Deterministic Program — The While Program

Shangping Ren

Department of Computer Science
Illinois Institute of Technology

February 3, 2014
Deterministic Programs

- What does a program do?
- What does a deterministic program mean?
Deterministic Programs

- What does a program do?
- What does a deterministic program mean?
The else branch can be left off if the subexpression is simply a skip.

\( var(S) \) is the set of variable names appearing in \( S \).

\( change(S) \) is the set of variables appearing on the LHS of \( := \).
Now we have to express what these programs mean.

- The meaning of a program $S$ is a map $\mathcal{M}[S]$ from initial proper states to final states.
- The symbol $\perp$ indicates divergence.
Sample Program

if n < 0 then
  y := 1
else
  x := 0;
  y := 1;
  while x < n
     do
         x := x + 1;
         y := y + y
      od
fi

When the program finishes, what can we say about \( n \), \( x \), and \( y \)?
Operational Semantics

- How can we define $\mathcal{M}$?
- Hennessey and Plotkin’s idea: use *transition semantics*.
- A *transition relation* $\rightarrow$ on program $S$ and state $\sigma$:

  $$< S, \sigma > \rightarrow < R, \tau >$$

- $E$ represents the empty program.
- $< E, \sigma >$ represents termination in state $\sigma$. 
Definition of $\rightarrow$

\[
\begin{align*}
< \text{skip}, \sigma > & \rightarrow < E, \sigma > \\
< u := t, \sigma > & \rightarrow < E, \sigma [u := \sigma(t)] >
\end{align*}
\]
Definition of Sequencing

\[
\langle S_1, \sigma \rangle \rightarrow \langle S_2, \tau \rangle
\]

\[
\langle S_1; S, \sigma \rangle \rightarrow \langle S_2; S, \tau \rangle
\]
Definition of If

\[ \langle \text{if} \ B \text{ then } S_1 \text{ else } S_2 \text{ fi} \ , \sigma \rangle \rightarrow \langle S_1, \sigma \rangle \text{ where } \sigma \models B \]

- This is half the rule. Can you write the other half?
Definition of If, ctd.

\[ < \text{if } B \text{ then } S_1 \text{ else } S_2 \text{ fi }, \sigma > \rightarrow < S_2, \sigma > \text{ where } \sigma \models \neg B \]
Definition of While

\[ < \text{while } B \text{ do } S_1 \text{ od, } \sigma > \rightarrow < S; \text{while } B \text{ do } S_1 \text{ od, } \sigma > \text{ where } \sigma \models B \]

- This is half the rule. Can you write the other half?
Definition of While, ctd.

\(<\text{ while } B \text{ do } S_1 \text{ od }, \sigma > \rightarrow < E, \sigma > \text{ where } \sigma \models \neg B>
Some Definitions

- A **transition sequence of S starting in** $\sigma$ **is a finite or infinite sequence** $< S, \sigma > = < S_0, \sigma_0 > \rightarrow < S_1, \sigma_1 > \rightarrow \cdots$
- A **computation of S starting in** $\sigma$ **is a transition sequence of S starting at** $\sigma$ **that cannot be extended.**
- A computation of $S$ **terminates in** $\tau$ **if it is finite and the last configuration is** $< E, \tau >$.
- A computation of $S$ **diverges** if it is infinite.
- The notation $\rightarrow^*$ indicates zero or more transitions. (i.e., the reflexive transitive closure of $\rightarrow$.)
Determinism Lemma

- For any deterministic program $S$ and proper state $\sigma$, there is exactly one computation of $S$ starting in $\sigma$.

How do we know this?
**Non-Blocking Lemma**

For any deterministic program \( S \), if \( S \not\equiv E \) then for any proper state \( \sigma \) there exists a configuration \( < S_1, \tau > \) such that \( < S, \sigma > \rightarrow < S_1, \tau > \).

