NICTA Advanced Course

Theorem Proving

Principles, Techniques, Applications

$$\{P\} \ldots \{Q\}$$

CONTENT

LAST TIME

- ➜ Recdef
- ➜ More induction
- ➜ Well founded orders
- ➜ Well founded recursion
- ➜ Calculations: also/finally
- ➜ [trans]-rules

A Crash Course in Semantics

IMP - A SMALL IMPERATIVE LANGUAGE

Commands:

datatype com	=	SKIP	
	$\mid$	Assign loc aexp	$(\_ := \_)$
	$\mid$	Semi com com	$(\_;\ \_)$
	$\mid$	Cond bexp com com	(IF _ THEN _ ELSE _)
	$\mid$	While bexp com	(WHILE _ DO _ OD)

IMP - A SMALL IMPERATIVE LANGUAGE

Commands:

datatype com	=	SKIP	
	$\mid$	Assign loc aexp	$(\_ := \_)$
	$\mid$	Semi com com	$(\_;\ \_)$
	$\mid$	Cond bexp com com	(IF _ THEN _ ELSE _)
	$\mid$	While bexp com	(WHILE _ DO _ OD)

types loc = string

types state = loc $\Rightarrow$ nat

Commands:

datatype com	=	SKIP	
	$\vert$	Assign loc aexp	(_ := _)
	$\vert$	Semi com com	(_; _)
	$\vert$	Cond bexp com com	(IF _ THEN _ ELSE _)
	$\vert$	While bexp com	(WHILE _ DO _ OD)

types loc	=	string
types state	=	loc $\Rightarrow$ nat
types aexp	=	state $\Rightarrow$ nat
types bexp	=	state $\Rightarrow$ bool

EXAMPLE PROGRAM

Usual syntax:

$B := 1;$

WHILE $A \neq 0$ DO

$\quad B := B * A;$

$\quad A := A - 1$

OD

EXAMPLE PROGRAM

Usual syntax:

$$B := 1;$$
WHILE $A \neq 0$ DO
$$B := B * A;$$
$$A := A - 1$$
OD

Expressions are functions from state to bool or nat:

$$B := (\lambda\sigma.\ 1);$$
WHILE $(\lambda\sigma.\ \sigma\ A \neq 0)$ DO
$$B := (\lambda\sigma.\ \sigma\ B * \sigma\ A);$$
$$A := (\lambda\sigma.\ \sigma\ A - 1)$$
OD

What does it do?

So far we have defined:

WHAT DOES IT DO?

So far we have defined:

➜ **Syntax** of commands and expressions

What does it do?

So far we have defined:

➜ **Syntax** of commands and expressions

➜ **State** of programs (function from variables to values)

Now we need:

WHAT DOES IT DO?

So far we have defined:

- ➜ **Syntax** of commands and expressions
- ➜ **State** of programs (function from variables to values)

Now we need: the meaning (semantics) of programs

How to define execution of a program?

WHAT DOES IT DO?

So far we have defined:

➜ **Syntax** of commands and expressions

➜ **State** of programs (function from variables to values)

Now we need: the meaning (semantics) of programs

How to define execution of a program?

➜ A wide field of its own (visit a semantics course!)

WHAT DOES IT DO?

So far we have defined:

➜ **Syntax** of commands and expressions

➜ **State** of programs (function from variables to values)

Now we need: the meaning (semantics) of programs

How to define execution of a program?

➜ A wide field of its own (visit a semantics course!)

➜ Some choices:

- Operational (inductive relations, big step, small step)
- Denotational (programs as functions on states, state transformers)
- Axiomatic (pre-/post conditions, Hoare logic)

Structural Operational Semantics

$$\frac{}{\langle \mathsf{SKIP}, \sigma \rangle \longrightarrow \sigma}$$

Structural Operational Semantics

$$\overline{\langle \mathsf{SKIP}, \sigma \rangle \longrightarrow \sigma}$$

$$\overline{\langle \mathsf{x} := \mathsf{e}, \sigma \rangle \longrightarrow \qquad\qquad\qquad}$$

Structural Operational Semantics

$$\frac{}{\langle \mathsf{SKIP}, \sigma\rangle \longrightarrow \sigma}$$

$$\frac{e\ \sigma = v}{\langle \mathsf{x := e}, \sigma\rangle \longrightarrow \sigma[x \mapsto v]}$$

$$\frac{}{\langle c_1; c_2, \sigma\rangle \longrightarrow \sigma''}$$

Structural Operational Semantics

$$\frac{}{\langle \mathsf{SKIP}, \sigma\rangle \longrightarrow \sigma}$$

$$\frac{e\ \sigma = v}{\langle \mathsf{x := e}, \sigma\rangle \longrightarrow \sigma[x \mapsto v]}$$

$$\frac{\langle c_1, \sigma\rangle \longrightarrow \sigma' \quad \langle c_2, \sigma'\rangle \longrightarrow \sigma''}{\langle c_1; c_2, \sigma\rangle \longrightarrow \sigma''}$$

$$\frac{b\ \sigma = \mathsf{True}}{\langle \mathsf{IF}\ b\ \mathsf{THEN}\ c_1\ \mathsf{ELSE}\ c_2, \sigma\rangle \longrightarrow \sigma'}$$

