COMP4161: Advanced Topics in Software Verification

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

Gerwin Klein, Johannes Åman Pohjola, Christine Rizkallah, Miki Tanaka
T3/2020
Content

→ Foundations & Principles
  • Intro, Lambda calculus, natural deduction [1,2]
  • Higher Order Logic, Isar (part 1) [2,3\textsuperscript{a}]
  • Term rewriting [3,4]

→ Proof & Specification Techniques
  • Inductively defined sets, rule induction, datatype induction, primitive recursion [4,5]
  • General recursive functions, termination proofs [7\textsuperscript{b}]
  • Proof automation, Hoare logic, proofs about programs, invariants [8]
  • C verification [9,10]
  • Practice, questions, examp prep [10\textsuperscript{c}]

\textsuperscript{a} a1 due; \textsuperscript{b} a2 due; \textsuperscript{c} a3 due
A Crash Course in Semantics
(For more, see Concrete Semantics)
IMP - a small Imperative Language

Commands:
\[
\text{datatype } \text{com} = \text{SKIP} \\
\text{Assign } v\text{name } a\text{exp } (\_:=\_ ) \\
\text{Semi com com } (\_;\_ ) \\
\text{Cond bexp com com } (\text{IF } \_\text{ THEN } \_\text{ ELSE } \_ ) \\
\text{While bexp com } (\text{WHILE } \_\text{ DO } \_\text{ OD} )
\]
**IMP - a small Imperative Language**

### Commands:

**datatype** com

- SKIP
- Assign vname aexp (vname := aexp)
- Semi com com (com ; com)
- Cond bexp com com (IF bexp THEN com ELSE com)
- While bexp com (WHILE bexp DO com OD)

**type_synonym** vname = string
**type_synonym** state = vname ⇒ nat
IMP - a small Imperative Language

Commands:
\[
\text{datatype com} = \text{SKIP} \\
\text{Assign vname aexp (}_:=_\text{)} \\
\text{Semi com com (}_{;}_\text{)} \\
\text{Cond bexp com com (IF \_ THEN \_ ELSE \_)} \\
\text{While bexp com (WHILE \_ DO \_ OD)}
\]

\[
\text{type synonym vname} = \text{string} \\
\text{type synonym state} = \text{vname} \Rightarrow \text{nat} \\
\text{type synonym aexp} = \text{state} \Rightarrow \text{nat} \\
\text{type synonym bexp} = \text{state} \Rightarrow \text{bool}
\]
Example Program

Usual syntax:

\[
\begin{align*}
B & := 1; \\
\text{WHILE } A \neq 0 \text{ DO} \\
B & := B \times A; \\
A & := A - 1 \\
\text{OD}
\end{align*}
\]
Example Program

Usual syntax:

\[
\begin{align*}
B & := 1; \\
\text{WHILE } A \neq 0 \text{ DO} \\
B & := B \times A; \\
A & := A - 1 \\
\text{OD}
\end{align*}
\]

Expressions are functions from state to bool or nat:

\[
\begin{align*}
B & := (\lambda \sigma. 1); \\
\text{WHILE } (\lambda \sigma. \sigma A \neq 0) \text{ DO} \\
B & := (\lambda \sigma. \sigma B \times \sigma A); \\
A & := (\lambda \sigma. \sigma A - 1) \\
\text{OD}
\end{align*}
\]
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
- Some choices:
  - Operational (inductive relations, big step, small step)
  - Denotational (programs as functions on states, state transformers)
  - Axiomatic (pre-/post conditions, Hoare logic)
What does it do?

So far we have defined:

- **Syntax** of commands and expressions

Now we need:

the meaning (semantics) of programs

How to define execution of a program?

- A wide field of its own
- Some choices:
  - Operational (inductive relations, big step, small step)
  - Denotational (programs as functions on states, state transformers)
  - Axiomatic (pre-/post conditions, Hoare logic)
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
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
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
→ 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

\[ \langle \text{SKIP}, \sigma \rangle \rightarrow \sigma \]
Structural Operational Semantics

\[ \langle \text{SKIP}, \sigma \rangle \rightarrow \sigma \]

\[ \langle x := e, \sigma \rangle \rightarrow \]

