Concrete Semantics

with Isabelle/HOL

Peter Lammich (slides adopted from Tobias Nipkow)

Fakultät für Informatik
Technische Universität München

2018-11-18

Part II

Semantics

Chapter 7

IMP:
A Simple Imperative Language

1. IMP Commands

2. Big-Step Semantics

3. Small-Step Semantics

1. IMP Commands

2. Big-Step Semantics

3. Small-Step Semantics

Terminology

Statement: declaration of fact or claim

Terminology

Statement: declaration of fact or claim

Semantics is easy.

Terminology

Statement: declaration of fact or claim

Semantics is easy.

Command: order to do something

Terminology

Statement: declaration of fact or claim

Semantics is easy.

Command: order to do something

Study the book until you have understood it.

Terminology

Statement: declaration of fact or claim

Semantics is easy.

Command: order to do something

Study the book until you have understood it.

Expressions are *evaluated*, commands are *executed*

Commands

Concrete syntax:

$$
\begin{array}{rcl}
com & ::= & \texttt{SKIP} \\
& | & string\ \texttt{::=}\ aexp \\
& | & com\ \texttt{;;}\ com \\
& | & \texttt{IF}\ bexp\ \texttt{THEN}\ com\ \texttt{ELSE}\ com \\
& | & \texttt{WHILE}\ bexp\ \texttt{DO}\ com
\end{array}
$$

Commands

Abstract syntax:

$$
\begin{array}{rcl}
\textbf{datatype}\ com & = & SKIP \\
& | & Assign\ string\ aexp \\
& | & Seq\ com\ com \\
& | & If\ \ bexp\ com\ com \\
& | & While\ bexp\ com
\end{array}
$$

Com.thy

1. IMP Commands

2. Big-Step Semantics

3. Small-Step Semantics

Big-step semantics

Concrete syntax:

$$(\mathit{com},\ \mathit{initial\text{-}state}) \Rightarrow \mathit{final\text{-}state}$$

Big-step semantics

Concrete syntax:

$$(com,\ initial\text{-}state) \Rightarrow final\text{-}state$$

Intended meaning of $(c,\ s) \Rightarrow t$:

Big-step semantics

Concrete syntax:

$$(com,\ initial\text{-}state) \Rightarrow final\text{-}state$$

Intended meaning of $(c,\ s) \Rightarrow t$:

Command c started in state s terminates in state t

Big-step semantics

Concrete syntax:

$$(com,\ initial\text{-}state) \Rightarrow final\text{-}state$$

Intended meaning of $(c,\ s) \Rightarrow t$:

Command c started in state s terminates in state t

"$\Rightarrow$" here not type!

Big-step rules

$$(SKIP,\ s) \Rightarrow\ s$$

Big-step rules

$$(SKIP,\ s) \Rightarrow s$$

$$(x ::= a,\ s) \Rightarrow s(x := aval\ a\ s)$$

Big-step rules

$$(SKIP,\ s) \Rightarrow s$$

$$(x ::= a,\ s) \Rightarrow s(x := aval\ a\ s)$$

$$\frac{(c_1,\ s_1) \Rightarrow s_2 \qquad (c_2,\ s_2) \Rightarrow s_3}{(c_1;;\ c_2,\ s_1) \Rightarrow s_3}$$

Big-step rules

$$\frac{bval\ b\ s \qquad (c_1,\ s) \Rightarrow t}{(IF\ b\ THEN\ c_1\ ELSE\ c_2,\ s) \Rightarrow t}$$

Big-step rules

$$\frac{bval\ b\ s \qquad (c_1,\ s) \Rightarrow t}{(IF\ b\ THEN\ c_1\ ELSE\ c_2,\ s) \Rightarrow t}$$

$$\frac{\neg\ bval\ b\ s \qquad (c_2,\ s) \Rightarrow t}{(IF\ b\ THEN\ c_1\ ELSE\ c_2,\ s) \Rightarrow t}$$

Big-step rules

$$\frac{\neg\ bval\ b\ s}{(WHILE\ b\ DO\ c,\ s) \Rightarrow s}$$

Big-step rules

$$\frac{\neg\ bval\ b\ s}{(WHILE\ b\ DO\ c,\ s) \Rightarrow s}$$

$$\frac{bval\ b\ s_1 \quad (c,\ s_1) \Rightarrow s_2 \quad (WHILE\ b\ DO\ c,\ s_2) \Rightarrow s_3}{(WHILE\ b\ DO\ c,\ s_1) \Rightarrow s_3}$$

Examples: derivation trees

$$\frac{\vdots}{(''x'' ::= N\;5;;\; ''y'' ::= V\; ''x'',\; s) \Rightarrow \;\; ?}$$

Examples: derivation trees

$$\frac{\vdots}{(''x'' ::= N\,5;;\ ''y'' ::= V\,''x'',\ s) \Rightarrow\ ?} \qquad \frac{\vdots}{(w,\ s_i) \Rightarrow\ ?}$$

where
$$\begin{array}{rcl} w & = & WHILE\ b\ DO\ c \\ b & = & NotEq\ (V\ ''x'')\ (N\,2) \\ c & = & ''x'' ::= Plus\ (V\ ''x'')\ (N\,1) \\ s_i & = & s(''x'' := i) \end{array}$$

Examples: derivation trees

$$\frac{\vdots}{(''x'' ::= N\ 5;;\ ''y'' ::= V\ ''x'',\ s) \Rightarrow\ ?} \qquad \frac{\vdots}{(w,\ s_i) \Rightarrow\ ?}$$

where

$$\begin{aligned} w &= WHILE\ b\ DO\ c \\ b &= NotEq\ (V\ ''x'')\ (N\ 2) \\ c &= ''x'' ::= Plus\ (V\ ''x'')\ (N\ 1) \\ s_i &= s(''x'' := i) \end{aligned}$$

$NotEq\ a_1\ a_2 =$
$Not(And\ (Not(Less\ a_1\ a_2))\ (Not(Less\ a_2\ a_1)))$

Logically speaking

$$(c,\ s) \Rightarrow t$$

is just infix syntax for

$$big\_step\ (c{,}s)\ t$$

Logically speaking

$$(c,\ s) \Rightarrow t$$

is just infix syntax for

$$big\_step\ (c,s)\ t$$

where

$$big\_step :: com \times state \Rightarrow state \Rightarrow bool$$

is an inductively defined predicate.

Big_Step.thy

Semantics

Rule inversion

What can we deduce from

- $(SKIP, s) \Rightarrow t$?

Rule inversion

What can we deduce from

- $(SKIP,\ s) \Rightarrow t$? $\quad t = s$

Rule inversion

What can we deduce from

- $(SKIP,\ s) \Rightarrow t$? $t = s$
- $(x ::= a,\ s) \Rightarrow t$?

Rule inversion

What can we deduce from

- $(SKIP,\ s) \Rightarrow t$? $\quad t = s$
- $(x ::= a,\ s) \Rightarrow t$? $\quad t = s(x := aval\ a\ s)$

Rule inversion

What can we deduce from

- $(SKIP,\ s) \Rightarrow t$? $\quad t = s$
- $(x ::= a,\ s) \Rightarrow t$? $\quad t = s(x := aval\ a\ s)$
- $(c_1;;\ c_2,\ s_1) \Rightarrow s_3$?

