20. Axiomatic Semantics

Goal: Prove that a program achieves the effect it was designed for.
So far, we considered a single initial state.
To prove correctness, we have to consider all inputs
(that are appropriate to us)

Approach: • Establish statements of the form
If before the execution of the program c we have $x \leq 1$,
then after the execution we obtain $y > 2$
– provided the program terminates.

• This is a statement of partial correctness, where partial refers to the fact that the program does not need to terminate, in which case no guarantees are given.
• To enforce termination, total correctness statements are used:
If before the execution of the program c we have $x \leq 1$,
then c is guaranteed to terminate and we obtain $y > 2$.
Total correctness is beyond the scope of this lecture.
• We denote partial correctness statements as Hoare triples of the form $\{x \leq 1\}\, c\, \{y > 2\}$,
named after Sir Tony Hoare (* 1934, Turing award 1980).
In a Hoare triple like the above
  • $x \leq 1$ is called the precondition.
    It specifies the set of initial states σ we are interested in.
  • $y > 2$ is the postcondition,
    the guarantee we require from the states σ' reached upon termination (only one, if c is deterministic).

1- Preconditions and postconditions are also called assertions.

History: • This is the old school of (manual) program verification.
It goes back to Turing's paper: Checking a large routine, 1949
The pioneers were Robert Floyd (1936–2001, Turing award 1978)
and Tony Hoare.

• Originally, the method was not only developed
for proving properties of programs,
but also to explain the meaning of programming constructs.
The meaning of a construct was given in terms of
↪ axioms in case of commands and
↪ rules in case of composition operators,
saying how to prove properties about the construct.
Therefore, the approach is traditionally called axiomatic semantics.
The term program verification may be more common nowadays.

We need a language of assertions.
The idea is to use first-order logic over the Boolean expressions.
One may think about extending the signature of the program
to include further function symbols and predicate symbols in assertions.
Moreover, one could extend the domain beyond the data elements
used in the program (auxiliary information, ghost/auxiliary variables).

Definition:
• Assertions are first-order formulas built over the signature of programs:
$A ::= \text{true} \mid \text{false} \mid a_1 < a_2 \mid \neg A \mid A_1 \wedge A_2 \mid \exists i.\, A.$
Here, i is an integer variable, different from the variables in the program.
Moreover, a_1 and a_2 are arithmetic expressions, potentially over integer variables.
We assume assertions to be closed wrt. integer variables
(all integer variables are quantified).

- We write $\mathcal{S}, \sigma \models A$, if $\mathcal{S}[\![A]\!](\sigma) = \text{true}$.
  Since $\mathcal{S}$ is fixed, we also write $\sigma \models A$ and say that σ satisfies A.
  We use $\mathcal{S}[\![A]\!] := \{\sigma \in State \mid \sigma \models A\}$ for
  the set of states that satisfy A.
- A partial correctness statement, also called a Hoare triple,
  has the form
  $$\{A\}\, c\, \{B\},$$
  where A and B are assertions and c is a program.
- A partial correctness statement holds or is valid,
  denoted by $\models \{A\}\, c\, \{B\}$, if
  $$\forall \sigma, \sigma' \in State: \sigma \models A \wedge (c, \sigma) \Downarrow \sigma' \Rightarrow \sigma' \models B.$$
  Using the denotational semantics, this can be written as
  $$C[\![c]\!](\mathcal{S}[\![A]\!]) \subseteq \mathcal{S}[\![B]\!].$$

Note:
- This definition also works for non-deterministic programs.
- If a program does not terminate, or if the precondition does not hold,
  the partial correctness statement is trivially valid.
- Although we use the term valid from logic,
  we are interested in the fixed structure $\mathcal{S}$ that the program is interpreted in.
  Therefore, a more precise notion would be to say that
  a partial correctness statement is $\mathcal{S}$-valid.
  Since there is no risk of confusion, we will stick with the term valid.

Example: (1) Let $c =$ while true do skip od.
Then $\models \{\text{true}\}\, c\, \{A\}$ holds for all assertions A.
The reason is that the program does not terminate.

(2) Let $c =$ *while* $x \leq 2$ *do* skip *od*.