Structural Operational Semantics

\[
\begin{align*}
\langle \text{SKIP}, \sigma \rangle &\rightarrow \sigma \\
\langle x := e, \sigma \rangle &\rightarrow \sigma[x \mapsto v]
\end{align*}
\]
Structural Operational Semantics

\[ \langle \text{SKIP}, \sigma \rangle \rightarrow \sigma \]

\[ e \; \sigma = \nu \]

\[ \langle x := e, \sigma \rangle \rightarrow \sigma[x \mapsto \nu] \]

\[ \langle c_1; \; c_2, \sigma \rangle \rightarrow \sigma'' \]
Structural Operational Semantics

\[
\begin{align*}
\langle \text{SKIP}, \sigma \rangle & \rightarrow \sigma \\
\langle x := e, \sigma \rangle & \rightarrow \sigma[x \mapsto v] \\
\langle c_1, \sigma \rangle & \rightarrow \sigma' \quad \langle c_2, \sigma' \rangle & \rightarrow \sigma'' \\
\langle c_1; c_2, \sigma \rangle & \rightarrow \sigma''
\end{align*}
\]
Structural Operational Semantics

\[
\langle \text{SKIP}, \sigma \rangle \rightarrow \sigma
\]

\[
e \sigma = v \\
\langle x := e, \sigma \rangle \rightarrow \sigma[x \mapsto v]
\]

\[
\langle c_1, \sigma \rangle \rightarrow \sigma' \quad \langle c_2, \sigma' \rangle \rightarrow \sigma''
\]

\[
\langle c_1; c_2, \sigma \rangle \rightarrow \sigma''
\]

\[
b \sigma = \text{True} \\
\langle \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2, \sigma \rangle \rightarrow \sigma'
\]
Structural Operational Semantics

\[ \langle \text{SKIP}, \sigma \rangle \rightarrow \sigma \]

\[ e \sigma = v \quad \rightarrow \quad \langle x := e, \sigma \rangle \rightarrow \sigma[x \mapsto v] \]

\[ \langle c_1, \sigma \rangle \rightarrow \sigma' \quad \langle c_2, \sigma' \rangle \rightarrow \sigma'' \]

\[ \langle c_1; c_2, \sigma \rangle \rightarrow \sigma'' \]

\[ b \sigma = \text{True} \quad \rightarrow \quad \langle c_1, \sigma \rangle \rightarrow \sigma' \]

\[ \langle \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2, \sigma \rangle \rightarrow \sigma' \]
Structural Operational Semantics

\[
\langle \text{SKIP}, \sigma \rangle \rightarrow \sigma
\]

\[
e \sigma = v
\]

\[
\langle x := e, \sigma \rangle \rightarrow \sigma[x \mapsto v]
\]

\[
\langle c_1, \sigma \rangle \rightarrow \sigma' \quad \langle c_2, \sigma' \rangle \rightarrow \sigma''
\]

\[
\langle c_1; c_2, \sigma \rangle \rightarrow \sigma''
\]

\[
b \sigma = \text{True} \quad \langle c_1, \sigma \rangle \rightarrow \sigma'
\]

\[
\langle \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2, \sigma \rangle \rightarrow \sigma'
\]

\[
b \sigma = \text{False}
\]

\[
\langle \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2, \sigma \rangle \rightarrow \sigma'
\]
Structural Operational Semantics

\[
\begin{align*}
\langle \text{SKIP}, \sigma \rangle & \rightarrow \sigma \\
e \sigma = v & \Rightarrow \langle x := e, \sigma \rangle \rightarrow \sigma[x \mapsto v] \\
\langle c_1, \sigma \rangle & \rightarrow \sigma' \quad \langle c_2, \sigma' \rangle \rightarrow \sigma'' \\
\langle c_1; c_2, \sigma \rangle & \rightarrow \sigma'' \\
b \sigma = \text{True} & \Rightarrow \langle c_1, \sigma \rangle \rightarrow \sigma' \\
\langle \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2, \sigma \rangle & \rightarrow \sigma' \\
b \sigma = \text{False} & \Rightarrow \langle c_2, \sigma \rangle \rightarrow \sigma' \\
\langle \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2, \sigma \rangle & \rightarrow \sigma'
\end{align*}
\]
Structural Operational Semantics

\[ \langle \text{WHILE } b \text{ DO } c \text{ OD, } \sigma \rangle \rightarrow \]
Structural Operational Semantics

\[
\begin{align*}
  b \sigma &= \text{False} \\
  \langle \text{WHILE } b \text{ DO } c \text{ OD, } \sigma \rangle &\rightarrow \sigma
\end{align*}
\]
Structural Operational Semantics