Structural Operational Semantics

$$\frac{}{\langle \mathsf{SKIP}, \sigma\rangle \longrightarrow \sigma}$$

$$\frac{e\ \sigma = v}{\langle \mathsf{x := e}, \sigma\rangle \longrightarrow \sigma[x \mapsto v]}$$

$$\frac{\langle c_1, \sigma\rangle \longrightarrow \sigma' \quad \langle c_2, \sigma'\rangle \longrightarrow \sigma''}{\langle c_1; c_2, \sigma\rangle \longrightarrow \sigma''}$$

$$\frac{b\ \sigma = \mathsf{True} \quad \langle c_1, \sigma\rangle \longrightarrow \sigma'}{\langle \mathsf{IF}\ b\ \mathsf{THEN}\ c_1\ \mathsf{ELSE}\ c_2, \sigma\rangle \longrightarrow \sigma'}$$

$$\frac{b\ \sigma = \mathsf{False}}{\langle \mathsf{IF}\ b\ \mathsf{THEN}\ c_1\ \mathsf{ELSE}\ c_2, \sigma\rangle \longrightarrow \sigma'}$$

Structural Operational Semantics

$$\frac{}{\langle \mathsf{SKIP}, \sigma\rangle \longrightarrow \sigma}$$

$$\frac{e\ \sigma = v}{\langle \mathsf{x := e}, \sigma\rangle \longrightarrow \sigma[x \mapsto v]}$$

$$\frac{\langle c_1, \sigma\rangle \longrightarrow \sigma' \quad \langle c_2, \sigma'\rangle \longrightarrow \sigma''}{\langle c_1; c_2, \sigma\rangle \longrightarrow \sigma''}$$

$$\frac{b\ \sigma = \mathsf{True} \quad \langle c_1, \sigma\rangle \longrightarrow \sigma'}{\langle \mathsf{IF}\ b\ \mathsf{THEN}\ c_1\ \mathsf{ELSE}\ c_2, \sigma\rangle \longrightarrow \sigma'}$$

$$\frac{b\ \sigma = \mathsf{False} \quad \langle c_2, \sigma\rangle \longrightarrow \sigma'}{\langle \mathsf{IF}\ b\ \mathsf{THEN}\ c_1\ \mathsf{ELSE}\ c_2, \sigma\rangle \longrightarrow \sigma'}$$

$$\frac{}{\langle \mathsf{WHILE}\ b\ \mathsf{DO}\ c\ \mathsf{OD}, \sigma\rangle \longrightarrow}$$

Structural Operational Semantics

$$\frac{b\ \sigma = \mathsf{False}}{\langle \mathsf{WHILE}\ b\ \mathsf{DO}\ c\ \mathsf{OD}, \sigma\rangle \longrightarrow \sigma}$$

Structural Operational Semantics

$$\frac{b\ \sigma = \mathsf{False}}{\langle \mathsf{WHILE}\ b\ \mathsf{DO}\ c\ \mathsf{OD}, \sigma\rangle \longrightarrow \sigma}$$

$$\frac{b\ \sigma = \mathsf{True}}{\langle \mathsf{WHILE}\ b\ \mathsf{DO}\ c\ \mathsf{OD}, \sigma\rangle \longrightarrow}$$

Structural Operational Semantics

$$\frac{b\ \sigma = \mathsf{False}}{\langle \mathsf{WHILE}\ b\ \mathsf{DO}\ c\ \mathsf{OD}, \sigma\rangle \longrightarrow \sigma}$$

$$\frac{b\ \sigma = \mathsf{True} \quad \langle c, \sigma\rangle \longrightarrow \sigma'}{\langle \mathsf{WHILE}\ b\ \mathsf{DO}\ c\ \mathsf{OD}, \sigma\rangle \longrightarrow}$$

Structural Operational Semantics

$$\frac{b\ \sigma = \mathsf{False}}{\langle \mathsf{WHILE}\ b\ \mathsf{DO}\ c\ \mathsf{OD}, \sigma\rangle \longrightarrow \sigma}$$

$$\frac{b\ \sigma = \mathsf{True} \quad \langle c, \sigma\rangle \longrightarrow \sigma' \quad \langle \mathsf{WHILE}\ b\ \mathsf{DO}\ c\ \mathsf{OD}, \sigma'\rangle \longrightarrow \sigma''}{\langle \mathsf{WHILE}\ b\ \mathsf{DO}\ c\ \mathsf{OD}, \sigma\rangle \longrightarrow \sigma''}$$

Demo: The Definitions in Isabelle

PROOFS ABOUT PROGRAMS

Now we know:

- ➜ What programs are: Syntax
- ➜ On what they work: State
- ➜ How they work: Semantics

PROOFS ABOUT PROGRAMS

Now we know:

- ➜ What programs are: Syntax
- ➜ On what they work: State
- ➜ How they work: Semantics

So we can prove properties about programs

Now we know:

- ➜ What programs are: Syntax
- ➜ On what they work: State
- ➜ How they work: Semantics

So we can prove properties about programs

Example:

Show that example program from slide 6 implements the factorial.

$$\textbf{lemma}\ \langle \text{factorial}, \sigma\rangle \longrightarrow \sigma' \Longrightarrow \sigma' B = \text{fac}\ (\sigma A)$$

(where $\text{fac}\ 0 = 0$, $\text{fac}\ (\text{Suc}\ n) = (\text{Suc}\ n) * \text{fac}\ n$)

DEMO: EXAMPLE PROOF

Too tedious

Induction needed for each loop

Too tedious

Induction needed for each loop

Is there something easier?