Rule inversion

What can we deduce from

- $(SKIP,\ s) \Rightarrow t$? $\quad t = s$
- $(x ::= a,\ s) \Rightarrow t$? $\quad t = s(x := aval\ a\ s)$
- $(c_1;;\ c_2,\ s_1) \Rightarrow s_3$?
  $\exists\, s_2.\ (c_1,\ s_1) \Rightarrow s_2 \wedge (c_2,\ s_2) \Rightarrow s_3$

Rule inversion

What can we deduce from

- $(SKIP,\ s) \Rightarrow t$? $t = s$
- $(x ::= a,\ s) \Rightarrow t$? $t = s(x := aval\ a\ s)$
- $(c_1;;\ c_2,\ s_1) \Rightarrow s_3$?
  $\exists\, s_2.\ (c_1,\ s_1) \Rightarrow s_2 \wedge (c_2,\ s_2) \Rightarrow s_3$
- $(IF\ b\ THEN\ c_1\ ELSE\ c_2,\ s) \Rightarrow t$?

Rule inversion

What can we deduce from

- $(SKIP,\ s) \Rightarrow t$? $\quad t = s$
- $(x ::= a,\ s) \Rightarrow t$? $\quad t = s(x := aval\ a\ s)$
- $(c_1;;\ c_2,\ s_1) \Rightarrow s_3$?
  $\exists\, s_2.\ (c_1,\ s_1) \Rightarrow s_2 \wedge (c_2,\ s_2) \Rightarrow s_3$
- $(IF\ b\ THEN\ c_1\ ELSE\ c_2,\ s) \Rightarrow t$?
  $bval\ b\ s \wedge (c_1,\ s) \Rightarrow t\ \vee$
  $\neg\ bval\ b\ s \wedge (c_2,\ s) \Rightarrow t$

Rule inversion

What can we deduce from

- $(SKIP,\ s) \Rightarrow t$? $\quad t = s$
- $(x ::= a,\ s) \Rightarrow t$? $\quad t = s(x := aval\ a\ s)$
- $(c_1;;\ c_2,\ s_1) \Rightarrow s_3$?
  $\exists\, s_2.\ (c_1,\ s_1) \Rightarrow s_2 \wedge (c_2,\ s_2) \Rightarrow s_3$
- $(IF\ b\ THEN\ c_1\ ELSE\ c_2,\ s) \Rightarrow t$?
  $bval\ b\ s \wedge (c_1,\ s) \Rightarrow t\ \vee$
  $\neg\ bval\ b\ s \wedge (c_2,\ s) \Rightarrow t$
- $(w,\ s) \Rightarrow t$ where $w = WHILE\ b\ DO\ c$?

Rule inversion

What can we deduce from

- $(SKIP,\ s) \Rightarrow t$? $\quad t = s$
- $(x ::= a,\ s) \Rightarrow t$? $\quad t = s(x := aval\ a\ s)$
- $(c_1;;\ c_2,\ s_1) \Rightarrow s_3$?
  $\exists\, s_2.\ (c_1,\ s_1) \Rightarrow s_2 \wedge (c_2,\ s_2) \Rightarrow s_3$
- $(IF\ b\ THEN\ c_1\ ELSE\ c_2,\ s) \Rightarrow t$?
  $bval\ b\ s \wedge (c_1,\ s) \Rightarrow t\ \vee$
  $\neg\ bval\ b\ s \wedge (c_2,\ s) \Rightarrow t$
- $(w,\ s) \Rightarrow t$ where $w = WHILE\ b\ DO\ c$?
  $\neg\ bval\ b\ s \wedge t = s\ \vee$
  $bval\ b\ s \wedge (\exists\, s'.\ (c,\ s) \Rightarrow s' \wedge (w,\ s') \Rightarrow t)$

Automating rule inversion

Isabelle command **inductive_cases** produces theorems that perform rule inversions automatically.

We reformulate the inverted rules. Example:

$$\frac{(c_1;;\ c_2,\ s_1) \Rightarrow s_3}{\exists\, s_2.\ (c_1,\ s_1) \Rightarrow s_2 \wedge (c_2,\ s_2) \Rightarrow s_3}$$

We reformulate the inverted rules. Example:

$$\frac{(c_1;;\ c_2,\ s_1) \Rightarrow\ s_3}{\exists\, s_2.\ (c_1,\ s_1) \Rightarrow\ s_2 \wedge (c_2,\ s_2) \Rightarrow\ s_3}$$

is logically equivalent to

$$\frac{(c_1;;\ c_2,\ s_1) \Rightarrow\ s_3 \qquad \bigwedge s_2.\ [\![(c_1,\ s_1) \Rightarrow\ s_2;\ (c_2,\ s_2) \Rightarrow\ s_3]\!] \Longrightarrow P}{P}$$

We reformulate the inverted rules. Example:

$$\frac{(c_1;;\ c_2,\ s_1) \Rightarrow s_3}{\exists\, s_2.\ (c_1,\ s_1) \Rightarrow s_2 \wedge (c_2,\ s_2) \Rightarrow s_3}$$

is logically equivalent to

$$\frac{(c_1;;\ c_2,\ s_1) \Rightarrow s_3 \qquad \bigwedge s_2.\ [\![(c_1,\ s_1) \Rightarrow s_2;\ (c_2,\ s_2) \Rightarrow s_3]\!] \Longrightarrow P}{P}$$

Replaces assm $(c_1;;\ c_2,\ s_1) \Rightarrow s_3$ by two assms $(c_1,\ s_1) \Rightarrow s_2$ and $(c_2,\ s_2) \Rightarrow s_3$

We reformulate the inverted rules. Example:

$$\frac{(c_1;;\ c_2,\ s_1) \Rightarrow s_3}{\exists\, s_2.\ (c_1,\ s_1) \Rightarrow s_2 \wedge (c_2,\ s_2) \Rightarrow s_3}$$

is logically equivalent to

$$\frac{(c_1;;\ c_2,\ s_1) \Rightarrow s_3 \qquad \bigwedge s_2.\ [\![(c_1,\ s_1) \Rightarrow s_2;\ (c_2,\ s_2) \Rightarrow s_3]\!] \Longrightarrow P}{P}$$

Replaces assm $(c_1;;\ c_2,\ s_1) \Rightarrow s_3$ by two assms $(c_1,\ s_1) \Rightarrow s_2$ and $(c_2,\ s_2) \Rightarrow s_3$ (with a new fixed s_2).

We reformulate the inverted rules. Example:

$$\frac{(c_1;;\ c_2,\ s_1) \Rightarrow s_3}{\exists\, s_2.\ (c_1,\ s_1) \Rightarrow s_2 \wedge (c_2,\ s_2) \Rightarrow s_3}$$

is logically equivalent to

$$\frac{\begin{array}{c}(c_1;;\ c_2,\ s_1) \Rightarrow s_3 \\ \bigwedge s_2.\ [\![(c_1,\ s_1) \Rightarrow s_2;\ (c_2,\ s_2) \Rightarrow s_3]\!] \Longrightarrow P\end{array}}{P}$$

Replaces assm $(c_1;;\ c_2,\ s_1) \Rightarrow s_3$ by two assms $(c_1,\ s_1) \Rightarrow s_2$ and $(c_2,\ s_2) \Rightarrow s_3$ (with a new fixed s_2).
No $\exists$ and $\wedge$!