\[
\frac{b \sigma = \text{False}}{
\langle \text{WHILE } b \text{ DO } c \text{ OD, } \sigma \rangle \rightarrow \sigma}
\]

\[
\frac{b \sigma = \text{True}}{
\langle \text{WHILE } b \text{ DO } c \text{ OD, } \sigma \rangle \rightarrow}
\]
Structural Operational Semantics

\[
\begin{align*}
&b \sigma = \text{False} \\
&\langle \text{WHILE } b \text{ DO } c \text{ OD}, \sigma \rangle \rightarrow \sigma
\end{align*}
\]

\[
\begin{align*}
&b \sigma = \text{True} \\
&\langle c, \sigma \rangle \rightarrow \sigma' \\
&\langle \text{WHILE } b \text{ DO } c \text{ OD}, \sigma \rangle \rightarrow
\end{align*}
\]
Structural Operational Semantics

\[ b \sigma = \text{False} \]
\[ \langle \text{WHILE } b \text{ DO } c \text{ OD}, \sigma \rangle \rightarrow \sigma \]

\[ b \sigma = \text{True} \]
\[ \langle c, \sigma \rangle \rightarrow \sigma' \]
\[ \langle \text{WHILE } b \text{ DO } c \text{ OD}, \sigma' \rangle \rightarrow \sigma'' \]
\[ \langle \text{WHILE } b \text{ DO } c \text{ OD}, \sigma \rangle \rightarrow \sigma'' \]
Demo: The Definitions in Isabelle
Proofs about Programs

Now we know:

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

Example:

Show that example program from before implements the factorial function.

\[ \text{lemma } \langle \text{factorial}, \sigma \rangle \rightarrow \sigma' = \Rightarrow \sigma' \text{B} = \text{fac} (\sigma \text{A}) \] (where \( \text{fac} 0 = 1 \), \( \text{fac} (\text{Suc} n) = (\text{Suc} n) \ast \text{fac} n \))
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
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

Example:
Show that example program from before implements the factorial function.

\[
\text{lemma } \langle \text{factorial}, \sigma \rangle \rightarrow \sigma' \implies \sigma' B = \text{fac } (\sigma A)
\]

(where \( \text{fac } 0 = 1, \text{ fac } (\text{Suc } n) = (\text{Suc } n) \ast \text{ fac } n \))
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:
{True}  $x := 2$  \{x = 2\}
Floyd/Hoare

**Idea:** describe meaning of program by pre/post conditions

**Examples:**

\[
\begin{align*}
\{ \text{True} \} & \quad x := 2 \quad \{ x = 2 \} \\
\{ y = 2 \} & \quad x := 21 \times y \quad \{ x = 42 \}
\end{align*}
\]
**Floyd/Hoare**

**Idea:** describe meaning of program by pre/post conditions

**Examples:**

\{True\} \ x := 2 \ \{x = 2\}  
\{y = 2\} \ x := 21 \ast y \ \{x = 42\}

\{x = n\} \ \text{IF} \ y < 0 \ \text{THEN} \ x := x + y \ \text{ELSE} \ x := x - y \ \{x = n - |y|\}
Floyd/Hoare

**Idea:** describe meaning of program by pre/post conditions

**Examples:**

\[
\begin{align*}
\{ \text{True} \} & \quad x := 2 \quad \{ x = 2 \} \\
\{ y = 2 \} & \quad x := 21 \times y \quad \{ x = 42 \} \\
\{ x = n \} & \quad \text{IF } y < 0 \text{ THEN } x := x + y \text{ ELSE } x := x - y \quad \{ x = n - |y| \} \\
\{ A = n \} & \quad \text{factorial} \quad \{ B = \text{fac } n \}
\end{align*}
\]
Floyd/Hoare

**Idea:** describe meaning of program by pre/post conditions

**Examples:**

\{True\} \ x := 2 \ {x = 2}\n\{y = 2\} \ x := 21 \times y \ {x = 42}\n
\{x = n\} \text{ IF } y < 0 \text{ THEN } x := x + y \text{ ELSE } x := x - y \ {x = n - |y|}\n
\{A = n\} \text{ factorial} \ {B = \text{fac } n}\n
**Proofs:** have rules that directly work on such triples
Meaning of a Hoare-Triple

\{P\} \ c \ \{Q\}

What are the assertions $P$ and $Q$?
Meaning of a Hoare-Triple

\[\{P\} \ c \ \{Q\}\]

What are the assertions \(P\) and \(Q\)?