Then $\vdash \{true\}\, c\, \{x > 2\}$,

since the program terminates only for $x > 2$.

(3) $\vdash \{s = 0,\ n = 1\}$

while $\neg(n = 101)$ *do*

$s := s + n;$

$n := n + 1;$

od

$\{s = \sum_{i=1}^{100} i\}$.

20.1 A Proof System for Partial Correctness

Goal: · Validity of Hoare triples is a semantic problem.

· To check validity algorithmically (with the help of a computer)

we need a syntactic counterpart.

· Therefore, we introduce a proof system (calculus)

the theorems provable in which are precisely the valid Hoare triples.

Definition (Hoare '69):

The rules of Hoare's proof system, Hoare rules for short,

are as follows:

(skip) $\dfrac{}{\{A\}\, skip\, \{A\}}$ (assign) $\dfrac{}{\{A[x/a]\}\, x := a\, \{A\}}$

(seq) $\dfrac{\{A\}\, c_1\, \{C\} \quad \{C\}\, c_2\, \{B\}}{\{A\}\, c_1; c_2\, \{B\}}$ (if) $\dfrac{\{A \wedge b\}\, c_1\, \{B\} \quad \{A \wedge \neg b\}\, c_2\, \{B\}}{\{A\}\, \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ fi}\, \{B\}}$

(while) $\dfrac{\{A \wedge b\}\, c\, \{A\}}{\{A\}\, \text{while } b \text{ do } c \text{ od}\, \{A \wedge \neg b\}}$ (consequence) $\dfrac{A \to A' \quad \{A'\}\, c\, \{B'\} \quad B' \to B}{\{A\}\, c\, \{B\}}$.

If Herr is a proof for $\{A\}\, c\, \{B\}$, we write $\vdash \{A\}\, c\, \{B\}$.

Comment:

- The rules are syntax-directed, they reduce proving partial correctness of a composed program to proving partial correctness of the components.
- The rule for assignments is indeed the right way around.

Consider $x := x+2$.
What is the requirement we need to guarantee $x>2$?
The rule states:

$$\{(x>2)[x/x+2]\}\ x := x+2\ \{x>2\}.$$

Indeed,

$$(x>2)[x/x+2] = x+2>2 \iff x>0.$$

- In the Rule (while), I is called a loop invariant, because the premise $\{I \wedge b\}\, c\, \{I\}$ says that the assertion I is preserved by a full execution of the loop body. Executing the loop is only possible when satisfying b. Hence, either the loop diverges, in which case we can assume $\neg b$ as a postcondition, or it terminates because b fails.
- The Rule (consequence) makes use of valid implications in the premise. Technically, proving such implications can be hard (undecidable). Fortunately, common programs do not involve deep Mathematics and therefore proving such implications is often easy.
- Intuitively, Rule (consequence) holds because we strengthen the precondition (so that it is satisfied by less states) and we weaken the postcondition (so that it is satisfied by more states). Rule (consequence) can be understood as the interface between program verification and logic.

Loop invariants:

Coming up with an appropriate loop invariant is difficult.

As we will see, program verification can be fully automated
↳ except for the step of finding loop invariants
↳ and under the assumption that the underlying logical reasoning can be automated.

Example (Factorial function):

Consider $w =$ **while** $x > 0$ **do**
$y := y \cdot x;$
$x := x - 1;$
od

Our goal is to show that w computes the factorial function
$n! = n(n-1)(n-2)\ldots 2 \cdot 1$, where $0! := 1$.

More precisely, we wish to prove the Hoare triple:
$\vdash \{x = n \wedge n \geq 0 \wedge y = 1\}\, w\, \{y = n!\}$.

- Since we will have to invoke the proof rule for while-programs, we need a loop invariant:

$$I := (y \cdot x! = n! \wedge x \geq 0).$$

To show that I is indeed an invariant, we need

$$\{I \wedge x > 0\}\, y := y \cdot x;\ x := x - 1\, \{I\}.$$

By Rule (assign):

$$\{I[x/x-1]\}\, x := x - 1\, \{I\},$$

where $I[x/x-1] = y \cdot (x-1)! = n! \wedge x - 1 \geq 0$.

