Proof Assistants

Thomas Bauereiss Leo Stefanesco

Department of Computer Science and Technology
University of Cambridge

Michaelmas 2026

Chapter 7

Semantics of 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

Commands

Concrete syntax:

$$com \ ::= \ \texttt{SKIP}$$

Commands

Concrete syntax:

$$\begin{array}{rcl} com & ::= & \texttt{SKIP} \\ & | & string \texttt{ ::= } aexp \end{array}$$

Commands

Concrete syntax:

$$
\begin{array}{rcl}
com & ::= & \texttt{SKIP} \\
& | & string \texttt{ ::= } aexp \\
& | & com \texttt{ ;; } com
\end{array}
$$

Commands

Concrete syntax:

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

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:

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

Com.thy

1. IMP Commands
2. Big-Step Semantics
3. Small-Step Semantics

Big-step semantics

Concrete syntax:

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

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

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

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

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

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

Rule inversion

What can we deduce from

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

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

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

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

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

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 $(c_2,\ s_2) \Rightarrow s_3$

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).

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*

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

(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 $w = WHILE\ b\ DO\ c$

$w' = IF\ b\ THEN\ c;;\ w\ ELSE\ SKIP$

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

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

$$\longleftrightarrow$$

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

$$\vee$$

$$\neg\ bval\ b\ s \wedge t = s$$

$$\longleftrightarrow$$

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

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) 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!

1. IMP Commands
2. Big-Step Semantics
3. Small-Step Semantics

Small-step semantics

Concrete syntax:

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

Small-step semantics

Concrete syntax:

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

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

Small-step semantics

Concrete syntax:

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

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

> *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:

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

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

> *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:

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

Terminology

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

Terminology

- A pair (c,s) is called a *configuration*.
- If $cs \rightarrow cs'$ 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:

$(SKIP,\ s)$ is final

The intention:

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

Why?

$SKIP$ is the empty program.

The intention:

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

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.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 $\rightarrow*$ 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 $cs \Rightarrow t \iff cs \rightarrow* (SKIP,\ t)$

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 $final\ (c,\ s) \Longrightarrow c = SKIP$

Can execution stop prematurely?

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

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

We prove the contrapositive

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

by induction on c.

Can execution stop prematurely?

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

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

We prove the contrapositive

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

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 $final\ (c,\ s) \Longrightarrow c = SKIP$

We prove the contrapositive

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

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 $final\ (c,\ s) \Longrightarrow c = SKIP$

We prove the contrapositive

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

by induction on c.

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

Can execution stop prematurely?

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

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

We prove the contrapositive

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

by induction on c.

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

Can execution stop prematurely?

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

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

We prove the contrapositive

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

by induction on c.

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

Can execution stop prematurely?

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

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

We prove the contrapositive

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

by induction on c.

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

Can execution stop prematurely?

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

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

We prove the contrapositive

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

by induction on c.

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

Can execution stop prematurely?

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

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

We prove the contrapositive

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

by induction on c.

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

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

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

Together:

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

Infinite executions

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

Infinite executions

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

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

Infinite executions

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

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

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

Infinite executions

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

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

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

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

Infinite executions

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

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

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

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

(by big $=$ small)

Infinite executions

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

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

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

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

(by big $=$ small)

$= (\exists\, cs'.\ cs \rightarrow* cs' \wedge final\ cs')$

Infinite executions

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

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

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

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

(by big $=$ small)

$= (\exists\, cs'.\ cs \rightarrow* cs' \wedge final\ cs')$

(by final $=$ SKIP)

Infinite executions

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

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

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

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

(by big $=$ small)

$= (\exists\, cs'.\ cs \rightarrow* cs' \wedge final\ cs')$

(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

$\rightarrow$ is deterministic:

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

(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 $cs \rightarrow cs' \rightarrow \ldots$