⇒ Here: again functions from state to bool
   (shallow embedding of assertions)
Meaning of a Hoare-Triple

\{P\} \ c \ \{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\} \ c \ \{Q\}

What are the assertions \(P\) and \(Q\)?

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

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

What does \(\{P\} \ c \ \{Q\}\) mean?

Partial Correctness:
\[\models \{P\} \ c \ \{Q\}\ \equiv \ \forall \sigma, \sigma'. \ P \ \sigma \land \langle c, \sigma \rangle \rightarrow \sigma' \rightarrow Q \ \sigma'\]
Meaning of a Hoare-Triple

\{P\} \ c \ \{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:
\[ \models \{P\} \ c \ \{Q\} \equiv \forall \sigma \ \sigma'. \ P \ \sigma \land \langle c, \sigma \rangle \rightarrow \sigma' \rightarrow Q \ \sigma' \]

Total Correctness:
\[ \models \{P\} \ c \ \{Q\} \equiv (\forall \sigma \ \sigma'. \ P \ \sigma \land \langle c, \sigma \rangle \rightarrow \sigma' \rightarrow Q \ \sigma') \land \\
(\forall \sigma. \ P \ \sigma \rightarrow \exists \sigma'. \langle c, \sigma \rangle \rightarrow \sigma') \]
Meaning of a Hoare-Triple

\{P\} \ c \ \{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:
$\models \{P\} \ c \ \{Q\} \equiv \forall \sigma \sigma'. P \sigma \land \langle c, \sigma \rangle \rightarrow \sigma' \rightarrow Q \sigma'$

Total Correctness:
$\models \{P\} \ c \ \{Q\} \equiv (\forall \sigma \sigma'. P \sigma \land \langle c, \sigma \rangle \rightarrow \sigma' \rightarrow Q \sigma') \land (\forall \sigma. P \sigma \rightarrow \exists \sigma'. \langle c, \sigma \rangle \rightarrow \sigma')$

This lecture: partial correctness only (easier)
Hoare Rules

\[
\{P\} \text{SKIP} \{P\}
\]
Hoare Rules

\[
\begin{align*}
\{P\} & \quad \text{SKIP} & \{P\} \\
\{P[x \mapsto e]\} & \quad x := e & \{P\}
\end{align*}
\]
Hoare Rules

\[
\begin{align*}
\{P\} & \text{ SKIP } \{P\} \\
\{P[x \mapsto e]\} & x := e \ {P} \\
\{P\} & c_1 \ {R} \quad \{R\} & c_2 \ {Q} \\
\{P\} & c_1; c_2 \ {Q}
\end{align*}
\]
Hoare Rules

\[
\begin{align*}
\{P\} & \text{ SKIP } \{P\} \\
\{P[x \rightarrow e]\} & \ x := e \quad \{P\} \\
\{P\} & \ c_1 \quad \{R\} \\
\{R\} & \ c_2 \quad \{Q\} \\
\{P\} & \ c_1 \ ; \ c_2 \quad \{Q\} \\
\{P\} & \ \text{IF } b \ \text{ THEN } c_1 \ \text{ELSE } c_2 \quad \{Q\}
\end{align*}
\]
Hoare Rules

\[
\begin{align*}
\{ P \} \quad & \text{SKIP} \quad \{ P \} \\
\{ P \} \quad & x := e \quad \{ P \} \\
\{ P \} \quad & c_1 \quad \{ R \} \quad \{ R \} \quad c_2 \quad \{ Q \} \\
\{ P \} & \quad c_1; c_2 \quad \{ Q \} \\
\{ P \} \quad & \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2 \quad \{ Q \}
\end{align*}
\]
Hoare Rules

\[
\begin{align*}
\{P\} & \text{SKIP} \quad \{P\} \\
\{P[x \mapsto e]\} & \quad x := e \quad \{P\} \\
\{P\} & \quad c_1 \quad \{R\} \quad \{R\} \quad c_2 \quad \{Q\} \\
\{P\} & \quad c_1 ; c_2 \quad \{Q\} \\
\{P \land b\} & \quad c_1 \quad \{Q\} \quad \{P \land \neg b\} \quad c_2 \quad \{Q\} \\
\{P\} & \quad \text{IF} \ b \ \text{THEN} \ c_1 \ \text{ELSE} \ c_2 \quad \{Q\}
\end{align*}
\]
Hoare Rules