What does that mean? How do we know this?
Meaning of Programs as State Transformers — Examples

1. $W \equiv \text{while } x \leq n \text{ do } S \text{ od}$, where $S \equiv x := x + 1; y := y + y$ and $\sigma = \{x = 0, n = 2, y = 1\}$

2. $W \equiv \text{while } x \neq n \text{ do } x := x - 1 \text{ od}$, where $\sigma = \{x = 2, n = 0\}$

3. $W \equiv \text{while } x \neq n \text{ do } x := x - 1 \text{ od}$, where $\sigma = \{x = -1, n = 0\}$
Meaning of Programs as State Transformers — Examples

1. \( W \equiv \text{while } x \leq n \text{ do } S \text{ od } \), where \( S \equiv x := x + 1; y := y + y \) and \( \sigma = \{ x = 0, n = 2, y = 1 \} \)

2. \( W \equiv \text{while } x \neq n \text{ do } x := x - 1 \text{ od } \), where \( \sigma = \{ x = 2, n = 0 \} \)

3. \( W \equiv \text{while } x \neq n \text{ do } x := x - 1 \text{ od } \), where \( \sigma = \{ x = -1, n = 0 \} \)
Meaning of Programs as State Transformers — Examples

1. \( W \equiv \text{while } x \leq n \text{ do } S \text{ od } \), where \( S \equiv x := x + 1; y := y + y \) and \( \sigma = \{ x = 0, n = 2, y = 1 \} \)

2. \( W \equiv \text{while } x \neq n \text{ do } x := x - 1 \text{ od } \), where \( \sigma = \{ x = 2, n = 0 \} \)

3. \( W \equiv \text{while } x \neq n \text{ do } x := x - 1 \text{ od } \), where \( \sigma = \{ x = -1, n = 0 \} \)
Partial Correctness: “If the program terminates, it will give you the right answer.”

Total Correctness: “The program always terminates, and will give you the right answer.”

How to formally express these two correctness?

...it would appear that total correctness is better than partial correctness. Why should we bother with partial correctness then?
Total vs. Partial Correctness

- Partial Correctness: “If the program terminates, it will give you the right answer.”
- Total Correctness: “The program always terminates, and will give you the right answer.”

How to formally express these two correctness?

...it would appear that total correctness is better than partial correctness. Why should we bother with partial correctness then?
Total vs. Partial Correctness

- Partial Correctness: “If the program terminates, it will give you the right answer.”
- Total Correctness: “The program always terminates, and will give you the right answer.”

How to formally express these two correctness?

...it would appear that total correctness is better than partial correctness. Why should we bother with partial correctness then?
Formal Definition of Partial Correctness

Partial correctness semantics is a mapping

$$\mathcal{M}[S] : \Sigma \to \mathcal{P}(\Sigma)$$

such that

$$\mathcal{M}[S](\sigma) = \{ \tau \mid < S, \sigma > \rightarrow^* < E, \tau > \}$$

What would we have to do to make this definition work for total correctness?
Formal Definition of Total Correctness

Total correctness semantics is a mapping

\[ M_{tot}[S] : \Sigma \rightarrow \mathcal{P}(\Sigma \cup \{\bot\}) \]

such that

\[ M_{tot}[S](\sigma) = M[S](\sigma) \cup \{\bot | S \text{ can diverge from } \sigma\} \]
The semantic function $M$ works, but it’s a bit awkward to use.

Define $\{p\} S \{q\}$, where $p$ and $q$ are assertions (pre-condition and post-condition, respectively), and $S$ is a program:

- $\vdash \{p\} S \{q\}$ if $M[S]([p]) \subseteq [q]$.
- $\vdash_{tot} \{p\} S \{q\}$ if $M_{tot}[S]([p]) \subseteq [q]$.

The book refers to Hoare Triples as correctness formulas.
Examples

- \( \{ x = 0 \} x := x + 1 \{ x = 1 \} \)
- \( \{ x = 0 \} x := x + 1 \{ x > 0 \} \)
- \( \{ x = 0 \} x := x + 1 \{ \text{true} \} \)

False formulas...

