CS477 Formal Software Dev Methods

Elsa L Gunter
2112 SC, UIUC
egunter@illinois.edu
http://courses.engr.illinois.edu/cs477

Slides based in part on previous lectures
by Mahesh Vishwanathan, and by Gul Agha

March 16, 2018

Algortihm for Proving Hoare Triples?

- Have seen in Isabelle that much of proving a Hoare triple is routine
- Will this always work?
- Why not automate the whole process?
  - Can't (always) calculate needed loop invariants
  - Can't (always) prove implications (side-conditions) in Rule of Consequence application
- Can we automate all but this?
- Yes! But how?

  1. Annotate all `while` loops with needed `invariants`

  2. Use routine to "roll back" post-condition to weakest precondition, gathering side-conditions as we go

- 2 called verification condition generation

Annotated Simple Imperative Language

- Give verification conditions for an annotated version of our simple imperative language
- Add a presumed invariant to each while loop

$$
\begin{array}{rcl}
\langle \textit{command} \rangle & ::= & \langle \textit{variable} \rangle := \langle \textit{term} \rangle \\
 & | & \langle \textit{command} \rangle;\ \ldots;\ \langle \textit{command} \rangle \\
 & | & \textit{if } \langle \textit{statement} \rangle \textit{ then } \langle \textit{command} \rangle \textit{ else } \langle \textit{command} \rangle \\
 & | & \textit{while } \langle \textit{statement} \rangle \textit{ inv } \langle \textit{statement} \rangle \textit{ do } \langle \textit{command} \rangle
\end{array}
$$

Example:

```
while y < n inv x = y * y
do
 x := (2 * y) + 1;
 y := y + 1
od
```

Hoare Logic for Annotated Programs

Assingment Rule

$$\frac{}{\{\!|P[e/x]|\!\}\ x\ :=\ e\ \{\!|P|\!\}}$$

Rule of Consequence

$$\frac{P \Rightarrow P' \quad \{\!|P'|\!\}\ C\ \{\!|Q'|\!\} \quad Q' \Rightarrow Q}{\{\!|P|\!\}\ C\ \{\!|Q|\!\}}$$

Sequencing Rule

$$\frac{\{\!|P|\!\}\ C_1\ \{\!|Q|\!\} \quad \{\!|Q|\!\}\ C_2\ \{\!|R|\!\}}{\{\!|P|\!\}\ C_1;\ C_2\ \{\!|R|\!\}}$$

If Then Else Rule

$$\frac{\{\!|P \wedge B|\!\}\ C_1\ \{\!|Q|\!\} \quad \{\!|P \wedge \neg B|\!\}\ C_2\ \{\!|Q|\!\}}{\{\!|P|\!\}\ if\ \ B\ \ then\ \ C_1\ \ else\ \ C - 2\ \{\!|Q|\!\}}$$

While Rule

$$\frac{\{\!|P \wedge B|\!\}\ C\ \{\!|P|\!\}}{\{\!|P|\!\}\ while\ \ B\ \ inv\ \ P\ \ do\ \ C\ \{\!|P \wedge \neg B|\!\}}$$

Relation Between Two Languages

- Hoare Logic for Simple Imperative Programs and Hoare Logic for Annotated Programs almost the same
- What it precise relationship?
- First need precise relation between the two languages

Definition

$\mathrm{strip}(v := e) \;=\; v := e$

$\mathrm{strip}(C_1 \,;\, C_2) \;=\; \mathrm{strip}(C_1) \,;\, \mathrm{strip}(C_2)$

$\mathrm{strip}(\texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2 \texttt{ fi}) \;=$

$\quad \texttt{if } B \texttt{ then } \mathrm{strip}(C_1) \texttt{ else } \mathrm{strip}(C_2) \texttt{ fi}$

$\mathrm{strip}(\texttt{while } B \texttt{ inv } P \texttt{ do } C \texttt{ od}) \;=\; \texttt{while } B \texttt{ do } \mathrm{strip}(C) \texttt{ od}$

- We recursively remove all invariant annotations from all `while` loops

Relation Between Two Hoare Logics

Theorem

For all pre- and post-conditions P and Q, and annotated programs C, if $\{\!|P|\!\}\ C\ \{\!|Q|\!\}$, then $\{P\}\ strip(C)\ \{Q\}$.

Proof.

(Sketch) Use rule induction on proof of $\{\!|P|\!\}\ C\ \{\!|Q|\!\}$; in case of While Rule, erase invariant from program $\square$

Relation Between Two Hoare Logics

Theorem

For all pre- and post-conditions P and Q, and unannotated programs C, if $\{P\}\ C\ \{Q\}$, then there exists an annotated program S such that $C = strip(S)$ and $\{\!|P|\!\}\ S\ \{\!|Q|\!\}$.