The general format: *elimination rules*

$$\frac{asm \quad asm_1 \Longrightarrow P \quad \ldots \quad asm_n \Longrightarrow P}{P}$$

The general format: *elimination rules*

$$\frac{asm \quad asm_1 \Longrightarrow P \quad \dots \quad asm_n \Longrightarrow P}{P}$$

(possibly with $\bigwedge \overline{x}$ in front of the $asm_i \Longrightarrow P$)

The general format: *elimination rules*

$$\frac{asm \quad asm_1 \Longrightarrow P \quad \ldots \quad asm_n \Longrightarrow P}{P}$$

(possibly with $\bigwedge \overline{x}$ in front of the $asm_i \Longrightarrow P$)

Reading:

To prove a goal P with assumption asm,
prove all $asm_i \Longrightarrow P$

The general format: *elimination rules*

$$\frac{asm \quad asm_1 \Longrightarrow P \quad \ldots \quad asm_n \Longrightarrow P}{P}$$

(possibly with $\bigwedge \overline{x}$ in front of the $asm_i \Longrightarrow P$)

Reading:

> To prove a goal P with assumption asm,
> prove all $asm_i \Longrightarrow P$

Example:

$$\frac{F \ \vee \ G \quad F \ \Longrightarrow \ P \quad G \ \Longrightarrow \ P}{P}$$

$elim$ attribute

- Theorems with $elim$ attribute are used automatically by $blast$, $fastforce$ and $auto$

elim attribute

- Theorems with *elim* attribute are used automatically by *blast*, *fastforce* and *auto*
- Can also be added locally, eg *(blast elim: ...)*

elim attribute

- Theorems with *elim* attribute are used automatically by *blast*, *fastforce* and *auto*
- Can also be added locally, eg *(blast elim: . . .)*
- Variant: *elim!* applies elim-rules eagerly.

Big_Step.thy

Rule inversion

Command equivalence

Two commands have the same input/output behaviour:

Command equivalence

Two commands have the same input/output behaviour:

$$c \sim c' \;\equiv\; (\forall\, s\ t.\ (c,s) \Rightarrow t \longleftrightarrow (c',s) \Rightarrow t)$$

Command equivalence

Two commands have the same input/output behaviour:

$$c \sim c' \;\equiv\; (\forall\, s\ t.\ (c,s) \Rightarrow t \longleftrightarrow (c',s) \Rightarrow t)$$

Example

$$w \sim w'$$

where $w = \mathit{WHILE}\ b\ \mathit{DO}\ c$
$w' = \mathit{IF}\ b\ \mathit{THEN}\ c;;\ w\ \mathit{ELSE}\ \mathit{SKIP}$

Equivalence proof

$$(w,\ s) \Rightarrow\ t$$

Equivalence proof

$$(w,\ s) \Rightarrow\ t$$

$$\longleftrightarrow$$

$$bval\ b\ s \wedge (\exists\, s'.\ (c,\ s) \Rightarrow\ s' \wedge (w,\ s') \Rightarrow\ t)$$
$$\vee$$
$$\neg\ bval\ b\ s \wedge t =\ s$$

Equivalence proof

$$(w,\ s) \Rightarrow t$$

$$\longleftrightarrow$$

$$\begin{gathered} bval\ b\ s \wedge (\exists\, s'.\ (c,\ s) \Rightarrow s' \wedge (w,\ s') \Rightarrow t) \\ \vee \\ \neg\ bval\ b\ s \wedge t = s \end{gathered}$$

$$\longleftrightarrow$$

$$(w',\ s) \Rightarrow t$$

Equivalence proof

$$(w,\ s) \Rightarrow t$$

$$\longleftrightarrow$$

$$bval\ b\ s \wedge (\exists\, s'.\ (c,\ s) \Rightarrow s' \wedge (w,\ s') \Rightarrow t)$$
$$\vee$$
$$\neg\ bval\ b\ s \wedge t = s$$

$$\longleftrightarrow$$

$$(w',\ s) \Rightarrow t$$

Using the rules and rule inversions for $\Rightarrow$.

`Big_Step.thy`

Command equivalence

Execution is deterministic

Any two executions of the same command in the same start state lead to the same final state:

$$(c,\ s) \Rightarrow t \implies (c,\ s) \Rightarrow t' \implies t = t'$$

Execution is deterministic

Any two executions of the same command in the same start state lead to the same final state:

$$(c,\ s) \Rightarrow t \implies (c,\ s) \Rightarrow t' \implies t = t'$$

Proof by rule induction, for arbitrary t'.

Big_Step.thy

Execution is deterministic

The boon and bane of big steps

We cannot observe intermediate states/steps

The boon and bane of big steps

We cannot observe intermediate states/steps

Example problem:

The boon and bane of big steps

We cannot observe intermediate states/steps

Example problem:

(c,s) does not terminate iff $\nexists\, t.\ (c,\ s) \Rightarrow\ t$?

The boon and bane of big steps

We cannot observe intermediate states/steps

Example problem:

$$(c,s) \text{ does not terminate iff } \nexists\, t.\ (c,\ s) \Rightarrow\ t\ ?$$

Needs a formal notion of nontermination to prove it.

The boon and bane of big steps

We cannot observe intermediate states/steps

Example problem:

(c,s) does not terminate iff $\nexists\, t.\ (c,\ s) \Rightarrow\ t$?

Needs a formal notion of nontermination to prove it.
Could be wrong if we have forgotten a $\Rightarrow$ rule.

Big-step semantics cannot directly describe

- nonterminating computations,

Big-step semantics cannot directly describe

- nonterminating computations,
- parallel computations.

Big-step semantics cannot directly describe

- nonterminating computations,
- parallel computations.

We need a finer grained semantics!

Small-step semantics

Concrete syntax:

$$(com, state) \rightarrow (com, state)$$

Small-step semantics

Concrete syntax:

$$(com, state) \to (com, state)$$

Intended meaning of $(c,\ s) \to (c',\ s')$:

Small-step semantics

Concrete syntax:

$$(com, state) \rightarrow (com, state)$$

Intended meaning of $(c,\ s) \rightarrow (c',\ s')$:

> *The first step in the execution of* c *in state* s
> *leaves a "remainder" command* c'
> *to be executed in state* s'*.*

Small-step semantics

Concrete syntax:

$$(com, state) \rightarrow (com, state)$$

Intended meaning of $(c,\ s) \rightarrow (c',\ s')$:

> *The first step in the execution of c in state s*
> *leaves a "remainder" command c'*
> *to be executed in state s'.*

Execution as finite or infinite reduction:

$$(c_1, s_1) \rightarrow (c_2, s_2) \rightarrow (c_3, s_3) \rightarrow \ldots$$

Terminology

- A pair (c, s) is called a *configuration*.

Terminology

- A pair (c, s) is called a *configuration*.
- If $cs \to cs'$ we say that cs *reduces* to cs'.

Terminology

- A pair (c,s) is called a *configuration*.
- If $cs \rightarrow cs'$ we say that cs *reduces* to cs'.
- A configuration cs is *final* iff $\nexists\, cs'.\ cs \rightarrow cs'$

The intention:

$$(SKIP,\ s) \text{ is final}$$

The intention:

$$(SKIP,\ s) \text{ is final}$$

Why?

$SKIP$ is the empty program.

The intention:

$$(SKIP,\ s) \ \text{ is final}$$

Why?

$SKIP$ is the empty program. Nothing more to be done.

