

Language Semantics and Implementation

Lecture 9

Alex Simpson (als) and Colin Stirling

School of Informatics

February 8th 2010

Block structured local state

- ▶ Construct for introducing locally scoped locations.

Block structured local state

- ▶ Construct for introducing locally scoped locations.
- ▶ Involves dynamic creation of fresh locations.

Block structured local state

- ▶ Construct for introducing locally scoped locations.
- ▶ Involves dynamic creation of fresh locations.
- ▶ Essential for properly dealing with recursive procedures.

```
public class QuickSort
{  public static void sort(Vector array)
    {  sort(array, 0, array.size() - 1);
      }
  public static void sort(Vector array, int start, int end)
    {  int p;
       if (end > start)
         {  p = partition(array, start, end);
            sort(array, start, p-1);
            sort(array, p+1, end);
          }
      }
    }
  ...
}
```

The language LC^{loc} : LC + block structure

$P ::= C \mid E \mid B$

Phrases

$C ::= \text{skip} \mid \ell := E \mid C ; C \mid \text{if } B \text{ then } C_1 \text{ else } C_2$
 $\text{begin loc } \ell := E ; C \text{ end} \mid \text{while } B \text{ do } C$

Commands

$E ::= n \mid !\ell \mid E \text{ iop } E$

Integer expressions

$B ::= b \mid E \text{ bop } E$

Boolean expressions

Evaluation rules for blocks

$$\frac{\langle E, s \rangle \Downarrow \langle n, s' \rangle \quad \langle C[\ell' / \ell], s'[\ell' \mapsto n] \rangle \Downarrow \langle \mathbf{skip}, s''[\ell' \mapsto n'] \rangle}{\langle \mathbf{begin\ loc\ } \ell := E ; C \mathbf{end}, s \rangle \Downarrow \langle \mathbf{skip}, s'' \rangle}$$

Provided

Evaluation rules for blocks

$$\frac{\langle E, s \rangle \Downarrow \langle n, s' \rangle \quad \langle C[\ell' / \ell], s'[\ell' \mapsto n] \rangle \Downarrow \langle \mathbf{skip}, s''[\ell' \mapsto n'] \rangle}{\langle \mathbf{begin\ loc\ } \ell := E ; C \mathbf{end}, s \rangle \Downarrow \langle \mathbf{skip}, s'' \rangle}$$

Provided

- ▶ $\ell' \notin \text{dom}(s') \cup \text{dom}(s'')$

Evaluation rules for blocks

$$\frac{\langle E, s \rangle \Downarrow \langle n, s' \rangle \quad \langle C[\ell' / \ell], s'[\ell' \mapsto n] \rangle \Downarrow \langle \mathbf{skip}, s''[\ell' \mapsto n'] \rangle}{\langle \mathbf{begin\ loc\ } \ell := E ; C \mathbf{end}, s \rangle \Downarrow \langle \mathbf{skip}, s'' \rangle}$$

Provided

- ▶ $\ell' \notin \text{dom}(s') \cup \text{dom}(s'')$
- ▶ ℓ' does not occur in C

Evaluation rules for blocks

$$\frac{\langle E, s \rangle \Downarrow \langle n, s' \rangle \quad \langle C[\ell'/\ell], s'[\ell' \mapsto n] \rangle \Downarrow \langle \mathbf{skip}, s''[\ell' \mapsto n'] \rangle}{\langle \mathbf{begin\ loc\ } \ell := E ; C \mathbf{end}, s \rangle \Downarrow \langle \mathbf{skip}, s'' \rangle}$$

Provided

- ▶ $\ell' \notin \text{dom}(s') \cup \text{dom}(s'')$
- ▶ ℓ' does not occur in C
- ▶ $C[\ell'/\ell]$ is the LC^{loc} command obtained from C by replacing all occurrences of ℓ with ℓ'

Intuition for the block rule

In order to evaluate $\langle \mathbf{begin\ loc\ } \ell := E; C \mathbf{end}, s \rangle$

- ▶ evaluate $\langle E, s \rangle$ to $\langle n, s' \rangle$

Intuition for the block rule

In order to evaluate $\langle \mathbf{begin\ loc\ } \ell := E; C \mathbf{end}, s \rangle$