Proof.

(Sketch) Use rule induction on proof of $\{P\}\ C\ \{Q\}$; in case of While Rule, add invariant from precondition as invariant to command. $\square$

Weakest Precondition

Question: Given post-condition Q, and annotated program C, what is the most general pre-condition P such that $\{\!|P|\!\}\ C\ \{\!|Q|\!\}$?

Answer: Weakest Precondition

Definition

$$
\begin{aligned}
&\mathrm{wp}\ (x := e)\ Q = Q[x \Leftarrow e]\\
&\mathrm{wp}\ (C_1; C_2)\ Q = \mathrm{wp}\ C_1\ (\mathrm{wp}\ C_2\ Q)\\
&\mathrm{wp}\ (\texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2\ \texttt{fi})\ Q =\\
&\quad (B \wedge (\mathrm{wp}\ C_1\ Q)) \vee ((\neg B) \wedge (\mathrm{wp}\ C_2\ Q))\\
&\mathrm{wp}\ (\texttt{while}\ B\ \texttt{inv}\ P\ \texttt{do}\ C\ \texttt{od})\ Q = P
\end{aligned}
$$

Assumes, without verifying, that P is the correct invariant

Weakest Justification

Weakest in weakest precondition means any other valid precondition implies it:

Theorem

For all annotated programs C, and pre- and post-conditions P and Q, if $\{\!|P|\!\}\ C\ \{\!|Q|\!\}$ then $P \Rightarrow wp\ C\ Q$.

- Proof somewhat complicated
- Uses induction on the structure of C
- In each case, want to assert triple proof must have used rule for that construct (e.g. Sequence Rule for sequences)
- Can't because of Rule Of Consequence
- Must induct on proof (rule induction) - in each case
- Uses:

Lemma

$\forall C\ P\ Q.\ (P \Rightarrow Q) \Rightarrow (wp\ C\ P \Rightarrow wp\ C\ Q)$

What About Precondition?

Question: Do we have $\{\!|\text{wp } C\ Q|\!\}\ C\ \{\!|Q|\!\}$?

What About Precondition?

Question: Do we have $\{\!|\mathrm{wp}\ C\ Q|\!\}\ C\ \{\!|Q|\!\}$?

Answer: Not always - need to check while-loop side-conditions – verification conditions

What About Precondition?

Question: Do we have $\{\!| \mathrm{wp}\ C\ Q |\!\}\ C\ \{\!| Q |\!\}$?

Answer: Not always - need to check while-loop side-conditions – verification conditions

Question: How to calculate verification conditions?

What About Precondition?

Question: Do we have $\{\!|\text{wp } C\ Q|\!\}\ C\ \{\!|Q|\!\}$?

Answer: Not always - need to check while-loop side-conditions – verification conditions

Question: How to calculate verification conditions?

Definition

$\text{vcg }(x := e)\ Q = \mathrm{true}$

What About Precondition?

Question: Do we have $\{\!|\text{wp } C\ Q|\!\}\ C\ \{\!|Q|\!\}$?

Answer: Not always - need to check while-loop side-conditions – verification conditions

Question: How to calculate verification conditions?

Definition

$\text{vcg } (x := e)\ Q = \mathrm{true}$
$\text{vcg } (C_1; C_2)\ Q = (\text{vcg } C_1\ (\text{wp } C_2\ Q)) \wedge (\text{vcg } C_2\ Q)$

What About Precondition?

Question: Do we have $\{\!|\text{wp } C\ Q|\!\}\ C\ \{\!|Q|\!\}$?

Answer: Not always - need to check while-loop side-conditions – verification conditions

Question: How to calculate verification conditions?

Definition

$\text{vcg } (x := e)\ Q = \mathrm{true}$
$\text{vcg } (C_1; C_2)\ Q = (\text{vcg } C_1\ (\text{wp } C_2\ Q)) \wedge (\text{vcg } C_2\ Q)$
$\text{vcg } (\texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2 \texttt{ fi})\ Q = (\text{vcg } C_1\ Q) \wedge (\text{vcg } C_2\ Q)$

What About Precondition?

Question: Do we have $\{\!|\text{wp } C\ Q|\!\}\ C\ \{\!|Q|\!\}$?

Answer: Not always - need to check while-loop side-conditions – verification conditions

Question: How to calculate verification conditions?