Small-step rules

$$(x ::= a,\ s) \ \rightarrow$$

Small-step rules

$$(x ::= a,\ s) \ \rightarrow\ (SKIP,\ s(x := \ aval\ a\ s))$$

Small-step rules

$$(x ::= a,\ s) \ \rightarrow\ (SKIP,\ s(x := \ aval\ a\ s))$$

$$(SKIP;;\ c,\ s) \ \rightarrow$$

Small-step rules

$$(x ::= a,\ s) \ \rightarrow\ (SKIP,\ s(x := \ aval\ a\ s))$$

$$(SKIP;;\ c,\ s) \ \rightarrow\ (c,\ s)$$

Small-step rules

$$(x ::= a,\ s) \ \rightarrow\ (SKIP,\ s(x := \ aval\ a\ s))$$

$$(SKIP;;\ c,\ s) \ \rightarrow\ (c,\ s)$$

$$\frac{(c_1, s) \ \rightarrow\ (c_1', s')}{(c_1;;c_2, s) \ \rightarrow}$$

Small-step rules

$$(x ::= a,\ s) \ \rightarrow\ (SKIP,\ s(x := \ aval\ a\ s))$$

$$(SKIP;;\ c,\ s) \ \rightarrow\ (c,\ s)$$

$$\frac{(c_1, s) \ \rightarrow\ (c_1', s')}{(c_1;; c_2, s) \ \rightarrow\ (c_1';; c_2, s')}$$

Small-step rules

$$\frac{bval\ b\ s}{(IF\ b\ THEN\ c_1\ ELSE\ c_2, s)\ \rightarrow}$$

Small-step rules

$$\frac{bval\ b\ s}{(IF\ b\ THEN\ c_1\ ELSE\ c_2, s) \ \rightarrow\ (c_1, s)}$$

Small-step rules

$$\frac{bval\ b\ s}{(IF\ b\ THEN\ c_1\ ELSE\ c_2, s) \ \rightarrow\ (c_1, s)}$$

$$\frac{\neg\ bval\ b\ s}{(IF\ b\ THEN\ c_1\ ELSE\ c_2, s) \ \rightarrow\ (c_2, s)}$$

Small-step rules

$$\frac{bval\ b\ s}{(IF\ b\ THEN\ c_1\ ELSE\ c_2, s) \ \rightarrow\ (c_1, s)}$$

$$\frac{\neg\ bval\ b\ s}{(IF\ b\ THEN\ c_1\ ELSE\ c_2, s) \ \rightarrow\ (c_2, s)}$$

$$(WHILE\ b\ DO\ c,\ s) \ \rightarrow$$

Small-step rules

$$\frac{bval\ b\ s}{(IF\ b\ THEN\ c_1\ ELSE\ c_2, s) \rightarrow (c_1, s)}$$

$$\frac{\neg\ bval\ b\ s}{(IF\ b\ THEN\ c_1\ ELSE\ c_2, s) \rightarrow (c_2, s)}$$

$$(WHILE\ b\ DO\ c,\ s) \rightarrow$$
$$(IF\ b\ THEN\ c;;\ WHILE\ b\ DO\ c\ ELSE\ SKIP,\ s)$$

Small-step rules

$$\frac{bval\ b\ s}{(IF\ b\ THEN\ c_1\ ELSE\ c_2, s)\ \rightarrow\ (c_1, s)}$$

$$\frac{\neg\ bval\ b\ s}{(IF\ b\ THEN\ c_1\ ELSE\ c_2, s)\ \rightarrow\ (c_2, s)}$$

$$(WHILE\ b\ DO\ c,\ s)\ \rightarrow$$
$$(IF\ b\ THEN\ c;;\ WHILE\ b\ DO\ c\ ELSE\ SKIP,\ s)$$

Fact $(SKIP, s)$ is a final configuration.

Small-step examples

$$(''z'' ::= \ V \ ''x'';; \ ''x'' ::= \ V \ ''y'';; \ ''y'' ::= \ V \ ''z'', \ s) \rightarrow$$
$$\ldots$$

where $s = <''x'' := 3, \ ''y'' := 7, \ ''z'' := 5>$.

Small-step examples

$$(''z'' ::= V\ ''x'';;\ ''x'' ::= V\ ''y'';;\ ''y'' ::= V\ ''z'',\ s) \to$$
$$\ldots$$

where $s = <''x'' := 3,\ ''y'' := 7,\ ''z'' := 5>$.

$$(w,\ s_0) \to \ldots$$

where
$$\begin{aligned} w &= \ WHILE\ b\ DO\ c \\ b &= \ Less\ (V\ ''x'')\ (N\ 1) \\ c &= \ ''x'' ::= Plus\ (V\ ''x'')\ (N\ 1) \\ s_n &= \ <''x'' := n> \end{aligned}$$

`Small_Step.thy`

Semantics

Are big and small-step semantics equivalent?

From $\Rightarrow$ to $\rightarrow*$

From $\Rightarrow$ to $\rightarrow*$

Theorem $cs \Rightarrow t \implies cs \rightarrow* (SKIP,\ t)$

From $\Rightarrow$ to $\rightarrow*$

Theorem $cs \Rightarrow t \implies cs \rightarrow* (SKIP, t)$

Proof by rule induction

From $\Rightarrow$ to $\rightarrow*$

Theorem $cs \Rightarrow t \implies cs \rightarrow* (SKIP, t)$

Proof by rule induction (of course on $cs \Rightarrow t$)

From $\Rightarrow$ to $\rightarrow*$

Theorem $cs \Rightarrow t \implies cs \rightarrow* (SKIP, t)$

Proof by rule induction (of course on $cs \Rightarrow t$)

In two cases a lemma is needed:

From $\Rightarrow$ to $\rightarrow*$

Theorem $cs \Rightarrow t \implies cs \rightarrow* (SKIP,\ t)$

Proof by rule induction (of course on $cs \Rightarrow t$)
In two cases a lemma is needed:

Lemma
$(c_1,\ s) \rightarrow* (c_1{}',\ s') \implies (c_1;;\ c_2,\ s) \rightarrow* (c_1{}';;\ c_2,\ s')$

From $\Rightarrow$ to $\rightarrow*$

Theorem $cs \Rightarrow t \implies cs \rightarrow* (SKIP, t)$

Proof by rule induction (of course on $cs \Rightarrow t$)
In two cases a lemma is needed:

Lemma
$(c_1, s) \rightarrow* (c_1', s') \implies (c_1;; c_2, s) \rightarrow* (c_1';; c_2, s')$

Proof by rule induction.

From $\to*$ to $\Rightarrow$

From $\rightarrow*$ to $\Rightarrow$

Theorem $cs \rightarrow* (SKIP, t) \implies cs \Rightarrow t$

From $\rightarrow*$ to $\Rightarrow$

Theorem $cs \rightarrow* (SKIP, t) \implies cs \Rightarrow t$

Proof by rule induction on $cs \rightarrow* (SKIP, t)$.

From $\rightarrow*$ to $\Rightarrow$

Theorem $cs \rightarrow* (SKIP,\ t) \implies cs \Rightarrow t$

Proof by rule induction on $cs \rightarrow* (SKIP,\ t)$.
In the induction step a lemma is needed:

From $\rightarrow*$ to $\Rightarrow$

Theorem $cs \rightarrow* (SKIP,\ t) \implies cs \Rightarrow t$

Proof by rule induction on $cs \rightarrow* (SKIP,\ t)$.
In the induction step a lemma is needed:

Lemma $cs \rightarrow cs' \implies cs' \Rightarrow t \implies cs \Rightarrow t$

From $\rightarrow*$ to $\Rightarrow$

Theorem $cs \rightarrow* (SKIP, t) \implies cs \Rightarrow t$

Proof by rule induction on $cs \rightarrow* (SKIP, t)$.
In the induction step a lemma is needed:

Lemma $cs \rightarrow cs' \implies cs' \Rightarrow t \implies cs \Rightarrow t$

Proof by rule induction on $cs \rightarrow cs'$.

Equivalence

Corollary $cs \Rightarrow t \ \longleftrightarrow \ cs \rightarrow * \ (SKIP, \ t)$

Small_Step.thy

Equivalence of big and small

Can execution stop prematurely?

Can execution stop prematurely?

That is, are there any final configs except $(SKIP, s)$?

Can execution stop prematurely?

That is, are there any final configs except $(SKIP, s)$?

Lemma $\mathit{final}\ (c,\ s) \Longrightarrow c = SKIP$

Can execution stop prematurely?

That is, are there any final configs except $(SKIP, s)$?