- \( \{ x = 0 \} x := x + 1 \{ x = 2 \} \)
- \( \{ x = 0 \} x := x + 1 \{ x < 0 \} \)

What does this one mean? \( \{ x = 0 \} x := x + 1 \{ \text{false} \} \)
Satisfaction and Validity of a Correctness Triple

- \( \sigma \models p \) and \( \models p \)
- \( \models \) for partial correctness
- \( \models_{\text{tot}} \) for total correctness
Partial Correctness

\[ \sigma \models \{ p \} S \{ q \} : \text{if } \sigma \models p \land M[S](\sigma) \text{ is defined then} \]
\[ M[S](\sigma) \models q \]

Total Correctness

\[ \sigma \models^{\text{tot}} \{ p \} S \{ q \} : \text{if } \sigma \models p \text{ then} \]
\[ M[S](\sigma) \text{ is defined } \land M[S](\sigma) \models q \]
Examples

Under both correctness, are the following triples valid?

1. \( \{x > 0\} x := x + 1 \{x > 0\} \)
2. \( \{x > 0\} x := x - 1 \{x > 0\} \)
3. \( \{x \geq 0\} \text{while } x \neq 0 \text{ do } x := x - 1 \text{ od } \{x = 0\} \)
4. \( \{\text{true}\} \text{while } x \neq 0 \text{ do } x := x - 1 \text{ od } \{x = 0\} \)
Some Common Questions

For \{p\} S\{q\}

**Precondition** \(p\) is not satisfied (\(\sigma \not\models p\)):
the triple is satisfied, but no guarantee if \(M[S](\sigma)\) exists or not, or if it satisfies \(q\) or not.

**Program** \(S\) terminates and satisfies \(q\):
under both partial and total correctness, if \(M[S](\sigma)\) exists and \(M[S](\sigma) \models q\), then the triple is satisfied whether \(\sigma \models p\) or \(\sigma \not\models p\)

**Postcondition** \(q\) is a tautology, i.e., \(\models q\):
the triple is satisfied as long as \(M[S](\sigma)\) exists.
Program $S$ diverges, i.e., if $\mathcal{M}[S](\sigma)$ does not exist:

- **under partial correctness**: the triple is satisfied whether $\sigma \models p$ or $\sigma \not\models p$.
- **under total correctness**: 
  - $\sigma \models p$ : the triple is not satisfied
  - $\sigma \not\models p$ : the triple is satisfied regardless
Proof System (PW) Axiom 1: Skip

\(\{p\} \text{skip} \{p\}\)
Proof System (PW) Axiom 2: Assignment

\[ \{ p[u := t] \} u := t \{ p \} \]

- Is this what you expected?

\[ \{ y > 10 \} x := y \{ x > 10 \} \]
Proof System (PW) Rule 3: Composition

\[
\frac{\{p\} S_1 \{r\}, \{r\} S_2 \{q\}}{\{p\} S_1; S_2 \{q\}}
\]
Proof System (PW) Rule 4: Conditional

\[
\begin{align*}
\{p \land B\} & S_1 \{q\}, \{p \land \neg B\} S_2 \{q\} \\
\{p\} & \text{if } B \text{ then } S_1 \text{ else } S_2 \text{ fi } \{q\}
\end{align*}
\]
Proof System (PW) Rule 5: Loop

\[
\begin{align*}
\{p \land B\} & S\{p\} \\
\{p\} & \text{while } B \text{ do } S \text{ od } \{p \land \neg B\}
\end{align*}
\]
Proof System (PW) Rule 6: Consequence

- This one you will use a lot.

\[
\begin{align*}
p & \rightarrow p_1, \{p_1\} S\{q_1\}, \ q_1 \rightarrow q \\
\{p\} S\{q\}
\end{align*}
\]
Try to verify the following program.

\[
\{ true \} \quad \text{if } x > y \text{ then } m := x \text{ fi ; } \quad \{ m = \max(x, y) \}
\]

\[
\text{if } x < y \text{ then } m := y \text{ fi}
\]

(Hint: actually, it’s not true....)
Example 1

s := 0;
i := 0;
while (i < |A|) do
    s := s + A[i];
i := i + 1
od

- What is \( p \) and \( B \)?
- Verify that the Loop Rule holds.
- What does \( p \land \neg B \) give you?
Example 2

```plaintext
s := 0;
i := 1;
while (i < |A|) do
    s := s * A[i];
i := i + 1
od
```