Definition

$\text{vcg } (x := e)\ Q = \text{true}$
$\text{vcg } (C_1; C_2)\ Q = (\text{vcg } C_1\ (\text{wp } C_2\ Q)) \wedge (\text{vcg } C_2\ Q)$
$\text{vcg } (\texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2 \texttt{ fi})\ Q = (\text{vcg } C_1\ Q) \wedge (\text{vcg } C_2\ Q)$
$\text{vcg } (\texttt{while } B \texttt{ inv } P \texttt{ do } C \texttt{ od})\ Q =$
$\quad ((P \wedge B) \Rightarrow (\text{wp } C\ P)) \wedge (\text{vcg } C\ P) \wedge ((P \wedge (\neg B)) \Rightarrow Q)$

Verification Condition Guarantees wp Precondition

Theorem

$vcg\ C\ Q \Rightarrow \{\!|wp\ C\ Q|\!\}\ C\ \{\!|Q|\!\}$

Proof.

(Sketch)

- Induct on structure of C
- For each case, wind back as we did in specific examples:
  - Assignment: wp C Q exactly what is needed for Assignment Axiom
  - Sequence: Follows from inductive hypotheses, all elim, and modus ponens
  - If_Then_Else: Need to use Precondition Strengthening with each branch of conditional; wp and inductive hypotheses give the needed side conditions
  - While: Need to use Postcondition Weakening, While Rule and Precondition Strengthening

□

Verification Condition Guarantees wp Precondition

Corollary

$((P \Rightarrow wp\ C\ Q) \wedge (vcg\ C\ Q)) \Rightarrow \{\!|P|\!\}\ C\ \{\!|Q|\!\}$

This amounts to a method for proving Hoare triple $\{P\}\ C\ \{Q\}$:

1. Annotate program with loop invariants
2. Calculate wp $C\ Q$ and vcg $C\ Q$ (automated)
3. Prove $P \Rightarrow$ wp $C\ Q$ and vcg $C\ Q$

Basic outline of interaction with Boogie: Human does 1, Boogie does 2, Z3 / Simplify / Isabelle + human / ... does 3

For more infomation

- http://research.microsoft.com/en-us/projects/boogie/
- http://research.microsoft.com/en-us/um/people/moskal/pdf/hol-boogie.pdf
- http://www.cl.cam.ac.uk/research/hvg/Isabelle/dist/library/HOL/HOL-Hoare/index.html

Model For Hoare Logic

- Seen proof system for Hoare Logic
- What about models?
- Informally, triple modeled by
  - pairs of assignments of program variables to values
  - where executing program starting with initial assignment results in a memory that gives the final assignment
- Calls for alternate definition of execution

Natural Semantics

- Aka Structural Operational Semantics, aka "Big Step Semantics"
- Provide value for a program by rules and derivations, similar to type derivations
- Rule conclusions look like

$$(C, m) \Downarrow m'$$

or

$$(E, m) \Downarrow v$$

Simple Imperative Programming Language

- $I \in Identifiers$
- $N \in Numerals$
- B ::= true | false | B & B | B or B | not B
  | $E < E$ | $E = E$
- $E ::= N \mid I \mid E + E \mid E * E \mid E - E \mid - E$
- C::= skip | $C;C$ | I ::= E
  | if B then C else C fi | while B do C od

Natural Semantics of Atomic Expressions

- Identifiers: $(I,m) \Downarrow m(I)$
- Numerals are values: $(N,m) \Downarrow N$
- Booleans: $(\text{true},m) \Downarrow \text{true}$

  $(\text{false},m) \Downarrow \text{false}$

Booleans:

$$\frac{(B, m) \Downarrow \text{false}}{(B \,\&\, B', m) \Downarrow \text{false}} \qquad \frac{(B, m) \Downarrow \text{true} \quad (B', m) \Downarrow b}{(B \,\&\, B', m) \Downarrow b}$$

$$\frac{(B, m) \Downarrow \text{true}}{(B \text{ or } B', m) \Downarrow \text{true}} \qquad \frac{(B, m) \Downarrow \text{false} \quad (B', m) \Downarrow b}{(B \text{ or } B', m) \Downarrow b}$$

$$\frac{(B, m) \Downarrow \text{true}}{(\text{not } B, m) \Downarrow \text{false}} \qquad \frac{(B, m) \Downarrow \text{false}}{(\text{not } B, m) \Downarrow \text{true}}$$

Relations

$$\frac{(E, m) \Downarrow U \qquad (E', m) \Downarrow V \quad U \sim V = b}{(E \sim E', m) \Downarrow b}$$

- By $U \sim V = b$, we mean does (the meaning of) the relation $\sim$ hold on the meaning of U and V
- May be specified by a mathematical expression/equation or rules matching U and V