\{
P\}\ SKIP \{P\}

\{P[ x \mapsto e]\}\ x := e \{P\}

\{P\} c_1 \{R\} \{R\} c_2 \{Q\}

\{P\} \ c_1; c_2 \{Q\}

\{P \land b\} c_1 \{Q\} \{P \land \neg b\} c_2 \{Q\}

\{P\} \ IF \ b \ THEN \ c_1 \ ELSE \ c_2 \{Q\}

\{P \land b\} c \{P\} \ P \land \neg b \implies Q

\{P\} \ WHILE \ b \ DO \ c \ OD \{Q\}
Hoare Rules

\[
\begin{align*}
\{P\} & \text{ SKIP } \{P\} & \{P[x \mapsto e]\} & \text{ \texttt{x := e} } & \{P\} \\
\{P\} & c_1 \{R\} \quad \{R\} & c_2 \{Q\} & \{P\} & c_1; c_2 \{Q\} \\
\{P \land b\} & c_1 \{Q\} \quad \{P \land \neg b\} & c_2 \{Q\} & \{P\} & \text{ IF } b \text{ THEN } c_1 \text{ ELSE } c_2 \{Q\} \\
\{P \land b\} & c \{P\} \quad P \land \neg b \implies Q & \{P\} & \text{ WHILE } b \text{ DO } c \text{ OD } \{Q\}.
\end{align*}
\]
Hoare Rules

\[
\begin{align*}
\{P\} & \text{ SKIP } \{P\} & \{P[x \mapsto e]\} & x := e & \{P\} \\
\{P\} & c_1 \{R\} & \{R\} & c_2 \{Q\} & \\
\{P\} & c_1; c_2 & \{Q\} \\
\{P \land b\} & c_1 \{Q\} & \{P \land \neg b\} & c_2 \{Q\} \\
\{P\} & \text{ IF } b \text{ THEN } c_1 \text{ ELSE } c_2 & \{Q\} \\
\{P \land b\} & c \{P\} & P \land \neg b & \implies Q \\
\{P\} & \text{ WHILE } b \text{ DO } c \text{ OD } \{Q\} \\
P & \implies P' & \{P'\} & c \{Q'\} & Q' & \implies Q \\
\{P\} & c & \{Q\}
\end{align*}
\]
Hoare Rules

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

\vdash \{P\} \quad c_1 \quad \{R\} \quad \vdash \{R\} \quad c_2 \quad \{Q\}

\vdash \{P\} \quad c_1; c_2 \quad \{Q\}

\vdash \{\lambda \sigma. P \sigma \land b \sigma\} \quad c_1 \quad \{Q\} \quad \vdash \{\lambda \sigma. P \sigma \land \neg b \sigma\} \quad c_2 \quad \{Q\}

\vdash \{P\} \quad \text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2 \quad \{Q\}

\vdash \{\lambda \sigma. P \sigma \land b \sigma\} \quad c \quad \{P\} \quad \land \sigma. P \sigma \land \neg b \sigma \implies Q \sigma

\vdash \{P\} \quad \text{WHILE } b \text{ DO } c \text{ OD} \quad \{Q\}

\land \sigma. P \sigma \implies P' \sigma \quad \vdash \{P'\} \quad c \quad \{Q'\} \quad \land \sigma. Q' \sigma \implies Q \sigma

\vdash \{P\} \quad c \quad \{Q\}
Are the Rules Correct?

Soundness: \( \vdash \{ P \} \ c \ \{ Q \} \implies \models \{ P \} \ c \ \{ Q \} \)
Are the Rules Correct?

**Soundness:** \( \vdash \{P\} \ c \ \{Q\} \Rightarrow \models \{P\} \ c \ \{Q\} \)

**Proof:** by rule induction on \( \vdash \{P\} \ c \ \{Q\} \)
Are the Rules Correct?

**Soundness:** \( \vdash \{ P \} \ c \ \{ Q \} \Rightarrow \models \{ P \} \ c \ \{ Q \} \)

**Proof:** by rule induction on \( \vdash \{ P \} \ c \ \{ Q \} \)

**Demo:** Hoare Logic in Isabelle