- ▶ evaluate $\langle E, s \rangle$ to $\langle n, s' \rangle$
- ▶ replace ℓ everywhere in C by a **new** location ℓ' satisfying

Intuition for the block rule

In order to evaluate $\langle \mathbf{begin\ loc\ } \ell := E; C \mathbf{end}, s \rangle$

- ▶ evaluate $\langle E, s \rangle$ to $\langle n, s' \rangle$
- ▶ replace ℓ everywhere in C by a **new** location ℓ' satisfying
 - ▶ ℓ' does not appear in C , and

Intuition for the block rule

In order to evaluate $\langle \mathbf{begin\ loc\ } \ell := E; C \mathbf{end}, s \rangle$

- ▶ evaluate $\langle E, s \rangle$ to $\langle n, s' \rangle$
- ▶ replace ℓ everywhere in C by a **new** location ℓ' satisfying
 - ▶ ℓ' does not appear in C , and
 - ▶ ℓ' hasn't been set to any value yet at s' ;

Intuition for the block rule

In order to evaluate $\langle \mathbf{begin\ loc\ } \ell := E; C \mathbf{ end}, s \rangle$

- ▶ evaluate $\langle E, s \rangle$ to $\langle n, s' \rangle$
- ▶ replace ℓ everywhere in C by a **new** location ℓ' satisfying
 - ▶ ℓ' does not appear in C , and
 - ▶ ℓ' hasn't been set to any value yet at s' ;
- ▶ extend s' by setting $\ell' \mapsto n$, yielding $s'[\ell' \mapsto n]$;

Intuition for the block rule

In order to evaluate $\langle \mathbf{begin\ loc\ } \ell := E; C \mathbf{ end}, s \rangle$

- ▶ evaluate $\langle E, s \rangle$ to $\langle n, s' \rangle$
- ▶ replace ℓ everywhere in C by a **new** location ℓ' satisfying
 - ▶ ℓ' does not appear in C , and
 - ▶ ℓ' hasn't been set to any value yet at s' ;
- ▶ extend s' by setting $\ell' \mapsto n$, yielding $s'[\ell' \mapsto n]$;
- ▶ evaluate $\langle C, s'[\ell' \mapsto n] \rangle$, yielding $\langle \mathbf{skip}, s''[\ell' \mapsto n'] \rangle$ for some state s'' ;
(why do we require $\ell' \notin \text{dom}(s'')$?)

Intuition for the block rule

In order to evaluate $\langle \mathbf{begin\ loc\ } \ell := E; C \mathbf{ end} , s \rangle$

- ▶ evaluate $\langle E, s \rangle$ to $\langle n, s' \rangle$
- ▶ replace ℓ everywhere in C by a **new** location ℓ' satisfying
 - ▶ ℓ' does not appear in C , and
 - ▶ ℓ' hasn't been set to any value yet at s' ;
- ▶ extend s' by setting $\ell' \mapsto n$, yielding $s'[\ell' \mapsto n]$;
- ▶ evaluate $\langle C, s'[\ell' \mapsto n] \rangle$, yielding $\langle \mathbf{skip}, s''[\ell' \mapsto n'] \rangle$ for some state s'' ;
(why do we require $\ell' \notin \text{dom}(s'')$?)
- ▶ let $\langle \mathbf{skip}, s'' \rangle$ be the result of evaluating $\langle \mathbf{begin\ loc\ } \ell := E; C \mathbf{ end} , s \rangle$

Some questions

- ▶ What does $\langle l := 1; \text{begin loc } l := !l + 1; \text{skip end}, s \rangle$ evaluate to?

Some questions

- ▶ What does $\langle l := 1; \text{begin loc } l := !l + 1; \text{skip end}, s \rangle$ evaluate to?
- ▶ Why not **begin loc** l C **end** as syntax?

Some questions

- ▶ What does $\langle \ell := 1; \text{begin loc } \ell := !\ell + 1; \text{skip end}, s \rangle$ evaluate to?
- ▶ Why not **begin loc** ℓ C **end** as syntax?
- ▶ Can we devise a transition semantics for LC^{loc} ?

Some questions

- ▶ What does $\langle l := 1; \text{begin loc } l := !l + 1; \text{skip end}, s \rangle$ evaluate to?
- ▶ Why not **begin loc** l C **end** as syntax?
- ▶ Can we devise a transition semantics for LC^{loc} ?
- ▶ What do the following two commands evaluate to starting from the state $\{l \mapsto 1, l' \mapsto 0\}$?