- What is $p$ and $B$?
- Verify that the Loop Rule holds.
- What does $p \land \lnot B$ give you?
Example 3.3: consider the program $S \equiv x := 1; a[1] := 2; a[x] := x$
prove in the system PW the correctness formula

\[
\{ \text{true } \} S \{ a[1] = 1 \}
\]
Example 3. 4: Given two nature numbers $x$ and $y$, the program $DIV$ computes the quotient and reminder:

$$DIV \equiv quo := 0; rem := x; S_0$$

where

$$S_0 \equiv \text{while } rem \geq y \text{ do } rem := rem - y; quo := quo + 1 \text{ od}$$

prove

$$\models \{ x \geq 0 \land y \geq 0 \} DIV \{ quo \cdot y + rem = x \land 0 \leq rem < y \}$$
Example 3.4

- What is loop invariant?
  - \( p \equiv \text{quo} \cdot y + \text{rem} = x \land \text{rem} \geq 0 \)

- What is \( B \)?
  - \( B \equiv \text{rem} \geq y \)

- Need to prove three facts:
  1. \( \{ x \geq 0 \land y \geq 0 \} \text{quo} := 0; \text{rem} := x \{ p \}, \) i.e., the program \( \text{quo} := 0; \text{rem} := x \) establishes \( p \);
  2. \( \{ p \land \text{rem} \geq y \} \text{rem} := \text{rem} - y; \text{quo} := \text{quo} + 1 \{ p \}, \) i.e., \( p \) is indeed a loop invariant of \( S_0 \); and
  3. \( p \land \neg(\text{rem} \geq y) \rightarrow \text{quo} \cdot y + \text{rem} = x \land 0 \leq \text{rem} < y \)
Example 3.4

- What is loop invariant?
  - \( p \equiv quo \cdot y + rem = x \land rem \geq 0 \)
- What is \( B \)?
  - \( B \equiv rem \geq y \)
- Need to prove three facts:
  1. \( \{ x \geq 0 \land y \geq 0 \} quo := 0; rem := x \{p\}, \text{ i.e., the program} \)
     \( quo := 0; rem := x \) establishes \( p \);
  2. \( \{ p \land rem \geq y \} rem := rem - y; quo := quo + 1 \{p\}, \text{ i.e.,} \ p \text{ is indeed} \)
     a loop invariant of \( S_0 \); and
  3. \( p \land \neg(rem \geq y) \rightarrow quo \cdot y + rem = x \land 0 \leq rem < y \)
Example 3.4

- What is loop invariant?
  \[ p \equiv \text{quo} \cdot y + \text{rem} = x \land \text{rem} \geq 0 \]

- What is \( B \)?
  \[ B \equiv \text{rem} \geq y \]

- Need to prove three facts:
  1. \( \{ x \geq 0 \land y \geq 0 \} \text{quo} := 0; \text{rem} := x \{ p \}, \text{i.e., the program} \text{quo} := 0; \text{rem} := x \text{ establishes } p; \]
  2. \( \{ p \land \text{rem} \geq y \} \text{rem} := \text{rem} - y; \text{quo} := \text{quo} + 1 \{ p \}, \text{i.e., } p \text{ is indeed a loop invariant of } S_0; \text{ and} \]
  3. \( p \land \neg(\text{rem} \geq y) \rightarrow \text{quo} \cdot y + \text{rem} = x \land 0 \leq \text{rem} < y \)
Example 3.4

- What is loop invariant?
- \( p \equiv quo \cdot y + rem = x \land rem \geq 0 \)
- What is \( B \)?
- \( B \equiv rem \geq y \)

Need to prove three facts:
1. \( \{ x \geq 0 \land y \geq 0 \} quo := 0; rem := x \{ p \}, \) i.e., the program \( quo := 0; rem := x \) establishes \( p \);
2. \( \{ p \land rem \geq y \} rem := rem - y; quo := quo + 1 \{ p \}, \) i.e., \( p \) is indeed a loop invariant of \( S_0 \); and
3. \( p \land \neg (rem \geq y) \rightarrow quo \cdot y + rem = x \land 0 \leq rem < y \)
Example 3.4