FLOYD/HOARE

Idea: describe meaning of program by pre/post conditions

Examples:

FLOYD/HOARE

Idea: describe meaning of program by pre/post conditions

Examples:

$$\{\mathsf{True}\} \quad x := 2 \quad \{x = 2\}$$

Floyd/Hoare

Idea: describe meaning of program by pre/post conditions

Examples:

$\{\mathsf{True}\} \quad x := 2 \quad \{x = 2\}$

$\{y = 2\} \quad x := 21 * y \quad \{x = 42\}$

Floyd/Hoare

Idea: describe meaning of program by pre/post conditions

Examples:

$\{\mathsf{True}\} \quad x := 2 \quad \{x = 2\}$

$\{y = 2\} \quad x := 21 * y \quad \{x = 42\}$

$\{x = n\} \quad \mathsf{IF}\ y < 0\ \mathsf{THEN}\ x := x + y\ \mathsf{ELSE}\ x := x - y \quad \{x = n - |y|\}$

Floyd/Hoare

Idea: describe meaning of program by pre/post conditions

Examples:

$\{\mathsf{True}\} \quad x := 2 \quad \{x = 2\}$

$\{y = 2\} \quad x := 21 * y \quad \{x = 42\}$

$\{x = n\} \quad \mathsf{IF}\ y < 0\ \mathsf{THEN}\ x := x + y\ \mathsf{ELSE}\ x := x - y \quad \{x = n - |y|\}$

$\{A = n\} \quad \mathsf{factorial} \quad \{B = \mathsf{fac}\ n\}$

Idea: describe meaning of program by pre/post conditions

Examples:

$\{\mathsf{True}\} \quad x := 2 \quad \{x = 2\}$

$\{y = 2\} \quad x := 21 * y \quad \{x = 42\}$

$\{x = n\} \quad \mathsf{IF}\ y < 0\ \mathsf{THEN}\ x := x + y\ \mathsf{ELSE}\ x := x - y \quad \{x = n - |y|\}$

$\{A = n\} \quad \mathsf{factorial} \quad \{B = \mathsf{fac}\ n\}$

Proofs: have rules that directly work on such triples

MEANING OF A HOARE-TRIPLE

$$\{P\} \quad c \quad \{Q\}$$

What are the assertions P and Q?

MEANING OF A HOARE-TRIPLE

$$\{P\} \quad c \quad \{Q\}$$

What are the assertions P and Q?

➜ Here: again functions from state to bool
(shallow embedding of assertions)

MEANING OF A HOARE-TRIPLE

$$\{P\} \quad c \quad \{Q\}$$

What are the assertions P and Q?

➜ Here: again functions from state to bool
(shallow embedding of assertions)

➜ Other choice: syntax and semantics for assertions (deep embedding)

What does $\{P\}\ c\ \{Q\}$ mean?

MEANING OF A HOARE-TRIPLE

$$\{P\} \quad c \quad \{Q\}$$

What are the assertions P and Q?

➜ Here: again functions from state to bool
(shallow embedding of assertions)

➜ Other choice: syntax and semantics for assertions (deep embedding)

What does $\{P\}\ c\ \{Q\}$ mean?

Partial Correctness:

$$\models \{P\}\ c\ \{Q\} \quad \equiv \quad (\forall \sigma\ \sigma'.\ P\ \sigma \wedge \langle c, \sigma\rangle \longrightarrow \sigma' \Longrightarrow Q\ \sigma')$$

MEANING OF A HOARE-TRIPLE

$$\{P\} \quad c \quad \{Q\}$$

What are the assertions P and Q?

- ➜ Here: again functions from state to bool
  (shallow embedding of assertions)
- ➜ Other choice: syntax and semantics for assertions (deep embedding)

What does $\{P\}\ c\ \{Q\}$ mean?

Partial Correctness:

$$\models \{P\}\ c\ \{Q\} \quad \equiv \quad (\forall \sigma\ \sigma'.\ P\ \sigma \wedge \langle c, \sigma\rangle \longrightarrow \sigma' \Longrightarrow Q\ \sigma')$$

Total Correctness:

$$\models \{P\}\ c\ \{Q\} \quad \equiv \quad (\forall \sigma.\ P\ \sigma \Longrightarrow \exists \sigma'.\ \langle c, \sigma\rangle \longrightarrow \sigma' \wedge Q\ \sigma')$$

MEANING OF A HOARE-TRIPLE

$$\{P\} \quad c \quad \{Q\}$$

What are the assertions P and Q?