Lemma $\mathit{final}\ (c,\ s) \Longrightarrow c = SKIP$

We prove the contrapositive

$c \neq SKIP \Longrightarrow \neg\ \mathit{final}(c, s)$

by induction on c.

Can execution stop prematurely?

That is, are there any final configs except $(SKIP,s)$?

Lemma $\mathit{final}\ (c,\ s) \Longrightarrow c = SKIP$

We prove the contrapositive

$c \neq SKIP \Longrightarrow \neg\ \mathit{final}(c,s)$

by induction on c.

- Case $c_1;;\ c_2$: by case distinction:

Can execution stop prematurely?

That is, are there any final configs except $(SKIP, s)$?

Lemma $\mathit{final}\ (c,\ s) \Longrightarrow c = SKIP$

We prove the contrapositive

$c \neq SKIP \Longrightarrow \neg\ \mathit{final}(c, s)$

by induction on c.

- Case $c_1;;\ c_2$: by case distinction:
  - $c_1 = SKIP$

Can execution stop prematurely?

That is, are there any final configs except $(SKIP, s)$?

Lemma $\mathit{final}\ (c,\ s) \Longrightarrow c = SKIP$

We prove the contrapositive

$c \neq SKIP \Longrightarrow \neg\ \mathit{final}(c, s)$

by induction on c.

- Case $c_1;;\ c_2$: by case distinction:
  - $c_1 = SKIP \Longrightarrow \neg\ \mathit{final}\ (c_1;;\ c_2,\ s)$

Can execution stop prematurely?

That is, are there any final configs except $(SKIP, s)$?

Lemma $\mathit{final}\ (c,\ s) \Longrightarrow c = SKIP$

We prove the contrapositive

$c \neq SKIP \Longrightarrow \neg\ \mathit{final}(c, s)$

by induction on c.

- Case $c_1;;\ c_2$: by case distinction:
  - $c_1 = SKIP \Longrightarrow \neg\ \mathit{final}\ (c_1;;\ c_2,\ s)$
  - $c_1 \neq SKIP$

Can execution stop prematurely?

That is, are there any final configs except $(SKIP, s)$?

Lemma $\mathit{final}\ (c,\ s) \Longrightarrow c = SKIP$

We prove the contrapositive

$c \neq SKIP \Longrightarrow \neg\ \mathit{final}(c, s)$

by induction on c.

- Case $c_1;;\ c_2$: by case distinction:
  - $c_1 = SKIP \Longrightarrow \neg\ \mathit{final}\ (c_1;;\ c_2,\ s)$
  - $c_1 \neq SKIP \Longrightarrow \neg\ \mathit{final}\ (c_1,\ s)$

Can execution stop prematurely?

That is, are there any final configs except $(SKIP, s)$?

Lemma $\mathit{final}\ (c,\ s) \Longrightarrow c = SKIP$

We prove the contrapositive

$c \neq SKIP \Longrightarrow \neg\ \mathit{final}(c, s)$

by induction on c.

- Case $c_1;;\ c_2$: by case distinction:
  - $c_1 = SKIP \Longrightarrow \neg\ \mathit{final}\ (c_1;;\ c_2,\ s)$
  - $c_1 \neq SKIP \Longrightarrow \neg\ \mathit{final}\ (c_1,\ s)$ (by IH)

Can execution stop prematurely?

That is, are there any final configs except $(SKIP, s)$?

Lemma $\mathit{final}\ (c,\ s) \Longrightarrow c = SKIP$

We prove the contrapositive

$$c \neq SKIP \Longrightarrow \neg\ \mathit{final}(c, s)$$

by induction on c.

- Case $c_1;;\ c_2$: by case distinction:
  - $c_1 = SKIP \Longrightarrow \neg\ \mathit{final}\ (c_1;;\ c_2,\ s)$
  - $c_1 \neq SKIP \Longrightarrow \neg\ \mathit{final}\ (c_1,\ s)$ (by IH)
    $\Longrightarrow \neg\ \mathit{final}\ (c_1;;\ c_2,\ s)$

Can execution stop prematurely?

That is, are there any final configs except $(SKIP, s)$?

Lemma $\mathit{final}\ (c,\ s) \Longrightarrow c = SKIP$

We prove the contrapositive

$c \neq SKIP \Longrightarrow \neg\ \mathit{final}(c, s)$

by induction on c.

- Case $c_1;;\ c_2$: by case distinction:
  - $c_1 = SKIP \Longrightarrow \neg\ \mathit{final}\ (c_1;;\ c_2,\ s)$
  - $c_1 \neq SKIP \Longrightarrow \neg\ \mathit{final}\ (c_1,\ s)$ (by IH)
    $\Longrightarrow \neg\ \mathit{final}\ (c_1;;\ c_2,\ s)$
- Remaining cases: trivial or easy

By rule inversion: $(SKIP,\ s) \rightarrow ct \Longrightarrow False$

By rule inversion: $(SKIP,\ s) \to ct \Longrightarrow False$

Together:

Corollary $\mathit{final}\ (c,\ s) = (c = SKIP)$

Infinite executions

$\Rightarrow$ yields final state iff $\rightarrow$ terminates

Infinite executions

$\Rightarrow$ yields final state iff $\rightarrow$ terminates

Lemma $(\exists\, t.\ cs \Rightarrow t) = (\exists\, cs'.\ cs \rightarrow\!* \ cs' \wedge \mathit{final}\ cs')$

Infinite executions

$\Rightarrow$ yields final state iff $\rightarrow$ terminates

Lemma $(\exists\, t.\; cs \Rightarrow\; t) = (\exists\, cs'.\; cs \rightarrow\!*\; cs' \wedge \mathit{final}\; cs')$

Proof: $(\exists\, t.\; cs \Rightarrow\; t)$

Infinite executions

$\Rightarrow$ yields final state iff $\rightarrow$ terminates

Lemma $(\exists\, t.\ cs \Rightarrow\ t) = (\exists\, cs'.\ cs \rightarrow\!*\ cs' \wedge \mathit{final}\ cs')$

Proof: $(\exists\, t.\ cs \Rightarrow\ t)$

$= \ (\exists\, t.\ cs \rightarrow\!*\ (\mathit{SKIP},t))$

Infinite executions

$\Rightarrow$ yields final state iff $\rightarrow$ terminates

Lemma $(\exists\, t.\ cs \Rightarrow t) = (\exists\, cs'.\ cs \rightarrow\!* \ cs' \wedge \mathit{final}\ cs')$

Proof: $(\exists\, t.\ cs \Rightarrow t)$

$= (\exists\, t.\ cs \rightarrow\!* \ (\mathit{SKIP}, t))$

(by big = small)

Infinite executions

$\Rightarrow$ yields final state iff $\rightarrow$ terminates

Lemma $(\exists\, t.\ cs \Rightarrow t) = (\exists\, cs'.\ cs \rightarrow\!* \ cs' \wedge \mathit{final}\ cs')$

Proof: $(\exists\, t.\ cs \Rightarrow t)$

$= (\exists\, t.\ cs \rightarrow\!* \ (SKIP, t))$

(by big = small)

$= (\exists\, cs'.\ cs \rightarrow\!* \ cs' \wedge \mathit{final}\ cs')$

Infinite executions

$\Rightarrow$ yields final state iff $\rightarrow$ terminates

Lemma $(\exists\, t.\ cs \Rightarrow t) = (\exists\, cs'.\ cs \rightarrow\!*\ cs' \wedge \mathit{final}\ cs')$

Proof: $(\exists\, t.\ cs \Rightarrow t)$

$= (\exists\, t.\ cs \rightarrow\!*\ (\mathit{SKIP},t))$

(by big = small)

$= (\exists\, cs'.\ cs \rightarrow\!*\ cs' \wedge \mathit{final}\ cs')$

(by final = SKIP)