- What is loop invariant?
- \( p \equiv quo \cdot y + rem = x \land rem \geq 0 \)
- What is \( B \)?
- \( B \equiv rem \geq y \)
- Need to prove three facts:
  1. \( \{x \geq 0 \land y \geq 0\}quo := 0; rem := x\{p\}, \text{i.e., the program}quo := 0; rem := x\text{ establishes } p;\)
  2. \( \{p \land rem \geq y\}rem := rem - y; quo := quo + 1\{p\}, \text{i.e., } p \text{ is indeed a loop invariant of } S_0; \text{ and} \)
  3. \( p \land \neg(rem \geq y) \rightarrow quo \cdot y + rem = x \land 0 \leq rem < y \)
The Problem

We now wish to use this to develop loops. We have a program of the form

\[ R \equiv T; \text{while } B \text{ do } S \text{ od } \]

satisfying

\[ \{r\} R \{q\} \]
The Proof Outline for Partial Correctness

\{r\}
\text{T ;}
\{inv : p\}
\text{while } B \text{ do}
\{p \land B\}
S
\{p\}
\text{od}
\{p \land \neg B\}
\{q\}
The Proof Outline for Partial Correctness

\begin{align*}
\{r\} & \quad \{x \geq y \land y \geq 0\} \\
T ; & \quad \text{quo := 0; rem := x;} \quad \{\text{inv : } p\} \\
\{\text{inv : } p\} & \quad \{\text{inv : } p\} \\
\text{while } B \text{ do} & \quad \text{while } \text{rem} \geq y \text{ do} \\
\quad \{p \land B\} & \quad \quad \{p \land \text{rem} \geq y\} \\
\quad S & \quad \quad \text{rem := rem - y; quo := quo + 1} \\
\quad \{p\} & \quad \quad \{p\} \\
\text{od} & \quad \text{od} \\
\{p \land \neg B\} & \quad \{p \land \text{rem} < y\} \\
\{q\} & \quad \{\text{quo \cdot y + rem} = x \land 0 \leq \text{rem} < y\} \\
\text{where } p & \equiv \text{quo \cdot y + rem} = x \land \text{rem} \geq 0
\end{align*}
How to Find the Loop Invariant

- The loop invariant is a weakened version of the postcondition.
  - Replace a constant with a range.
  - Delete a conjunct.
  - Add a disjunct (more rare).
- The basic technique: write the proof, then write the code.
Proof Outline: Partial Correctness — Skip, Assignment, Sequence, Consequence

\[
\begin{align*}
\{p\}\text{skip }\{p\} \\
\{p[u := t]\}u := t\{p\} \\
\{p\}S_1^*\{r\}, \{r\}S_2^*\{q\} \\
\{p\}S_1^*; \{r\}S_2^*\{q\} \\
p \rightarrow p_1, \{p_1\}S^*\{q_1\}, q_1 \rightarrow q \\
\{p\}\{p_1\}S^*\{q_1\}\{q\}
\end{align*}
\]
Proof Outline: Partial Correctness — If, While, Annotation

\[
\begin{align*}
\{p \land B\} S_1^* \{q\}, \{p \land \neg B\} S_2^* \{q\} \\
\{p\} \text{ if } B \text{ then } \{p \land B\} S_1^* \{q\} \text{ else } \{p \land \neg B\} S_2^* \{q\} \text{ fi } \{q\}
\end{align*}
\]

\[
\begin{align*}
\{p \land B\} S^* \{p\} \\
\text{inv: } \{p\} \text{ while } B \text{ do } \{p \land B\} S^* \{p\} \text{ od } \{p \land \neg B\}
\end{align*}
\]

\[
\begin{align*}
\{p\} S^* \{q\} \\
\{p\} S^{**} \{q\}
\end{align*}
\]

S^{**} results from S^* by omitting some annotations of the form \{r\}. 