Some questions

- ▶ What does $\langle l := 1; \text{begin loc } l := !l + 1; \text{skip end}, s \rangle$ evaluate to?
- ▶ Why not **begin loc** l C **end** as syntax?
- ▶ Can we devise a transition semantics for LC^{loc} ?
- ▶ What do the following two commands evaluate to starting from the state $\{l \mapsto 1, l' \mapsto 0\}$?
 1. **begin loc** $l := !l + 1; l' := !l + 1$ **end**

Some questions

- ▶ What does $\langle l := 1; \text{begin loc } l := !l + 1; \text{skip end}, s \rangle$ evaluate to?
- ▶ Why not **begin loc** l C **end** as syntax?
- ▶ Can we devise a transition semantics for LC^{loc} ?
- ▶ What do the following two commands evaluate to starting from the state $\{l \mapsto 1, l' \mapsto 0\}$?
 1. **begin loc** $l := !l + 1; l' := !l + 1$ **end**
 2. **begin loc** $l := !l' + 1;$
 $\text{begin loc } l'' := !l + !l'; l := !l' + !l''$ **end**
end

Reasons for a formal semantics of a programming language

- ▶ form basis for reasoning about program properties and program specifications

Reasons for a formal semantics of a programming language

- ▶ form basis for reasoning about program properties and program specifications
- ▶ form basis for code optimisation

Reasons for a formal semantics of a programming language

- ▶ form basis for reasoning about program properties and program specifications
- ▶ form basis for code optimisation
 - ▶ when two program phrases have the same meaning

Reasons for a formal semantics of a programming language

- ▶ form basis for reasoning about program properties and program specifications
- ▶ form basis for code optimisation
 - ▶ when two program phrases have the same meaning
 - ▶ Equations $P = P'$: P and P' have the same meaning. Allows equational reasoning

Reasons for a formal semantics of a programming language

- ▶ form basis for reasoning about program properties and program specifications
- ▶ form basis for code optimisation
 - ▶ when two program phrases have the same meaning
 - ▶ Equations $P = P'$: P and P' have the same meaning. Allows equational reasoning
 - ▶ Especially interested when $=$ is a congruence

Reasons for a formal semantics of a programming language

- ▶ form basis for reasoning about program properties and program specifications
- ▶ form basis for code optimisation
 - ▶ when two program phrases have the same meaning
 - ▶ Equations $P = P'$: P and P' have the same meaning. Allows equational reasoning
 - ▶ Especially interested when $=$ is a congruence
 - ▶ if $P = P'$ then $C[P] = C[P']$ where $C[]$ a program phrase with a “hole”

Basic properties of equality

Reflexivity

$$P = P$$

Symmetry

$$\frac{P = P'}{P' = P}$$

Transitivity

$$\frac{P = P' \quad P' = P''}{P = P''}$$

Congruence

$$\frac{P = P'}{\mathcal{C}[P] = \mathcal{C}[P']}$$

where $\mathcal{C}[P]$ is a phrase containing an occurrence of P and $\mathcal{C}[P']$ is the same phrase with that occurrence replaced by P' .

Definition of semantic equivalence of LC phrases

Two phrases of the same type are semantically equivalent

$$P_1 \cong P_2$$

if and only if for all states s and all terminal configurations $\langle V, s' \rangle$

$$\langle P_1, s \rangle \Downarrow \langle V, s' \rangle \iff \langle P_2, s \rangle \Downarrow \langle V, s' \rangle.$$

Examples of semantically equivalent LC commands

▶ $C ; \text{skip} \cong C \cong \text{skip} ; C$

Examples of semantically equivalent LC commands

- ▶ $C ; \mathbf{skip} \cong C \cong \mathbf{skip} ; C$
- ▶ $(C ; C') ; C'' \cong C ; (C' ; C'')$

Examples of semantically equivalent LC commands

- ▶ $C ; \text{skip} \cong C \cong \text{skip} ; C$
- ▶ $(C ; C') ; C'' \cong C ; (C' ; C'')$
- ▶ $\text{if true then } C \text{ else } C' \cong C$