Infinite executions

$\Rightarrow$ yields final state iff $\rightarrow$ terminates

Lemma $(\exists\, t.\ cs \Rightarrow t) = (\exists\, cs'.\ cs \rightarrow\!* \ cs' \wedge \mathit{final}\ cs')$

Proof: $(\exists\, t.\ cs \Rightarrow t)$

$= (\exists\, t.\ cs \rightarrow\!* \ (\mathit{SKIP},t))$

(by big = small)

$= (\exists\, cs'.\ cs \rightarrow\!* \ cs' \wedge \mathit{final}\ cs')$

(by final = SKIP)

Equivalent:

$\Rightarrow$ does not yield final state iff $\rightarrow$ does not terminate

May versus Must

$\rightarrow$ is deterministic:

May versus Must

$\rightarrow$ is deterministic:

Lemma $cs \rightarrow cs' \implies cs \rightarrow cs'' \implies cs'' = cs'$

May versus Must

$\to$ is deterministic:

Lemma $cs \to cs' \implies cs \to cs'' \implies cs'' = cs'$

(Proof by rule induction)

May versus Must

$\rightarrow$ is deterministic:

Lemma $cs \rightarrow cs' \implies cs \rightarrow cs'' \implies cs'' = cs'$
(Proof by rule induction)

Therefore: no difference between

may terminate (there is a terminating $\rightarrow$ path)
must terminate (all $\rightarrow$ paths terminate)

May versus Must

$\rightarrow$ is deterministic:

Lemma $cs \rightarrow cs' \implies cs \rightarrow cs'' \implies cs'' = cs'$
(Proof by rule induction)

Therefore: no difference between

may terminate (there is a terminating $\rightarrow$ path)

must terminate (all $\rightarrow$ paths terminate)

Therefore: $\Rightarrow$ correctly reflects termination behaviour.

May versus Must

$\rightarrow$ is deterministic:

Lemma $cs \rightarrow cs' \implies cs \rightarrow cs'' \implies cs'' = cs'$
(Proof by rule induction)

Therefore: no difference between

may terminate (there is a terminating $\rightarrow$ path)

must terminate (all $\rightarrow$ paths terminate)

Therefore: $\Rightarrow$ correctly reflects termination behaviour.

With nondeterminism: may have both $cs \Rightarrow t$ and a nonterminating reduction $cs \rightarrow cs' \rightarrow \ldots$

Chapter 8

Hoare Logic

4 Weakest Preconditions

4 Weakest Preconditions

4 Weakest Preconditions

Introduction

We have proved functional programs correct

We have proved functional programs correct

We have modeled semantics of imperative languages

We have proved functional programs correct

We have modeled semantics of imperative languages

But how do we prove imperative programs correct?

An example program:

$$
\begin{array}{l}
\mathit{program}\ \mathit{exp}\ \{ \\
\quad a := 1 \\
\quad \mathit{while}\ (0{<}n)\ \mathit{do}\ \{ \\
\quad\quad a := a + a; \\
\quad\quad n := n - 1 \\
\quad \} \\
\}
\end{array}
$$

An example program:

$$
\begin{array}{l}
\textit{program exp} \ \{ \\
\quad a := 1 \\
\quad \textit{while} \ (0 < n) \ \textit{do} \ \{ \\
\quad\quad a := a + a; \\
\quad\quad n := n - 1 \\
\quad \} \\
\}
\end{array}
$$

At the end of the execution, variable a should contain 2^n,

An example program:

$$
\begin{array}{l}
\textit{program exp } \{ \\
\quad a := 1 \\
\quad \textit{while } (0 < n) \textit{ do } \{ \\
\quad\quad a := a + a; \\
\quad\quad n := n - 1 \\
\quad \} \\
\}
\end{array}
$$

At the end of the execution, variable a should contain 2^n, where n is the original value of variable n!

An example program:

$$
\begin{array}{l}
\textit{program exp} \ \{ \\
\quad a := 1 \\
\quad \textit{while} \ (0 < n) \ \textit{do} \ \{ \\
\qquad a := a + a; \\
\qquad n := n - 1 \\
\quad \} \\
\}
\end{array}
$$

At the end of the execution, variable a should contain 2^n, where n is the original value of variable n!
and $0 \leq n$!

In general: If *precondition* holds for initial state
then, program terminates, and
final state satisfies *postcondition*

In general: If *precondition* holds for initial state
then, program terminates, and
final state satisfies *postcondition*

Formally

In general: If *precondition* holds for initial state
then, program terminates, and
final state satisfies *postcondition*

Formally?

In general: If *precondition* holds for initial state
then, program terminates, and
final state satisfies *postcondition*

Formally?

$$P\ s \Longrightarrow \exists\, t.\ (c,\ s) \Rightarrow\ t \wedge\ Q\ t$$

In general: If *precondition* holds for initial state
then, program terminates, and
final state satisfies *postcondition*

Formally?

$$P\ s \Longrightarrow \exists\, t.\ (c,\ s) \Rightarrow\ t \wedge\ Q\ t$$

The RHS of this implication is called *weakest precondition*

$$wp\ c\ Q\ s \equiv \exists\, t.\ (c,\ s) \Rightarrow\ t \wedge\ Q\ t$$

In general: If *precondition* holds for initial state then, program terminates, and final state satisfies *postcondition*

Formally?

$$P\ s \Longrightarrow \exists\, t.\ (c,\ s) \Rightarrow\ t \wedge\ Q\ t$$

The RHS of this implication is called *weakest precondition*

$$wp\ c\ Q\ s \equiv \exists\, t.\ (c,\ s) \Rightarrow\ t \wedge\ Q\ t$$

Weakest condition on state, such that program c will satisfy postcondition Q.

Some obvious facts:

Some obvious facts:

Consequence rule:

$$\llbracket wp\ c\ P\ s;\ \bigwedge s.\ P\ s \Longrightarrow Q\ s \rrbracket \Longrightarrow wp\ c\ Q\ s$$

Some obvious facts:

Consequence rule:

$$\llbracket wp\ c\ P\ s;\ \bigwedge s.\ P\ s \Longrightarrow Q\ s \rrbracket \Longrightarrow wp\ c\ Q\ s$$

wp of equivalent programs is equal

$$c \sim c' \Longrightarrow wp\ c = wp\ c'$$

Correctness of exp

Correctness of exp?

Correctness of exp?

$$s \ ''n'' \le 0 \Longrightarrow wp \ exp \ (\lambda s'. \ s' \ ''a'' = 2^{\mathsf{nat} \ (s \ ''n'')}) \ s$$

Correctness of exp?

$$s\ ''n'' \leq 0 \Longrightarrow wp\ exp\ (\lambda s'.\ s'\ ''a'' = 2^{\mathsf{nat}\ (s\ ''n'')})\ s$$

$nat{::}int \Rightarrow nat$ required b/c $(\hat{\ }){::}'a \Rightarrow nat \Rightarrow 'a$ only defined on nat.

Correctness of exp?

$$s\ ''n'' \overset{>}{\cancel{\le}} 0 \Longrightarrow wp\ exp\ (\lambda s'.\ s'\ ''a'' = 2^{nat\ (s\ ''n'')})\ s$$

$nat{::}int \Rightarrow nat$ required b/c $(\hat{\ }){::}'a \Rightarrow nat \Rightarrow 'a$ only defined on nat.

In general: $P\ s \Longrightarrow wp\ c\ Q\ s$

How to prove correctness of programs?

$$P\ s \Longrightarrow wp\ c\ Q\ s$$

How to prove correctness of programs?

$$P\ s \Longrightarrow wp\ c\ Q\ s$$

$$wp\ SKIP\ Q\ s =$$

How to prove correctness of programs?

$$P\ s \Longrightarrow wp\ c\ Q\ s$$

$$wp\ SKIP\ Q\ s = Q\ s$$

How to prove correctness of programs?