Arithmetic Expressions

$$\frac{(E, m) \Downarrow U \quad (E', m) \Downarrow V \quad U \; op \; V = N}{(E \; op \; E', m) \Downarrow N}$$

where *N* is the specified value for *U op V*

Commands

Skip: $(\text{skip}, m) \Downarrow m$

Assignment:
$$\frac{(E,m) \Downarrow V}{(I::=E,m) \Downarrow m[I \text{ <-- } V\,]}$$

Sequencing:
$$\frac{(C,m) \Downarrow m' \quad (C',m') \Downarrow m''}{(C;C',\ m) \Downarrow m''}$$

If Then Else Command

$$\frac{(B,m) \Downarrow \text{true} \quad (C,m) \Downarrow m'}{(\text{if } B \text{ then } C \text{ else } C' \text{ fi}, m) \Downarrow m'}$$

$$\frac{(B,m) \Downarrow \text{false} \quad (C',m) \Downarrow m'}{(\text{if } B \text{ then } C \text{ else } C' \text{ fi}, m) \Downarrow m'}$$

While Command

$$\frac{(B,m) \Downarrow \text{false}}{(\text{while } B \text{ do } C \text{ od}, m) \Downarrow m}$$

$$\frac{(B,m)\Downarrow\text{true} \quad (C,m)\Downarrow m' \quad (\text{while } B \text{ do } C \text{ od}, m')\Downarrow m''}{(\text{while } B \text{ do C od}, m) \Downarrow m''}$$

Example: If Then Else Rule

$$\frac{}{\text{(if } x > 5 \text{ then } y{:=}\ 2 + 3 \text{ else } y{:=}3 + 4 \text{ fi,} \quad \{x \text{ -> } 7\}) \Downarrow ?}$$

Example: If Then Else Rule

$$\frac{\overline{(x > 5, \{x \to 7\}) \Downarrow ?}}{(\text{if } x > 5 \text{ then } y{:=}\ 2 + 3 \text{ else } y{:=}3 + 4 \text{ fi}, \{x \to 7\}) \Downarrow \ ?}$$

Example: Arith Relation

$$\cfrac{\cfrac{? > ? = ?}{(x,\{x\text{->}7\})\Downarrow ? \quad (5,\{x\text{->}7\})\Downarrow ?}}{\cfrac{(x > 5, \{x \text{->} 7\})\Downarrow ?}{(\text{if } x > 5 \text{ then } y{:=}\ 2 + 3 \text{ else } y{:=}3 + 4 \text{ fi}, \{x \text{->} 7\}) \Downarrow ?}}$$

Example: Identifier(s)

$$\dfrac{\dfrac{7 > 5 = \text{true} \qquad (x,\{x\text{->}7\})\Downarrow 7 \quad (5,\{x\text{->}7\})\Downarrow 5}{(x > 5,\ \{x \text{ -> } 7\})\Downarrow ?}}{(\text{if } x > 5 \text{ then } y{:=}\ 2 + 3 \text{ else } y{:=}3 + 4 \text{ fi},\ \{x \text{ -> } 7\}) \Downarrow ?}$$

Example: Arith Relation

$$\dfrac{\dfrac{7 > 5 = \text{true}}{(x,\{x\text{->}7\})\Downarrow 7 \quad (5,\{x\text{->}7\})\Downarrow 5}}{(x > 5,\ \{x \text{ -> } 7\})\Downarrow \text{true}}$$

$$\overline{\text{(if } x > 5 \text{ then } y{:=}\ 2 + 3 \text{ else } y{:=}3 + 4 \text{ fi},}$$

$$\{x \text{ -> } 7\}) \Downarrow\ ?$$

Example: If Then Else Rule

$$\dfrac{\dfrac{7 > 5 = \text{true}}{(x,\{x\text{->}7\})\Downarrow 7 \quad (5,\{x\text{->}7\})\Downarrow 5}}{(x > 5, \{x \text{ -> } 7\})\Downarrow \text{true}} \qquad \dfrac{}{(y{:=}\ 2 + 3, \{x\text{-> } 7\} \Downarrow ?)}$$

$$\overline{(\text{if } x > 5 \text{ then } y{:=}\ 2 + 3 \text{ else } y{:=}3 + 4 \text{ fi}, \{x \text{ -> } 7\}) \Downarrow ?}$$

Example: Assignment

$$\dfrac{\dfrac{(x,\{x\text{->}7\})\Downarrow 7 \quad (5,\{x\text{->}7\})\Downarrow 5 \quad 7 > 5 = \text{true}}{(x > 5,\ \{x \text{->} 7\})\Downarrow \text{true}} \qquad \dfrac{\dfrac{\overline{}}{(2+3,\ \{x\text{->}7\})\Downarrow ?}}{(y{:=}\ 2 + 3,\ \{x\text{->}\ 7\})\Downarrow ?}}{(\text{if } x > 5 \text{ then } y{:=}\ 2 + 3 \text{ else } y{:=}3 + 4 \text{ fi},\ \{x \text{->} 7\}) \Downarrow ?}$$