Examples of semantically equivalent LC commands

- ▶ $C ; \text{skip} \cong C \cong \text{skip} ; C$
- ▶ $(C ; C') ; C'' \cong C ; (C' ; C'')$
- ▶ $\text{if true then } C \text{ else } C' \cong C$
- ▶ $\text{if false then } C \text{ else } C' \cong C'$

Examples of semantically equivalent LC commands

- ▶ $C ; \text{skip} \cong C \cong \text{skip} ; C$
- ▶ $(C ; C') ; C'' \cong C ; (C' ; C'')$
- ▶ $\text{if true then } C \text{ else } C' \cong C$
- ▶ $\text{if false then } C \text{ else } C' \cong C'$
- ▶ $\text{while } B \text{ do } C \cong \text{if } B \text{ then } C ; (\text{while } B \text{ do } C) \text{ else skip}$

Examples of semantically equivalent LC commands

- ▶ $C ; \text{skip} \cong C \cong \text{skip} ; C$
- ▶ $(C ; C') ; C'' \cong C ; (C' ; C'')$
- ▶ **if true then C else $C' \cong C$**
- ▶ **if false then C else $C' \cong C'$**
- ▶ **while B do $C \cong \text{if } B \text{ then } C ; (\text{while } B \text{ do } C) \text{ else skip}$**
- ▶ $l := n ; l' := n' \cong \begin{cases} l' := n' ; l := n & \text{if } l \neq l' \\ l := n' & \text{if } l = l'. \end{cases}$

Exercise

- ▶ Is the following true?

(if B then C else C'); $C'' \cong$ (if B then C ; C'' else C' ; C'')

Exercise

- ▶ Is the following true?
 $(\text{if } B \text{ then } C \text{ else } C'); C'' \cong (\text{if } B \text{ then } C; C'' \text{ else } C'; C'')$
- ▶ What about?
 $\text{if } B \text{ then } C; C' \text{ else } C'' \cong \text{if } B \text{ then } C'; C \text{ else } C''$

Properties of \cong

Does \cong obey equality properties?

Properties of \cong

Does \cong obey equality properties?

- ▶ Reflexivity, Symmetry, Transitivity ?

Properties of \cong

Does \cong obey equality properties?

- ▶ Reflexivity, Symmetry, Transitivity ?
- ▶ Congruence? How to prove this?

Semantic equivalence is a congruence: the while case

Need to prove

if $C_1 \cong C_2$,
then **while** B **do** $C_1 \cong$ **while** B **do** C_2

Semantic equivalence is a congruence: the while case

Need to prove

if $C_1 \cong C_2$,
then **while** B **do** $C_1 \cong$ **while** B **do** C_2

By the equivalence of the LC transition and evaluation semantics,
it suffices to prove

if $C_1 \cong C_2$,
then for all states s, s' ,
 $\langle \mathbf{while} B \mathbf{do} C_1, s \rangle \rightarrow^* \langle \mathbf{skip}, s' \rangle$
if and only if
 $\langle \mathbf{while} B \mathbf{do} C_2, s \rangle \rightarrow^* \langle \mathbf{skip}, s' \rangle$

By symmetry and by the definition of \rightarrow^* , it suffices to prove

if $C_1 \cong C_2$,

then for all states s, s' ,

for all $n \geq 0$,

if $\langle \mathbf{while} B \mathbf{ do} C_1, s \rangle \rightarrow^n \langle \mathbf{skip}, s' \rangle$

then $\langle \mathbf{while} B \mathbf{ do} C_2, s \rangle \rightarrow^* \langle \mathbf{skip}, s' \rangle$

By symmetry and by the definition of \rightarrow^* , it suffices to prove

if $C_1 \cong C_2$,

then for all states s, s' ,

for all $n \geq 0$,

if $\langle \mathbf{while} B \mathbf{ do} C_1, s \rangle \rightarrow^n \langle \mathbf{skip}, s' \rangle$

then $\langle \mathbf{while} B \mathbf{ do} C_2, s \rangle \rightarrow^* \langle \mathbf{skip}, s' \rangle$

which is equivalent to proving

if $C_1 \cong C_2$,

then for all $n \geq 0$,

(*) for all states s, s' ,

if $\langle \mathbf{while} B \mathbf{ do} C_1, s \rangle \rightarrow^n \langle \mathbf{skip}, s' \rangle$

then $\langle \mathbf{while} B \mathbf{ do} C_2, s \rangle \rightarrow^* \langle \mathbf{skip}, s' \rangle$