$P\ s \Longrightarrow wp\ c\ Q\ s$

$wp\ SKIP\ Q\ s = Q\ s$

$wp\ (x ::= a)\ Q\ s =$

How to prove correctness of programs?

$$P\ s \Longrightarrow wp\ c\ Q\ s$$

$$wp\ SKIP\ Q\ s = Q\ s$$
$$wp\ (x ::= a)\ Q\ s = Q\ (s(x := aval\ a\ s))$$

How to prove correctness of programs?

$$P\ s \Longrightarrow wp\ c\ Q\ s$$

$$wp\ SKIP\ Q\ s = Q\ s$$
$$wp\ (x ::= a)\ Q\ s = Q\ (s(x := aval\ a\ s))$$
$$wp\ (c_1;;\ c_2)\ Q\ s =$$

How to prove correctness of programs?

$$P\ s \Longrightarrow wp\ c\ Q\ s$$

$$wp\ SKIP\ Q\ s = Q\ s$$
$$wp\ (x ::= a)\ Q\ s = Q\ (s(x := aval\ a\ s))$$
$$wp\ (c_1;;\ c_2)\ Q\ s = wp\ c_1\ (wp\ c_2\ Q)\ s$$

How to prove correctness of programs?

$$P\ s \Longrightarrow wp\ c\ Q\ s$$

$$wp\ SKIP\ Q\ s =\ Q\ s$$
$$wp\ (x ::=\ a)\ Q\ s =\ Q\ (s(x :=\ aval\ a\ s))$$
$$wp\ (c_1;;\ c_2)\ Q\ s =\ wp\ c_1\ (wp\ c_2\ Q)\ s$$
$$wp\ (IF\ b\ THEN\ c_1\ ELSE\ c_2)\ Q\ s$$
$$=$$

How to prove correctness of programs?

$$P\ s \Longrightarrow wp\ c\ Q\ s$$

$$wp\ SKIP\ Q\ s = Q\ s$$
$$wp\ (x ::= a)\ Q\ s = Q\ (s(x := aval\ a\ s))$$
$$wp\ (c_1;;\ c_2)\ Q\ s = wp\ c_1\ (wp\ c_2\ Q)\ s$$
$$wp\ (IF\ b\ THEN\ c_1\ ELSE\ c_2)\ Q\ s$$
$$= \textit{if}\ bval\ b\ s\ \textsf{then}\ wp\ c_1\ Q\ s\ \textsf{else}\ wp\ c_2\ Q\ s$$

How to prove correctness of programs?

$$P\ s \Longrightarrow wp\ c\ Q\ s$$

$$wp\ \mathit{SKIP}\ Q\ s = Q\ s$$

$$wp\ (x ::= a)\ Q\ s = Q\ (s(x := aval\ a\ s))$$

$$wp\ (c_1;;\ c_2)\ Q\ s = wp\ c_1\ (wp\ c_2\ Q)\ s$$

$$wp\ (\mathit{IF}\ b\ \mathit{THEN}\ c_1\ \mathit{ELSE}\ c_2)\ Q\ s$$
$$= \textit{if}\ bval\ b\ s\ \textit{then}\ wp\ c_1\ Q\ s\ \textit{else}\ wp\ c_2\ Q\ s$$

Reasoning along syntax of program!

That was easy!

That was easy! But what about $While$?

That was easy! But what about $While$?

$$wp\ (WHILE\ b\ DO\ c)\ Q\ s$$
$$=$$

That was easy! But what about $While$?

$wp\ (WHILE\ b\ DO\ c)\ Q\ s$
$=$*if* $bval\ b\ s$ *then* $wp\ c\ (wp\ (WHILE\ b\ DO\ c)\ Q)\ s$ *else*
$Q\ s$

That was easy! But what about $While$?

$wp\ (WHILE\ b\ DO\ c)\ Q\ s$
$=$*if* $bval\ b\ s$ *then* $wp\ c\ (wp\ (WHILE\ b\ DO\ c)\ Q)\ s$ *else* $Q\ s$

Unfolding will continue forever!

That was easy! But what about $While$?

$wp\ (WHILE\ b\ DO\ c)\ Q\ s$
$=$*if* $bval\ b\ s$ *then* $wp\ c\ (wp\ (WHILE\ b\ DO\ c)\ Q)\ s$ *else* $Q\ s$

Unfolding will continue forever!

Obviously, need some inductive argument!

That was easy! But what about While?

$wp\ (WHILE\ b\ DO\ c)\ Q\ s$
$=$*if* $bval\ b\ s$ *then* $wp\ c\ (wp\ (WHILE\ b\ DO\ c)\ Q)\ s$ *else* $Q\ s$

Unfolding will continue forever!

Obviously, need some inductive argument!

But, let's get less ambitious (for first)

Weakest liberal precondition

$$wlp\ c\ Q\ s \equiv \forall\, t.\ (c,\ s) \Rightarrow\ t \longrightarrow\ Q\ t$$

Weakest liberal precondition

$$wlp\ c\ Q\ s \equiv \forall\, t.\ (c,\ s) \Rightarrow\ t \longrightarrow\ Q\ t$$

If c terminates on s, then new state satisfies Q

Weakest liberal precondition

$$wlp\ c\ Q\ s \equiv \forall\, t.\ (c,\ s) \Rightarrow t \longrightarrow Q\ t$$

If c terminates on s, then new state satisfies Q

Cannot reason about termination. This is called *partial correctness*.

Some obvious facts:

$$c \sim c' \Longrightarrow wlp\ c = wlp\ c'$$
$$[\![wlp\ c\ P\ s;\ \bigwedge s.\ P\ s \Longrightarrow Q\ s]\!] \Longrightarrow wlp\ c\ Q\ s$$

Some obvious facts:

$$c \sim c' \Longrightarrow wlp\ c = wlp\ c'$$
$$[\![wlp\ c\ P\ s;\ \bigwedge s.\ P\ s \Longrightarrow Q\ s]\!] \Longrightarrow wlp\ c\ Q\ s$$

Relation between wp and wlp

$$wp\ c\ Q\ s \Longrightarrow wlp\ c\ Q\ s$$
$$wlp\ c\ Q\ s \wedge (c,\ s) \Rightarrow t \Longrightarrow wp\ c\ Q\ s$$

Some obvious facts:

$$c \sim c' \Longrightarrow wlp\ c = wlp\ c'$$
$$[\![wlp\ c\ P\ s;\ \bigwedge s.\ P\ s \Longrightarrow Q\ s]\!] \Longrightarrow wlp\ c\ Q\ s$$

Relation between wp and wlp

$$wp\ c\ Q\ s \Longrightarrow wlp\ c\ Q\ s$$
$$wlp\ c\ Q\ s \wedge (c,\ s) \Rightarrow t \Longrightarrow wp\ c\ Q\ s$$

Unfold rules still hold:

$$wlp\ SKIP\ Q\ s = Q\ s$$
$$wlp\ (x ::= a)\ Q\ s = Q\ (s(x := aval\ a\ s))$$
$$wlp\ (c_1;;\ c_2)\ Q\ s = wlp\ c_1\ (wlp\ c_2\ Q)\ s$$
$$wlp\ (IF\ b\ THEN\ c_1\ ELSE\ c_2)\ Q\ s =$$
$$(\textsf{\textit{if}}\ bval\ b\ s\ \textsf{\textit{then}}\ wlp\ c_1\ Q\ s\ \textsf{\textit{else}}\ wlp\ c_2\ Q\ s)$$