Example: Arith Op

$$
\dfrac{\dfrac{7 > 5 = \text{true}}{(x,\{x\text{->}7\})\Downarrow 7 \quad (5,\{x\text{->}7\})\Downarrow 5}{(x > 5,\ \{x \text{->} 7\})\Downarrow \text{true}} \qquad \dfrac{\dfrac{\dfrac{? + ? = ?}{(2,\{x\text{->}7\})\Downarrow ? \quad (3,\{x\text{->}7\})\Downarrow ?}}{(2+3,\ \{x\text{->}7\})\Downarrow ?}}{(y{:=}\ 2 + 3,\ \{x\text{->}\ 7\}\ \Downarrow ? \qquad .}}{(\text{if } x > 5 \text{ then } y{:=}\ 2 + 3 \text{ else } y{:=}3 + 4 \text{ fi},\ \{x \text{->} 7\}) \Downarrow ?}
$$

Example: Numerals

$$
\dfrac{
\dfrac{(x,\{x\text{->}7\})\Downarrow 7 \quad (5,\{x\text{->}7\})\Downarrow 5 \quad 7 > 5 = \text{true}}{(x > 5,\ \{x \text{->}\ 7\})\Downarrow \text{true}}
\qquad
\dfrac{\dfrac{\dfrac{(2,\{x\text{->}7\})\Downarrow 2 \quad (3,\{x\text{->}7\})\Downarrow 3 \quad 2 + 3 = 5}{(2+3,\ \{x\text{->}7\})\Downarrow ?}}{(y{:=}\ 2 + 3,\ \{x\text{->}\ 7\}}}{\Downarrow ?}
}{
(\text{if } x > 5 \text{ then } y{:=}\ 2 + 3 \text{ else } y{:=}3 + 4 \text{ fi},\ \{x \text{->}\ 7\}) \Downarrow ?
}
$$

Example: Arith Op

$$\dfrac{\dfrac{7 > 5 = \text{true} \quad \dfrac{(x,\{x\text{->}7\})\Downarrow 7 \quad (5,\{x\text{->}7\})\Downarrow 5}{}}{(x > 5,\ \{x \text{->}\ 7\})\Downarrow \text{true}} \qquad \dfrac{\dfrac{2 + 3 = 5 \quad \dfrac{(2,\{x\text{->}7\})\Downarrow 2 \quad (3,\{x\text{->}7\})\Downarrow 3}{}}{(2+3,\ \{x\text{->}7\})\Downarrow 5}}{(y{:=}\ 2 + 3,\ \{x\text{->}\ 7\}\ \Downarrow ?}}{(\text{if } x > 5 \text{ then } y{:=}\ 2 + 3 \text{ else } y{:=}3 + 4 \text{ fi},\ \{x \text{->}\ 7\})\ \Downarrow\ ?}$$

Example: Assignment

$2 + 3 = 5$

$(2,\{x\text{->}7\})\Downarrow 2 \quad (3,\{x\text{->}7\})\Downarrow 3$

$7 > 5 = \text{true}$ $\quad$ $(2+3, \{x\text{->}7\})\Downarrow 5$

$(x,\{x\text{->}7\})\Downarrow 7 \quad (5,\{x\text{->}7\})\Downarrow 5$ $\quad$ $(y{:}= 2 + 3, \{x\text{->} 7\}$

$(x > 5, \{x \text{->} 7\})\Downarrow \text{true}$ $\quad$ $\Downarrow \{x\text{->}7, y\text{->}5\}$

(if x > 5 then y:= 2 + 3 else y:=3 + 4 fi,

$\{x \text{->} 7\}) \Downarrow ?$

Example: If Then Else Rule

$$
\dfrac{
\dfrac{7 > 5 = \text{true} \qquad (x,\{x\text{->}7\})\Downarrow 7 \quad (5,\{x\text{->}7\})\Downarrow 5}{(x > 5,\ \{x \text{->} 7\})\Downarrow \text{true}}
\qquad
\dfrac{\dfrac{2 + 3 = 5 \qquad (2,\{x\text{->}7\})\Downarrow 2 \quad (3,\{x\text{->}7\})\Downarrow 3}{(2+3,\ \{x\text{->}7\})\Downarrow 5}}{(y{:=}\ 2 + 3,\ \{x\text{->}\ 7\}) \Downarrow \{x\text{->}7,\ y\text{->}5\}}
}{
(\text{if } x > 5 \text{ then } y{:=}\ 2 + 3 \text{ else } y{:=}3 + 4 \text{ fi},\ \{x \text{->} 7\}) \Downarrow \{x\text{->}7,\ y\text{->}5\}
}
$$

