel}(f, y) = 10)[\text{upd}(f, x, 5)/f]
\equiv \text{sel}(\text{upd}(f x 5) x) + \text{sel}(\text{upd}(f x 5) y) = 10
\equiv 5 + \text{sel}(\text{upd}(f, x, 5), y) = 10
\equiv \text{if } x = y \text{ then } 5 + 5 = 10 \text{ else } 5 + \text{sel}(f, y) = 10
\equiv x = y \lor y.f = 5
\]

• 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:
  \( \forall x \ t. \text{isAllocated}(x, t) \implies \text{isAllocated}(x, t+1) \)

• Translate \texttt{new x.T()} to
  havoc x;
  assume \( \neg \text{isAllocated}(x, \text{allocTime}) \);
  assume Type(x) = T;
  assume x \neq \text{null};
  assume isAllocated(x, allocTime + 1);
  allocTime := allocTime + 1;
  **Translation of call to constructor \texttt{x.T()}**