By symmetry and by the definition of \rightarrow^* , it suffices to prove

if $C_1 \cong C_2$,

then for all states s, s' ,

for all $n \geq 0$,

if $\langle \mathbf{while} B \mathbf{do} C_1, s \rangle \rightarrow^n \langle \mathbf{skip}, s' \rangle$

then $\langle \mathbf{while} B \mathbf{do} C_2, s \rangle \rightarrow^* \langle \mathbf{skip}, s' \rangle$

which is equivalent to proving

if $C_1 \cong C_2$,

then for all $n \geq 0$,

(*) for all states s, s' ,

if $\langle \mathbf{while} B \mathbf{do} C_1, s \rangle \rightarrow^n \langle \mathbf{skip}, s' \rangle$

then $\langle \mathbf{while} B \mathbf{do} C_2, s \rangle \rightarrow^* \langle \mathbf{skip}, s' \rangle$

This we show by complete mathematical induction.

We assume $C_1 \cong C_2$. We prove:

- ▶ (*) holds for $n = 0$.
- ▶ if (*) holds for all $m \leq n$, then it also holds for $n + 1$.

Exercise: compiler optimization

- ▶ Does the following hold?

$$i := E; j := E \cong i := E; j := !i$$

Exercise: compiler optimization

- ▶ Does the following hold?

$$i := E; j := E \cong i := E; j := !i$$

- ▶ Are the following pair equivalent?

```
while (1 ≤ !i and !i ≤ n) do  
  s := !s + (!i × E);  
  i := !i + 1  
  
begin loc k := E;  
  while (1 ≤ !i and !i ≤ n) do  
    s := !s + (!i × !k);  
    i := !i + 1  
  
end
```

Reasons for a formal semantics of a programming language

- ▶ form basis for code optimisation

Reasons for a formal semantics of a programming language

- ▶ form basis for code optimisation
- ▶ form basis for reasoning about program properties and program specifications

An application of semantics to correctness

- ▶ The idea of partial correctness

An application of semantics to correctness

- ▶ **The idea of partial correctness**
- ▶ $\{ \text{formula} \} P \{ \text{formula}' \}$ means if formula is true before execution of P and P terminates then formula' is true after execution

An application of semantics to correctness

- ▶ The idea of partial correctness
- ▶ $\{ \text{formula} \} P \{ \text{formula}' \}$ means if formula is true before execution of P and P terminates then formula' is true after execution
- ▶ We can make this precise as follows

An application of semantics to correctness

- ▶ The idea of partial correctness
- ▶ $\{ \text{formula} \} P \{ \text{formula}' \}$ means if formula is true before execution of P and P terminates then formula' is true after execution
- ▶ We can make this precise as follows
- ▶ if formula is true in state s and $\langle P, s \rangle \Downarrow \langle \text{skip}, s' \rangle$, then formula' is true in state s'

An application of semantics to correctness

- ▶ The idea of partial correctness
- ▶ $\{ \text{formula} \} P \{ \text{formula}' \}$ means if formula is true before execution of P and P terminates then formula' is true after execution
- ▶ We can make this precise as follows
- ▶ if formula is true in state s and $\langle P, s \rangle \Downarrow \langle \mathbf{skip}, s' \rangle$, then formula' is true in state s'
- ▶ Example: $\{!x = 3\} x := !x + 5; x := !x * !x \{!x = 64\}$

An application of semantics to correctness

- ▶ The idea of partial correctness
- ▶ $\{ \text{formula} \} P \{ \text{formula}' \}$ means if formula is true before execution of P and P terminates then formula' is true after execution
- ▶ We can make this precise as follows
- ▶ if formula is true in state s and $\langle P, s \rangle \Downarrow \langle \mathbf{skip}, s' \rangle$, then formula' is true in state s'
- ▶ Example: $\{!x = 3\} x := !x + 5; x := !x * !x \{!x = 64\}$
- ▶ More than this: use the semantics to justify proof system