Natural Semantics Models Hoare Logic (Soundness)

Definition

Say a pair of states (aka assignments) (m_1, m_2) satsifies, or models the Hoare triple $\{P\}\ C\ \{Q\}$ if $m_1 \models P$ and $m_2 \models Q$. Write $(m_1, m_2) \models \{P\}\ C\ \{Q\}$

Theorem

Let $\{P\}\ C\ \{Q\}$ be a valid Hoare triple (i.e., it's provable). Let m_1 be a state (aka assignment) such that $m_1 \models P$. Let m_2 be a state such that $(C, m_1) \Downarrow m_2$. Then $(m_1, m_2) \models \{P\}\ C\ \{Q\}$

Natural Semantics Models Hoare Logic (Completeness)

Theorem

Let $\{P\}\ C\ \{Q\}$ *be a such that for all* $m1$ *and* $m2$, *if (we can prove)* $m_1 \models P$ *and* $(C, m_1) \Downarrow m_2$ *then (we can prove)* $m_2 \models Q$. *Then* $\{P\}\ C\ \{Q\}$ *is provable in Hoare logic.*

Simple Imperative Programming Language #2

$$
\begin{array}{lcl}
I & \in & \mathit{Identifiers} \\
N & \in & \mathit{Numerals} \\
E & ::= & N \mid I \mid E + E \mid E * E \mid E - E \mid I ::= E \\
B & ::= & \mathrm{true} \mid \mathrm{false} \mid B \& B \mid B \ \mathrm{or}\ B \mid \mathrm{not}\ B \\
 & & \mid E < E \mid E = E \\
C & ::= & \mathrm{skip} \mid C; C \mid \{C\} \mid E \\
 & & \mid \mathrm{if}\ B\ \mathrm{then}\ C\ \mathrm{else}\ C\ \mathrm{fi} \\
 & & \mid \mathrm{while}\ B\ \mathrm{do}\ C
\end{array}
$$

Changes for Expressions

- Need new type of *result* for expressions

$$(E, m) \Downarrow (v, m')$$

- Modify old rules for expressions:

Atomic Expressions:

$$(I, m) \Downarrow (m(I), m) \quad (N, m) \Downarrow (N, m)$$

Binary Operators:

$$\frac{(E, m) \Downarrow (U, m') \quad (E', m') \Downarrow (V, m'') \quad U \oplus V = N}{(E \oplus E', m) \Downarrow (N, m'')}$$

New Rule for Expressions

$$\frac{(E, m) \Downarrow (V, m')}{(I ::= E, m) \Downarrow (V, m'[I \leftarrow V])}$$

Changes for Commands

- Replace rule for Assignment by one for Expressions as Commands:

$$\frac{(E, m) \Downarrow (v, m')}{(E, m) \Downarrow m'}$$

- Unfortunately, can't stop there
  - Relations use Expressions; must be changed
  - Relations produce Booleans; all Booleans must be changed
  - `if_then_else` and `while` use Booleans; must be changed

Relations

- Must thread state through the relations:

$$\frac{(E, m) \Downarrow (U, m') \quad (E', m') \Downarrow (V, m'') \quad U \sim V = b}{(E \sim E', m) \Downarrow (b, m'')}$$

Changes for Boolean Expressions

- Arithmetic Expressions occur in Boolean Expresion; must change type of result for Boolens:

$$(B, m) \Downarrow (b, m')$$

- Modify old rules for Booleans to reflect new type:

  Atomic Booleans:

$$(\mathsf{true}, m) \Downarrow (\mathsf{true}, m)$$

$$(\mathsf{false}, m) \Downarrow (\mathsf{false}, m)$$

Changes for Boolean Expressions

$$\frac{(B, m) \Downarrow (\text{false}, m')}{(B \& B', m) \Downarrow (\text{false}, m')} \qquad \frac{(B, m) \Downarrow (\text{true}, m') \quad (B', m') \Downarrow (b, m'')}{(B \& B', m) \Downarrow (b, m'')}$$

$$\frac{(B, m) \Downarrow (\text{true}, m')}{(B \text{ or } B', m) \Downarrow (\text{true}, m')} \qquad \frac{(B, m) \Downarrow (\text{false}, m') \quad (B', m') \Downarrow (b, m'')}{(B \text{ or } B', m) \Downarrow (b, m'')}$$

$$\frac{(B, m) \Downarrow (\text{true}, m')}{(\text{not } B, m) \Downarrow (\text{false}, m')} \qquad \frac{(B, m) \Downarrow (\text{false}, m')}{(\text{not } B, m) \Downarrow (\text{true}, m')}$$