➜ Here: again functions from state to bool
(shallow embedding of assertions)

➜ Other choice: syntax and semantics for assertions (deep embedding)

What does $\{P\}\ c\ \{Q\}$ mean?

Partial Correctness:

$$\models \{P\}\ c\ \{Q\} \quad \equiv \quad (\forall \sigma\ \sigma'.\ P\ \sigma \wedge \langle c, \sigma \rangle \longrightarrow \sigma' \Longrightarrow Q\ \sigma')$$

Total Correctness:

$$\models \{P\}\ c\ \{Q\} \quad \equiv \quad (\forall \sigma.\ P\ \sigma \Longrightarrow \exists \sigma'.\ \langle c, \sigma \rangle \longrightarrow \sigma' \wedge Q\ \sigma')$$

This lecture: partial correctness only (easier)

Hoare Rules

$$\frac{}{\{P\} \quad \textsf{SKIP} \quad \{P\}}$$

HOARE RULES

$$\frac{}{\{P\} \quad \textsf{SKIP} \quad \{P\}} \qquad \frac{}{\{P[x \mapsto e]\} \quad x := e \quad \{P\}}$$

Hoare Rules

$$\frac{}{\{P\} \quad \mathsf{SKIP} \quad \{P\}} \qquad \frac{}{\{P[x \mapsto e]\} \quad x := e \quad \{P\}}$$

$$\frac{\{P\}\ c_1\ \{R\} \quad \{R\}\ c_2\ \{Q\}}{\{P\} \quad c_1; c_2 \quad \{Q\}}$$

Hoare Rules

$$\frac{}{\{P\} \quad \textsf{SKIP} \quad \{P\}} \qquad \frac{}{\{P[x \mapsto e]\} \quad x := e \quad \{P\}}$$

$$\frac{\{P\}\ c_1\ \{R\} \quad \{R\}\ c_2\ \{Q\}}{\{P\} \quad c_1; c_2 \quad \{Q\}}$$

$$\frac{}{\{P\} \quad \textsf{IF}\ b\ \textsf{THEN}\ c_1\ \textsf{ELSE}\ c_2 \quad \{Q\}}$$

Hoare Rules

$$\frac{}{\{P\} \quad \textsf{SKIP} \quad \{P\}} \qquad \frac{}{\{P[x \mapsto e]\} \quad x := e \quad \{P\}}$$

$$\frac{\{P\}\; c_1\; \{R\} \quad \{R\}\; c_2\; \{Q\}}{\{P\} \quad c_1; c_2 \quad \{Q\}}$$

$$\frac{\{P \wedge b\}\; c_1\; \{Q\}}{\{P\} \quad \textsf{IF}\; b\; \textsf{THEN}\; c_1\; \textsf{ELSE}\; c_2 \quad \{Q\}}$$

Hoare Rules

$$\frac{}{\{P\} \quad \mathsf{SKIP} \quad \{P\}} \qquad \frac{}{\{P[x \mapsto e]\} \quad x := e \quad \{P\}}$$

$$\frac{\{P\}\ c_1\ \{R\} \quad \{R\}\ c_2\ \{Q\}}{\{P\} \quad c_1; c_2 \quad \{Q\}}$$

$$\frac{\{P \wedge b\}\ c_1\ \{Q\} \quad \{P \wedge \neg b\}\ c_2\ \{Q\}}{\{P\} \quad \mathsf{IF}\ b\ \mathsf{THEN}\ c_1\ \mathsf{ELSE}\ c_2 \quad \{Q\}}$$

Hoare Rules

$$\frac{}{\{P\} \quad \textsf{SKIP} \quad \{P\}} \qquad \frac{}{\{P[x \mapsto e]\} \quad x := e \quad \{P\}}$$

$$\frac{\{P\}\ c_1\ \{R\} \quad \{R\}\ c_2\ \{Q\}}{\{P\} \quad c_1; c_2 \quad \{Q\}}$$

$$\frac{\{P \wedge b\}\ c_1\ \{Q\} \quad \{P \wedge \neg b\}\ c_2\ \{Q\}}{\{P\} \quad \textsf{IF}\ b\ \textsf{THEN}\ c_1\ \textsf{ELSE}\ c_2 \quad \{Q\}}$$

$$\frac{\{P \wedge b\}\ c\ \{P\} \quad P \wedge \neg b \Longrightarrow Q}{\{P\} \quad \textsf{WHILE}\ b\ \textsf{DO}\ c\ \textsf{OD} \quad \{Q\}}$$