Proof rules of Hoare Logic

$$\{p[E/!x]\}x := E\{p\}$$

$p[E/!x]$ is p when all (free) occurrences of $!x$ are replaced by E

$$\frac{\{p\}C_1\{q\} \quad \{q\}C_2\{r\}}{\{p\}C_1; C_2\{r\}}$$

$$\frac{\{B \wedge p\}C_1\{q\} \quad \{\neg B \wedge p\}C_2\{q\}}{\{p\}\mathbf{if } B \mathbf{ then } C_1 \mathbf{ else } C_2\{q\}}$$

$$\frac{\{B \wedge p\}C\{p\}}{\{p\}\mathbf{while } B \mathbf{ do } C\{p \wedge \neg B\}}$$

$$p \Rightarrow p' \quad \frac{\{p'\}C\{q'\}}{\{p\}C\{q\}} \quad q' \Rightarrow q$$

And $p \Rightarrow p'$ means p implies p'

Hoare Logic continued

- ▶ Example of axiom (+ final rule)

$$\{!x = 3\}x := !x + 5\{!x = 8\}$$

Because $(!x = 8)[!x + 5/!x]$ is $(!x + 5 = 8)$ is $!x = 3$

Hoare Logic continued

- ▶ Example of axiom (+ final rule)

$$\{!x = 3\}x := !x + 5\{!x = 8\}$$

Because $(!x = 8)[!x + 5/!x]$ is $(!x + 5 = 8)$ is $!x = 3$

- ▶ **Exercise:** use LC evaluation semantics to show soundness of the logic. Show that the axiom is true and that the rules preserve truth.

Hoare Logic continued

- ▶ Example of axiom (+ final rule)

$$\{!x = 3\}x := !x + 5\{!x = 8\}$$

Because $(!x = 8)[!x + 5/!x]$ is $(!x + 5 = 8)$ is $!x = 3$

- ▶ **Exercise:** use LC evaluation semantics to show soundness of the logic. Show that the axiom is true and that the rules preserve truth.
- ▶ **That is prove:** if $p[E/!x]$ is true at s and $\langle x := E, s \rangle \Downarrow \langle \mathbf{skip}, s' \rangle$ then p is true at state s'

Hoare Logic continued

- ▶ Example of axiom (+ final rule)

$$\{!x = 3\}x := !x + 5\{!x = 8\}$$

Because $(!x = 8)[!x + 5/!x]$ is $(!x + 5 = 8)$ is $!x = 3$

- ▶ **Exercise:** use LC evaluation semantics to show soundness of the logic. Show that the axiom is true and that the rules preserve truth.

- ▶ **That is prove:** if $p[E/!x]$ is true at s and $\langle x := E, s \rangle \Downarrow \langle \mathbf{skip}, s' \rangle$ then p is true at state s'

- ▶ And, for example, for the while rule.

Assume for any s if $B \wedge p$ is true at s and $\langle C, s \rangle \Downarrow \langle \mathbf{skip}, s' \rangle$ then p is true at s' .

Then show: for any s , if p is true at s and $\langle \mathbf{while } B \mathbf{ do } C, s \rangle \Downarrow \langle \mathbf{skip}, s' \rangle$ then $p \wedge \neg B$ is true at s' .

Hoare Logic continued

- ▶ Example of axiom (+ final rule)

$$\{\!|x = 3|\!\} x := !x + 5 \{\!|x = 8|\!\}$$

Because $(\!|x = 8|\!) [!x + 5 / !x]$ is $(\!|x + 5 = 8|\!)$ is $\!|x = 3$

- ▶ **Exercise:** use LC evaluation semantics to show soundness of the logic. Show that the axiom is true and that the rules preserve truth.

- ▶ **That is prove:** if $p[E / !x]$ is true at s and $\langle x := E, s \rangle \Downarrow \langle \mathbf{skip}, s' \rangle$ then p is true at state s'

- ▶ And, for example, for the while rule.

Assume for any s if $B \wedge p$ is true at s and $\langle C, s \rangle \Downarrow \langle \mathbf{skip}, s' \rangle$ then p is true at s' .

Then show: for any s , if p is true at s and $\langle \mathbf{while } B \mathbf{ do } C, s \rangle \Downarrow \langle \mathbf{skip}, s' \rangle$ then $p \wedge \neg B$ is true at s' .

- ▶ And similarly for the other rules