Revised if_then_else Rule

$$\frac{(B, m) \Downarrow (\text{true}, m') \quad (C, m') \Downarrow m''}{(\text{if } B \text{ then } C \text{ else } C' \text{ fi}, m) \Downarrow m''}$$

$$\frac{(B, m) \Downarrow (\text{false}, m' \quad (C', m') \Downarrow m''}{(\text{if } B \text{ then } C \text{ else } C' \text{ fi}, m) \Downarrow m''}$$

Revised while Rule

$$\frac{(B, m) \Downarrow (\text{false}, m')}{(\text{while } B \text{ do } C, m) \Downarrow m'}$$

$$\frac{(B, m) \Downarrow (\text{true}, m') \quad (C, m') \Downarrow m'' \quad (\text{while } B \text{ do } C, m'') \Downarrow m'''}{(\text{while } B \text{ do } C, m) \Downarrow m'''}$$

Termination and Errors in SOS

- (C,m), (E,m), (B,m) called configurations
- A configuration c evaluates to a result r if $c \Downarrow r$.
- If a configuration c evaluates to a result r, then c terminates without error
- Problem: Can not distinguish between untermination (*e.g.* a while loop that runs forever), versus and error (*e.g.* referencing an unassigned value
- Can be (partially) remedied by adding error result
  - Roughly doubles number of rules

Transition Semantics

- Aka "small step structured operational semantics"
- Defines a relation of "one step" of computation, instead of complete evaluation
  - Determines granularity of atomic computaions
- Typically have two kinds of "result": configurations and final values
- Written $(C, m) \rightarrow (C', m')$ or $(C, m) \rightarrow m'$

Simple Imperative Programming Language #1 (SIMPL1)

$$
\begin{array}{lcl}
I & \in & \mathit{Identifiers} \\
N & \in & \mathit{Numerals} \\
E & ::= & N \mid I \mid E + E \mid E * E \mid E - E \\
B & ::= & \text{true} \mid \text{false} \mid B \& B \mid B \text{ or } B \mid \text{not } B \\
 & & \mid E < E \mid E = E \\
C & ::= & \text{skip} \mid C; C \mid \{C\} \mid I ::= E \\
 & & \mid \text{if } B \text{ then } C \text{ else } C \text{ fi} \\
 & & \mid \text{while } B \text{ do } C
\end{array}
$$

Transitions for Atomic Expressions

Identifiers: $(I, m) \longrightarrow m(I)$

Numerals are values: $(N, m) \longrightarrow N$

Booleans: $(\mathrm{true}, m) \longrightarrow \mathrm{true}$

$(\mathrm{false}, m) \longrightarrow \mathrm{false}$

Booleans:

- Values $= \{\text{true}, \text{ false}\}$
- Operators: (short-circuit)

$$(\text{false}\&B, m) \longrightarrow \text{false} \qquad \frac{(B, m) \longrightarrow (B'', m)}{(B\&B', m) \longrightarrow (B''\&B', m)}$$
$$(\text{true}\&B, m) \longrightarrow (B, m)$$

$$(\text{true or } B, m) \longrightarrow \text{true} \qquad \frac{(B, m) \longrightarrow (B'', m)}{(B \text{ or } B', m) \longrightarrow (B'' \text{ or } B', m)}$$
$$(\text{false or } B, m) \longrightarrow (B, m)$$

$$(\text{not true}, m) \longrightarrow \text{false} \qquad \frac{(B, m) \longrightarrow (B', m)}{(\text{not } B, m) \longrightarrow (\text{not } B', m)}$$
$$(\text{not false}, m) \longrightarrow \text{true}$$

Relations

- Let U, V be arithmetic values

$$\frac{(E, m) \longrightarrow (E'', m)}{(E \sim E', m) \longrightarrow (E'' \sim E', m)}$$

$$\frac{(E, m) \longrightarrow (E', m)}{(V \sim E, m) \longrightarrow (V \sim E', m)}$$

$$(U \sim V, m) \longrightarrow b$$

where $U \sim V = b$

Arithmetic Expressions

$$\frac{(E, m) \longrightarrow (E'', m)}{(E \oplus E', m) \longrightarrow (E'' \oplus E', m)}$$

$$\frac{(E, m) \longrightarrow (E', m)}{(V \oplus E, m) \longrightarrow (V \oplus E', m)}$$

$$(U \oplus V, m) \longrightarrow N$$

where N is the specified value for $U \oplus V$

Commands - in English

- `skip` means done evaluating
- When evaluating an assignment, evaluate expression first
- If the expression being assigned is a value, update the memory with the new value for the identifier
- When evaluating a sequence, work on the first command in the sequence first
- If the first command evaluates to a new memory (ie completes), evaluate remainder with new memory