Hoare Rules

$$\frac{}{\{P\} \quad \textsf{SKIP} \quad \{P\}} \qquad \frac{}{\{P[x \mapsto e]\} \quad x := e \quad \{P\}}$$

$$\frac{\{P\}\ c_1\ \{R\} \quad \{R\}\ c_2\ \{Q\}}{\{P\} \quad c_1; c_2 \quad \{Q\}}$$

$$\frac{\{P \wedge b\}\ c_1\ \{Q\} \quad \{P \wedge \neg b\}\ c_2\ \{Q\}}{\{P\} \quad \textsf{IF}\ b\ \textsf{THEN}\ c_1\ \textsf{ELSE}\ c_2 \quad \{Q\}}$$

$$\frac{\{P \wedge b\}\ c\ \{P\} \quad P \wedge \neg b \Longrightarrow Q}{\{P\} \quad \textsf{WHILE}\ b\ \textsf{DO}\ c\ \textsf{OD} \quad \{Q\}}$$

$$\frac{\{P'\}\ c\ \{Q'\}}{\{P\} \quad c \quad \{Q\}}$$

HOARE RULES

$$\frac{}{\{P\} \quad \textsf{SKIP} \quad \{P\}} \qquad \frac{}{\{P[x \mapsto e]\} \quad x := e \quad \{P\}}$$

$$\frac{\{P\}\ c_1\ \{R\} \quad \{R\}\ c_2\ \{Q\}}{\{P\} \quad c_1; c_2 \quad \{Q\}}$$

$$\frac{\{P \wedge b\}\ c_1\ \{Q\} \quad \{P \wedge \neg b\}\ c_2\ \{Q\}}{\{P\} \quad \textsf{IF}\ b\ \textsf{THEN}\ c_1\ \textsf{ELSE}\ c_2 \quad \{Q\}}$$

$$\frac{\{P \wedge b\}\ c\ \{P\} \quad P \wedge \neg b \Longrightarrow Q}{\{P\} \quad \textsf{WHILE}\ b\ \textsf{DO}\ c\ \textsf{OD} \quad \{Q\}}$$

$$\frac{P \Longrightarrow P' \quad \{P'\}\ c\ \{Q'\} \quad Q' \Longrightarrow Q}{\{P\} \quad c \quad \{Q\}}$$

Hoare Rules

$$\frac{}{\vdash \{P\} \quad \text{SKIP} \quad \{P\}} \qquad \frac{}{\vdash \{\lambda\sigma.\ P\ (\sigma(x := e\ \sigma))\} \quad x := e \quad \{P\}}$$

$$\frac{\vdash \{P\}\ c_1\ \{R\} \quad \vdash \{R\}\ c_2\ \{Q\}}{\vdash \{P\} \quad c_1; c_2 \quad \{Q\}}$$

$$\frac{\vdash \{\lambda\sigma.\ P\ \sigma \wedge b\ \sigma\}\ c_1\ \{R\} \quad \vdash \{\lambda\sigma.\ P\ \sigma \wedge \neg b\ \sigma\}\ c_2\ \{Q\}}{\vdash \{P\} \quad \text{IF}\ b\ \text{THEN}\ c_1\ \text{ELSE}\ c_2 \quad \{Q\}}$$

$$\frac{\vdash \{\lambda\sigma.\ P\ \sigma \wedge b\ \sigma\}\ c\ \{P\} \quad \bigwedge\sigma.\ P\ \sigma \wedge \neg b\ \sigma \Longrightarrow Q\ \sigma}{\vdash \{P\} \quad \text{WHILE}\ b\ \text{DO}\ c\ \text{OD} \quad \{Q\}}$$

$$\frac{\bigwedge\sigma.\ P\ \sigma \Longrightarrow P'\ \sigma \quad \vdash \{P'\}\ c\ \{Q'\} \quad \bigwedge\sigma.\ Q'\ \sigma \Longrightarrow Q\sigma}{\vdash \{P\} \quad c \quad \{Q\}}$$

ARE THE RULES CORRECT?

Soundness: $\vdash \{P\}\ c\ \{Q\} \Longrightarrow \models \{P\}\ c\ \{Q\}$

Are the Rules Correct?

Soundness: $\vdash \{P\}\ c\ \{Q\} \Longrightarrow \models \{P\}\ c\ \{Q\}$

Proof: by rule induction on $\vdash \{P\}\ c\ \{Q\}$

ARE THE RULES CORRECT?

Soundness: $\vdash \{P\}\ c\ \{Q\} \Longrightarrow \models \{P\}\ c\ \{Q\}$

Proof: by rule induction on $\vdash \{P\}\ c\ \{Q\}$

Demo: Hoare Logic in Isabelle

Nicer, but still kind of tedious

Hoare rule application seems boring & mechanical.

Automation?

NICER, BUT STILL KIND OF TEDIOUS

Hoare rule application seems boring & mechanical.

Automation?

Problem: While – need creativity to fi nd right (invariant) P

NICER, BUT STILL KIND OF TEDIOUS

Hoare rule application seems boring & mechanical.

Automation?

Problem: While – need creativity to fi nd right (invariant) P

Solution:

➜ annotate program with invariants

Nicer, but still kind of tedious

Hoare rule application seems boring & mechanical.

Automation?

Problem: While – need creativity to fi nd right (invariant) P

Solution:

➜ annotate program with invariants

➜ then, Hoare rules can be applied automatically

NICER, BUT STILL KIND OF TEDIOUS

Hoare rule application seems boring & mechanical.

Automation?