$wlp\ (WHILE\ b\ DO\ c)\ Q\ s =$
$(\textsf{\textit{if}}\ bval\ b\ s\ \textsf{\textit{then}}\ wlp\ c\ (wlp\ (WHILE\ b\ DO\ c)\ Q)\ s\ \textsf{\textit{else}}$
$Q\ s)$

$wlp\ (WHILE\ b\ DO\ c)\ Q\ s =$
$(\textsf{if}\ bval\ b\ s\ \textsf{then}\ wlp\ c\ (wlp\ (WHILE\ b\ DO\ c)\ Q)\ s\ \textsf{else}$
$Q\ s)$

Lets try to find predicate I, such that

$\bigwedge s.\ I\ s \Longrightarrow \textsf{if}\ bval\ b\ s\ \textsf{then}\ wp\ c\ I\ s\ \textsf{else}\ Q\ s$

$wlp\ (WHILE\ b\ DO\ c)\ Q\ s =$
$(\textsf{\textit{if}}\ bval\ b\ s\ \textsf{\textit{then}}\ wlp\ c\ (wlp\ (WHILE\ b\ DO\ c)\ Q)\ s\ \textsf{\textit{else}}$
$Q\ s)$

Lets try to find predicate I, such that

$\bigwedge s.\ I\ s \Longrightarrow \textsf{\textit{if}}\ bval\ b\ s\ \textsf{\textit{then}}\ wp\ c\ I\ s\ \textsf{\textit{else}}\ Q\ s$

and I holds for start state.

$wlp\ (WHILE\ b\ DO\ c)\ Q\ s =$
$(\textit{if}\ bval\ b\ s\ \textit{then}\ wlp\ c\ (wlp\ (WHILE\ b\ DO\ c)\ Q)\ s\ \textit{else}\ Q\ s)$

Lets try to find predicate I, such that

$\bigwedge s.\ I\ s \Longrightarrow \textit{if}\ bval\ b\ s\ \textit{then}\ wp\ c\ I\ s\ \textit{else}\ Q\ s$

and I holds for start state.

Intuition: I holds initially, is preserved by iteration, and implies Q at end of loop.

$wlp\ (WHILE\ b\ DO\ c)\ Q\ s =$
$(\textit{if}\ bval\ b\ s\ \textit{then}\ wlp\ c\ (wlp\ (WHILE\ b\ DO\ c)\ Q)\ s\ \textit{else}$
$Q\ s)$

Lets try to find predicate I, such that

$\bigwedge s.\ I\ s \Longrightarrow \textit{if}\ bval\ b\ s\ \textit{then}\ wp\ c\ I\ s\ \textit{else}\ Q\ s$

and I holds for start state.

Intuition: I holds initially, is preserved by iteration, and implies Q at end of loop. I is called *loop invariant*

While-rule for partial correctness

$$\llbracket I\ s_0;\ \bigwedge s.\ I\ s \Longrightarrow \textit{if}\ bval\ b\ s\ \textit{then}\ wlp\ c\ I\ s\ \textit{else}\ Q\ s\rrbracket$$
$$\Longrightarrow wlp\ (WHILE\ b\ DO\ c)\ Q\ s_0$$

Wp_Demo.thy

Weakest Precondition

Now we can start proving programs ...

Now we can start proving programs ...

$$P\ s \Longrightarrow wlp\ c\ Q\ s$$

Now we can start proving programs ...

$P\ s \Longrightarrow wlp\ c\ Q\ s$

If $c = \ WHILE\ \_\ DO\ \_$, provide invariant and apply while rule

Now we can start proving programs ...

$$P\ s \Longrightarrow wlp\ c\ Q\ s$$

If $c = \ WHILE\ \_\ DO\ \_$, provide invariant and apply while rule

Otherwise, use unfold rules.

Now we can start proving programs ...

$$P\ s \Longrightarrow wlp\ c\ Q\ s$$

If $c = \ WHILE\ \_\ DO\ \_$, provide invariant and apply while rule

Otherwise, use unfold rules.

Iterate, until all wlps gone!

wlp_if_eq and wlp_whileI' produce $if{-}then{-}else$

wlp_if_eq and wlp_whileI' produce $if{-}then{-}else$ which we have to split.

wlp_if_eq and wlp_whileI' produce $if{-}then{-}else$ which we have to split.

Combine rule with splitting!

Wp_Demo.thy

Proving Partial Correctness

But how about termination?

But how about termination?

An (ordering) relation $<$ is *well-founded*, iff every non-empty set has a minimal element.

But how about termination?

An (ordering) relation $<$ is *well-founded*, iff every non-empty set has a minimal element.

Equivalently: No infinite sequence with $x_1 > x_2 > \ldots$

But how about termination?

An (ordering) relation $<$ is *well-founded*, iff every non-empty set has a minimal element.

Equivalently: No infinite sequence with $x_1 > x_2 > \dots$

Well-foundedness implies induction principle

But how about termination?

An (ordering) relation $<$ is *well-founded*, every non-empty set has a minimal element.

Equivalently: No infinite sequence with $x_1 > x_2 > \ldots$

Well-foundedness implies induction principle

$$\frac{wf\ r \qquad \bigwedge x.\ \dfrac{\forall\, y.\ (y,\ x) \in r \longrightarrow P\ y}{P\ x}}{P\ a}$$

`Wellfounded_Demo.thy`

For while loop: Find wf relation $<$ such that state decreases in each iteration

For while loop: Find wf relation $<$ such that state decreases in each iteration

$$\bigwedge s.\ I\ s \Longrightarrow \textit{if}\ bval\ b\ s\ \textit{then}\ wp\ c\ (\lambda s'.\ I\ s' \wedge\ s' <\ s)\ s\ \textit{else}\ Q\ s$$

For while loop: Find wf relation $<$ such that state decreases in each iteration

$\bigwedge s.\ I\ s \Longrightarrow$ *if* $bval\ b\ s$ **then** $wp\ c\ (\lambda s'.\ I\ s' \wedge s' < s)\ s$
else $Q\ s$

Then use wf-induction to prove:

$[\![wf\ R;\ I\ s_0;$
$\bigwedge s.\ I\ s \Longrightarrow$ *if* $bval\ b\ s$ **then** $wp\ c\ (\lambda s'.\ I\ s' \wedge (s',\ s) \in$
$R)\ s$ *else* $Q\ s]\!]$
$\Longrightarrow wp\ (WHILE\ b\ DO\ c)\ Q\ s_0$

Or, equivalently

assumes WF: $wf\ R$
assumes $INIT$: $I\ s_0$
assumes $STEP$: $\bigwedge s.\ \llbracket\ I\ s;\ bval\ b\ s\ \rrbracket$
$\implies wp\ c\ (\lambda s'.\ I\ s' \wedge (s',s) \in R)\ s$
assumes $FINAL$: $\bigwedge s.\ \llbracket\ I\ s;\ \neg bval\ b\ s\ \rrbracket \implies Q\ s$
shows $wp\ (WHILE\ b\ DO\ c)\ Q\ s_0$

Or, equivalently

assumes WF: $wf\ R$
assumes $INIT$: $I\ s_0$
assumes $STEP$: $\bigwedge s.\ [\![\ I\ s;\ bval\ b\ s\ ]\!]$
$\Longrightarrow wp\ c\ (\lambda s'.\ I\ s' \wedge (s',s) \in R)\ s$
assumes $FINAL$: $\bigwedge s.\ [\![\ I\ s;\ \neg bval\ b\ s\ ]\!] \Longrightarrow Q\ s$
shows $wp\ (WHILE\ b\ DO\ c)\ Q\ s_0$

Now we can prove total correctness ...

Wp_Demo.thy

Total Correctness