Commands

Skip: $(\text{skip}, m) \longrightarrow m$

Assignment: $$\frac{(E, m) \longrightarrow (E', m)}{(I ::= E, m) \longrightarrow (I ::= E', m)}$$

$$(I ::= V, m) \longrightarrow m[I \leftarrow V]$$

Sequencing:

$$\frac{(C, m) \longrightarrow (C'', m')}{(C; C', m) \longrightarrow (C''; C', m')} \qquad \frac{(C, m) \longrightarrow m'}{(C; C', m) \longrightarrow (C', m')}$$

Block Command

- Choice of level of granularity:
  - Choice 1: Open a block is a unit of work

$$(\{C\}, m) \longrightarrow (C, m)$$

  - Choice 2: Blocks are syntactic sugar

$$\frac{(C, m) \longrightarrow (C', m')}{(\{C\}, m) \longrightarrow (C', m')} \qquad \frac{(C, m) \longrightarrow m'}{(\{C\}, m) \longrightarrow m'}$$

If Then Else Command - in English

- If the boolean guard in an `if_then_else` is true, then evaluate the first branch
- If it is false, evaluate the second branch
- If the boolean guard is not a value, then start by evaluating it first.

If Then Else Command

$$(\text{if true then } C \text{ else } C' \text{ fi}, m) \longrightarrow (C, m)$$

$$(\text{if false then } C \text{ else } C' \text{ fi}, m) \longrightarrow (C', m)$$

$$\frac{(B, m) \longrightarrow (B', m)}{(\text{if } B \text{ then } C \text{ else } C' \text{ fi}, m) \longrightarrow (\text{if } B' \text{ then } C \text{ else } C' \text{ fi}, m)}$$

While Command

$$(\text{while } B \text{ do } C, m) \longrightarrow (\text{if } B \text{ then } C; \text{while } B \text{ do } C \text{ else skip fi}, m)$$

- In English: Expand a `while` into a test of the boolean guard, with the true case being to do the body and then try the while loop again, and the false case being to stop.

Example

$$(\text{y := i; while i} > 0 \text{ do } \{\text{i := i - 1; y := y * i}\}, \langle \text{i} \mapsto 3\rangle)$$

$$\longrightarrow \underline{\ \ ?\ \ }$$

Alternate Semantics for SIMPL1

- Can mix Natural Semantics with Transition Semantics to get larger atomic computations
- Use $(E, m) \Downarrow v$ and $(B, m) \Downarrow b$ for arithmetics and boolean expressions
- Revise rules for commmands

Revised Rules for SIMPL1

Skip: $(\mathsf{skip}, m) \longrightarrow m$

Assignment: $$\frac{(E, m) \Downarrow v}{(I ::= E, m)} \longrightarrow m[I \leftarrow V]$$

Sequencing:

$$\frac{(C, m) \longrightarrow (C'', m')}{(C; C', m) \longrightarrow (C''; C', m')} \qquad \frac{(C, m) \longrightarrow m'}{(C; C', m) \longrightarrow (C', m')}$$

Blocks:

$$\frac{(C, m) \longrightarrow (C', m')}{(\{C\}, m) \longrightarrow (C', m')} \qquad \frac{(C, m) \longrightarrow m'}{(\{C\}, m) \longrightarrow m'}$$

If Then Else Command

$$\frac{(B, m) \Downarrow \text{true}}{(\text{if } B \text{ then } C \text{ else } C' \text{ fi}, m) \longrightarrow (C, m)}$$

$$\frac{(B, m) \Downarrow \text{false}}{(\text{if } B \text{ then } C \text{ else } C' \text{ fi}, m) \longrightarrow (C', m)}$$

Transition Semantics for SIMPL2?

- What are the choices and consequences for giving a transition semantics for the Simple Concurrent Imperative Programming Language #2, SIMP2?

Simple Concurrent Imperative Programming Language

$$
\begin{array}{lcl}
I & \in & \mathit{Identifiers} \\
N & \in & \mathit{Numerals} \\
E & ::= & N \mid I \mid E + E \mid E * E \mid E - E \\
B & ::= & \mathsf{true} \mid \mathsf{false} \mid B \& B \mid B \ \mathsf{or} \ B \mid \mathsf{not} \ B \\
 & & \mid E < E \mid E = E \\
C & ::= & \mathsf{skip} \mid C; C \mid \{C\} \mid I ::= E \mid C \| C' \\
 & & \mid \mathsf{if} \ B \ \mathsf{then} \ C \ \mathsf{else} \ C \ \mathsf{fi} \\
 & & \mid \mathsf{while} \ B \ \mathsf{do} \ C
\end{array}
$$