Problem: While – need creativity to fi nd right (invariant) P

Solution:

- ➜ annotate program with invariants
- ➜ then, Hoare rules can be applied automatically

Example:

$\{M = 0 \wedge N = 0\}$

WHILE $M \neq a$ INV $\{N = M * b\}$ DO $N := N + b; M := M + 1$ OD

$\{N = a * b\}$

WEAKEST PRECONDITIONS

pre $c\ Q$ **= weakest** P **such that** $\{P\}\ c\ \{Q\}$

With annotated invariants, easy to get:

WEAKEST PRECONDITIONS

pre $c\ Q$ **= weakest** P **such that** $\{P\}\ c\ \{Q\}$

With annotated invariants, easy to get:

pre SKIP Q $\quad = \quad Q$

WEAKEST PRECONDITIONS

pre $c\ Q$ **= weakest** P **such that** $\{P\}\ c\ \{Q\}$

With annotated invariants, easy to get:

pre SKIP $Q \quad = \quad Q$

pre $(x := a)\ Q \quad = \quad \lambda\sigma.\ Q(\sigma(x := a\sigma))$

WEAKEST PRECONDITIONS

pre $c\ Q$ **= weakest** P **such that** $\{P\}\ c\ \{Q\}$

With annotated invariants, easy to get:

$$\begin{array}{lcl}
\text{pre SKIP } Q & = & Q \\
\text{pre } (x := a)\ Q & = & \lambda\sigma.\ Q(\sigma(x := a\sigma)) \\
\text{pre } (c_1; c_2)\ Q & = & \text{pre } c_1 \text{ (pre } c_2\ Q)
\end{array}$$

Weakest Preconditions

pre $c\ Q$ **= weakest** P **such that** $\{P\}\ c\ \{Q\}$

With annotated invariants, easy to get:

$$
\begin{array}{lcl}
\text{pre SKIP } Q & = & Q \\
\text{pre } (x := a)\ Q & = & \lambda\sigma.\ Q(\sigma(x := a\sigma)) \\
\text{pre } (c_1; c_2)\ Q & = & \text{pre } c_1 \text{ (pre } c_2\ Q\text{)} \\
\text{pre } (\text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2)\ Q & = & \lambda\sigma.\ (b \longrightarrow \text{pre } c_1\ Q\ \sigma) \wedge \\
 & & \quad (\neg b \longrightarrow \text{pre } c_2\ Q\ \sigma)
\end{array}
$$

Weakest Preconditions

pre $c\ Q$ **= weakest** P **such that** $\{P\}\ c\ \{Q\}$

With annotated invariants, easy to get:

pre SKIP Q	$=$	Q
pre $(x := a)\ Q$	$=$	$\lambda\sigma.\ Q(\sigma(x := a\sigma))$
pre $(c_1; c_2)\ Q$	$=$	pre c_1 (pre c_2 Q)
pre (IF b THEN c_1 ELSE c_2) Q	$=$	$\lambda\sigma.\ (b \longrightarrow$ pre $c_1\ Q\ \sigma) \wedge$ $(\neg b \longrightarrow$ pre $c_2\ Q\ \sigma)$
pre (WHILE b INV I DO c OD) Q	$=$	I

VERIFICATION CONDITIONS

$\{\mathsf{pre}\ c\ Q\}\ c\ \{Q\}$ **only true under certain conditions**

VERIFICATION CONDITIONS

$\{\text{pre } c\ Q\}\ c\ \{Q\}$ **only true under certain conditions**

These are called **verification conditions** vc $c\ Q$:

vc SKIP Q $=$ True

VERIFICATION CONDITIONS

$\{\text{pre } c\ Q\}\ c\ \{Q\}$ **only true under certain conditions**

These are called **verification conditions** vc $c\ Q$:

vc SKIP Q	$=$	True
vc $(x := a)\ Q$	$=$	True

VERIFICATION CONDITIONS

$\{\text{pre } c\ Q\}\ c\ \{Q\}$ **only true under certain conditions**

These are called **verification conditions** vc $c\ Q$:

vc SKIP Q	$=$	True
vc $(x := a)\ Q$	$=$	True
vc $(c_1; c_2)\ Q$	$=$	vc $c_2\ Q \wedge$ (vc c_1 (pre $c_2\ Q$))

VERIFICATION CONDITIONS

$\{\text{pre } c\ Q\}\ c\ \{Q\}$ **only true under certain conditions**

These are called **verification conditions** vc $c\ Q$:

vc SKIP Q	$=$	True
vc $(x := a)\ Q$	$=$	True
vc $(c_1; c_2)\ Q$	$=$	vc $c_2\ Q \wedge$ (vc c_1 (pre $c_2\ Q$))
vc (IF b THEN c_1 ELSE c_2) Q	$=$	vc $c_1\ Q \wedge$ vc $c_2\ Q$