Again by Rule (assign):

$$\{y \cdot x \cdot (x-1)! = n! \wedge x - 1 \geq 0\}\, y := y \cdot x\, \{y \cdot (x-1)! = n! \wedge x - 1 \geq 0\}$$

By Rule (seq):

$$\{y \cdot x \cdot (x-1)! = n! \wedge x - 1 \geq 0\}\, y := y \cdot x;\ x := x - 1\, \{I\}.$$

We have $I \wedge x > 0 \Rightarrow y \cdot x! = n! \wedge x \geq 0 \wedge x > 0$

$\Rightarrow y \cdot x! = n! \wedge x \geq 1$

$\Rightarrow y \cdot x \cdot (x-1)! = n! \wedge x - 1 \geq 0.$

Hence, Rule (consequence) applies and gives

$\{I \wedge x > 0\}\ y := y \cdot x;\ x := x - 1\ \{I\}.$

We now apply Rule (while) and get

$\{I\}\ w\ \{I \wedge \neg(x > 0)\}.$

- To establish the desired statement, we <u>strengthen the precondition</u>

$x = n \wedge n \geq 0 \wedge y = 1 \Rightarrow \underbrace{y \cdot x! = n! \wedge x \geq 0}_{I}$

Moreover, we <u>weaken the postcondition</u>:

$I \wedge \neg(x > 0) \Rightarrow y \cdot x! = n! \wedge x \geq 0 \wedge \neg(x > 0)$

$\Rightarrow y \cdot x! = n! \wedge x = 0$

$\Rightarrow y \cdot 0! = n!$

$\Rightarrow y = n!$

By Rule (consequence), we get

$\{x = n \wedge n \geq 0 \wedge y = 1\}\ w\ \{y = n!\}.$

<u>Comments:</u>

- It is often easier to proceed from right to left when proving facts about a sequence of commands.
- Why did we include $x \geq 0$ in the invariant?
  Because we needed $x = 0$ when we left the loop.
  One often has to strengthen loop invariants,
  they are like induction hypotheses.
  A good idea is to strengthen the invariant by information about the variable used in Boolean expressions (<u>if</u>- and <u>while</u>-conditions).
- Automating proofs like the above is the research interest of our group.

20.2 Weakest Liberal Preconditions and Expressiveness

Goal: Show that Hoare's proof system is complete:

$\models \{A\}\, c\, \{B\}$ implies $\vdash \{A\}\, c\, \{B\}$.

Problem: We would like to proceed by induction on the structure of programs.

Then, when considering

$\models \{A\}\, c_1; c_2\, \{B\}$,

to invoke the induction hypothesis,

we need a predicate C with

$\models \{A\}\, c_1\, \{C\}$ and $\models \{C\}\, c_2\, \{B\}$.

How do we know that such an assertion C exists?

Solution: For every program c and postcondition B

we need to be able to represent the set of states

↳ from which the computation diverges or

↳ from which it terminates in a state satisfying B

by means of an assertion.

Definition:

Given a program c and a postcondition B,

the weakest liberal precondition of B wrt. c

is the set of states

$$wlp(c, B) := \{\sigma \in State \mid \forall \sigma' \in State \text{ with } (c, \sigma) \to \sigma' : \sigma' \models B\}.$$

- The term liberal refers to the fact that we deal with partial correctness.

For total correctness, the corresponding definition is called weakest precondition.

· It is not clear that our language of assertions contains an element A that characterizes $wlp(c, B)$.

Definition:

An assertion language $\mathcal{L}$ is *expressive*.

i) for all $c \in Prog$ and $B \in \mathcal{L}$, there is an $A \in \mathcal{L}$ with $[\![A]\!] = wlp(c, B)$.

$[\![A]\!] = \{\sigma \in State \mid \sigma \models A\}$

Lemma:

Let $[\![A]\!] = wlp(c, B)$.

Then (1) $\{A\}\, c\, \{B\}$ is valid and

(2) If $\{A'\}\, c\, \{B\}$ is valid, then $A' \Rightarrow A$.

Statement (1) says that A is indeed a precondition that makes the Hoare triple valid.