VERIFICATION CONDITIONS

$\{\textsf{pre}\ c\ Q\}\ c\ \{Q\}$ **only true under certain conditions**

These are called **verification conditions** vc c Q:

$$
\begin{array}{lcl}
\textsf{vc SKIP}\ Q & = & \textsf{True} \\
\textsf{vc}\ (x := a)\ Q & = & \textsf{True} \\
\textsf{vc}\ (c_1; c_2)\ Q & = & \textsf{vc}\ c_2\ Q \wedge (\textsf{vc}\ c_1\ (\textsf{pre}\ c_2\ Q)) \\
\textsf{vc}\ (\textsf{IF}\ b\ \textsf{THEN}\ c_1\ \textsf{ELSE}\ c_2)\ Q & = & \textsf{vc}\ c_1\ Q \wedge \textsf{vc}\ c_2\ Q \\
\textsf{vc}\ (\textsf{WHILE}\ b\ \textsf{INV}\ I\ \textsf{DO}\ c\ \textsf{OD})\ Q & = & (\forall \sigma.\ I\sigma \wedge b\sigma \longrightarrow \textsf{pre}\ c\ I\ \sigma) \wedge \\
 & & (\forall \sigma.\ I\sigma \wedge \neg b\sigma \longrightarrow Q\ \sigma) \wedge \\
 & & \textsf{vc}\ c\ I
\end{array}
$$

VERIFICATION CONDITIONS

$\{\text{pre } c\ Q\}\ c\ \{Q\}$ **only true under certain conditions**

These are called **verification conditions** vc $c\ Q$:

$$
\begin{array}{lcl}
\text{vc SKIP } Q & = & \text{True} \\
\text{vc } (x := a)\ Q & = & \text{True} \\
\text{vc } (c_1; c_2)\ Q & = & \text{vc } c_2\ Q \wedge (\text{vc } c_1\ (\text{pre } c_2\ Q)) \\
\text{vc } (\text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2)\ Q & = & \text{vc } c_1\ Q \wedge \text{vc } c_2\ Q \\
\text{vc } (\text{WHILE } b \text{ INV } I \text{ DO } c \text{ OD})\ Q & = & (\forall \sigma.\ I\sigma \wedge b\sigma \longrightarrow \text{pre } c\ I\ \sigma) \wedge \\
 & & (\forall \sigma.\ I\sigma \wedge \neg b\sigma \longrightarrow Q\ \sigma) \wedge \\
 & & \text{vc } c\ I
\end{array}
$$

$$\text{vc } c\ Q \wedge (\text{pre } c\ Q \Longrightarrow P) \Longrightarrow \{P\}\ c\ \{Q\}$$

SYNTAX TRICKS

➜ $x := \lambda\sigma.\ 1$ instead of $x := 1$ sucks

➜ $\{\lambda\sigma.\ \sigma\ x = n\}$ instead of $\{x = n\}$ sucks as well

Syntax Tricks

- ➜ $x := \lambda\sigma.\ 1$ instead of $x := 1$ sucks
- ➜ $\{\lambda\sigma.\ \sigma\ x = n\}$ instead of $\{x = n\}$ sucks as well

Problem: program variables are functions, not values

SYNTAX TRICKS

- ➜ $x := \lambda\sigma.\ 1$ instead of $x := 1$ sucks
- ➜ $\{\lambda\sigma.\ \sigma\ x = n\}$ instead of $\{x = n\}$ sucks as well

Problem: program variables are functions, not values

Solution: distinguish program variables syntactically

SYNTAX TRICKS

- ➜ $x := \lambda\sigma.\ 1$ instead of $x := 1$ sucks
- ➜ $\{\lambda\sigma.\ \sigma\ x = n\}$ instead of $\{x = n\}$ sucks as well

Problem: program variables are functions, not values

Solution: distinguish program variables syntactically

Choices:

- ➜ declare program variables with each Hoare triple

SYNTAX TRICKS

- ➜ $x := \lambda\sigma.\ 1$ instead of $x := 1$ sucks
- ➜ $\{\lambda\sigma.\ \sigma\ x = n\}$ instead of $\{x = n\}$ sucks as well

Problem: program variables are functions, not values

Solution: distinguish program variables syntactically

Choices:

- ➜ declare program variables with each Hoare triple
  - nice, usual syntax
  - works well if you state full program and only use vcg

SYNTAX TRICKS

- ➜ $x := \lambda\sigma.\ 1$ instead of $x := 1$ sucks
- ➜ $\{\lambda\sigma.\ \sigma\ x = n\}$ instead of $\{x = n\}$ sucks as well

Problem: program variables are functions, not values

Solution: distinguish program variables syntactically

Choices:

- ➜ declare program variables with each Hoare triple
  - nice, usual syntax
  - works well if you state full program and only use vcg
- ➜ separate program variables from Hoare triple (use extensible records), indicate usage as function syntactically

SYNTAX TRICKS

➜ $x := \lambda\sigma.\ 1$ instead of $x := 1$ sucks

➜ $\{\lambda\sigma.\ \sigma\ x = n\}$ instead of $\{x = n\}$ sucks as well

Problem: program variables are functions, not values

Solution: distinguish program variables syntactically

Choices:

➜ declare program variables with each Hoare triple

- nice, usual syntax
- works well if you state full program and only use vcg

➜ separate program variables from Hoare triple (use extensible records), indicate usage as function syntactically

- more syntactic overhead
- program pieces compose nicely

Records in Isabelle

Records are a tuples with named components

RECORDS IN ISABELLE

Records are a tuples with named components

Example:

record A = a :: nat
b :: int

RECORDS IN ISABELLE

Records are a tuples with named components

Example:

record A = a :: nat
b :: int

➜ Selectors: a :: A $\Rightarrow$ nat, b :: A $\Rightarrow$ int, a $r = \mathsf{Suc}\ 0$

Records are a tuples with named components

Example:

record A = a :: nat
b :: int

- ➜ Selectors: a :: A $\Rightarrow$ nat, b :: A $\Rightarrow$ int, a $r = \mathsf{Suc}\ 0$
- ➜ Constructors: $(\!|\ \mathsf{a} = \mathsf{Suc}\ 0,\ \mathsf{b} = -1\ |\!)$

Records in Isabelle

Records are a tuples with named components

Example:

record A = a :: nat
b :: int

➜ Selectors: a :: A $\Rightarrow$ nat, b :: A $\Rightarrow$ int, a $r = \mathsf{Suc}\ 0$

➜ Constructors: $(\!|\ \mathsf{a} = \mathsf{Suc}\ 0,\ \mathsf{b} = -1\ |\!)$

➜ Update: $r(\!|\ \mathsf{a} := \mathsf{Suc}\ 0\ |\!)$

Records in Isabelle

Records are a tuples with named components

Example:

record A = a :: nat
b :: int

- Selectors: a :: A $\Rightarrow$ nat, b :: A $\Rightarrow$ int, a $r = \mathsf{Suc}\ 0$
- Constructors: $(\!|\ \mathsf{a} = \mathsf{Suc}\ 0,\ \mathsf{b} = -1\ |\!)$
- Update: $r(\!|\ \mathsf{a} := \mathsf{Suc}\ 0\ |\!)$

Records are extensible:

record B = A +
c :: nat list

RECORDS IN ISABELLE

Records are a tuples with named components

Example:

record A = a :: nat
b :: int

- Selectors: a :: A $\Rightarrow$ nat, b :: A $\Rightarrow$ int, a $r = \mathsf{Suc}\ 0$
- Constructors: $(\!|\ \mathsf{a} = \mathsf{Suc}\ 0,\ \mathsf{b} = -1\ |\!)$
- Update: $r(\!|\ \mathsf{a} := \mathsf{Suc}\ 0\ |\!)$

Records are extensible:

record B = A +
c :: nat list

$$(\!|\ \mathsf{a} = \mathsf{Suc}\ 0,\ \mathsf{b} = -1,\ \mathsf{c} = [0, 0]\ |\!)$$

DEMO

Available now in Isablle:

- ➜ procedures

MORE

Available now in Isablle:

- ➜ procedures
- ➜ with blocks and local variables

MORE

Available now in Isablle:

- ➜ procedures
- ➜ with blocks and local variables
- ➜ and (mutual) recursion

MORE

Available now in Isablle:

- ➜ procedures
- ➜ with blocks and local variables
- ➜ and (mutual) recursion
- ➜ exceptions

MORE

Available now in Isablle:

- ➜ procedures
- ➜ with blocks and local variables
- ➜ and (mutual) recursion
- ➜ exceptions
- ➜ arrays

MORE

Available now in Isablle:

- ➜ procedures
- ➜ with blocks and local variables
- ➜ and (mutual) recursion
- ➜ exceptions
- ➜ arrays
- ➜ pointers

MORE

Available now in Isablle:

- ➜ procedures
- ➜ with blocks and local variables
- ➜ and (mutual) recursion
- ➜ exceptions
- ➜ arrays
- ➜ pointers

We're working at:

- ➜ nondeterminsm

More

Available now in Isablle:

- ➜ procedures
- ➜ with blocks and local variables
- ➜ and (mutual) recursion
- ➜ exceptions
- ➜ arrays
- ➜ pointers

We're working at:

- ➜ nondeterminsm
- ➜ probability

MORE

Available now in Isablle:

- ➜ procedures
- ➜ with blocks and local variables
- ➜ and (mutual) recursion
- ➜ exceptions
- ➜ arrays
- ➜ pointers

We're working at:

- ➜ nondeterminsm
- ➜ probability
- ➜ object orientation

We have seen today ...

- ➜ Syntax and semantics of IMP
- ➜ Hoare logic rules
- ➜ Soundness of Hoare logic
- ➜ Verification conditions
- ➜ Example program proofs

EXERCISES

- ➜ Write a program in IMP that calculates quotient and reminder of $x \in \mathbb{N}$ and $y \in \mathbb{N}$
- ➜ Find the right invariant for its while loop.
- ➜ Show its correctness in Isabelle:
  $\vdash \{\mathsf{True}\}\ \ \mathsf{program}\ \ \{\,\acute{}Q * y + \acute{}R = x \wedge \acute{}R < y\,\}$

- ➜ Write an IMP program that sorts arrays (lists) by insertion sort.
- ➜ Formulate and show its correctness in Isabelle.