Statement (2) says that A is the *weakest* (wrt. $\Rightarrow$) precondition that makes the Hoare triple valid

(in the assertion language, ordered by implication, and factorized along $\Leftrightarrow$)

Proof: Homework.

Theorem (Dijkstra '76, Edsger Dijkstra, 1930–2002, Turing award 1972):

The language of assertions defined above is expressive.

· The theorem on the one hand states that such a weakest assertion (that makes the Hoare triple valid) *exists*.

· Moreover, it shows how to *compute* the weakest assertion.

We focus here on *acyclic* programs.

Handling while is non-trivial and beyond the scope of this lecture (needs Gödel's β-predicate, see Winskel's book).

Proof:

- We give the weakest assertion via the function $pred(c, B)$:

$$pred(skip, B) := B$$
$$pred(x := a, B) := B[x/a]$$
$$pred(c_1; c_2, B) := pred(c_1, pred(c_2, B))$$
$$pred(\text{if } b \text{ then } c_1 \text{ else } c_2 \text{ fi}, B) := (b \wedge pred(c_1, B)) \vee (\neg b \wedge pred(c_2, B)).$$

- One can check that

$$\forall \sigma \in State: \ \sigma \models pred(c, B) \iff \sigma \in wlp(c, B).$$

Hence $[\![pred(c, B)]\!] = wlp(c, B)$. □

20.3 Completeness and Soundness

Goal: . With expressiveness at hand, we can prove Hoare's proof system complete.

. Soundness is easier.

The following is the key lemma.

Lemma:

$$\vdash \{pred(c, B)\}\ c\ \{B\}.$$

Proof:

We proceed by induction on the structure of c and show that $\forall B$ an assertion: $\vdash \{pred(c, B)\}\ c\ \{B\}$.

Base case: $c = skip$

We have $pred(skip, B) = B$ and $\{B\}\ skip\ \{B\}$ by Hoare (skip).

$c = x := a$ Similar, by the axiom for assignments.

Induction: Assume the claim holds for c_1, c_2 and c.

Step

$c_1; c_2$ We have

$$pred(c_1; c_2, B) = pred(c_1, pred(c_2, B)).$$

By the induction hypothesis:

$$\vdash \{pred(c_2, B)\}\ c_2\ \{B\} \quad \text{and}$$

$$\vdash \{pred(c_1, pred(c_2, B))\}\ c_1\ \{pred(c_2, B)\}.$$

By Rule (seq):

$$\vdash \{\underbrace{pred(c_1, pred(c_2, B))}_{= pred(c_1; c_2, B)}\}\ c_1; c_2\ \{B\}.$$

$\underline{\text{if } b \text{ then } c_1 \text{ else } c_2 \text{ fi}}$

By the induction hypothesis:

$$\vdash \{pred(c_1, B)\}\ c_1\ \{B\} \quad \text{and} \tag{*}$$

$$\vdash \{pred(c_2, B)\}\ c_2\ \{B\}.$$

Moreover:

$$pred(\text{if } b \text{ then } c_1 \text{ else } c_2 \text{ fi}, B) \wedge b$$

$$= [(b \wedge pred(c_1, B)) \vee (\neg b \wedge pred(c_2, B))] \wedge b$$

$$\Leftrightarrow (b \wedge pred(c_1, B)) \vee (\underbrace{\neg b \wedge b \wedge pred(c_2, B)}_{\Leftrightarrow \text{ false}}) \quad \text{(Distributivity)} \tag{**}$$

$$\Leftrightarrow b \wedge pred(c_1, B).$$

Similarly:

$$pred(\text{if } b \text{ then } c_1 \text{ else } c_2 \text{ fi}, B) \wedge \neg b \Leftrightarrow \neg b \wedge pred(c_2, B). \tag{***}$$

Combining (*) with (**), and using Rule (consequence):

$$\vdash \{pred(\text{if } b \text{ then } c_1 \text{ else } c_2 \text{ fi}, B) \wedge b\}\ c_1\ \{B\}.$$

Similarly, (*) and (***) give

$$\vdash \{pred(\underline{if}\ b\ \underline{then}\ c_1\ \underline{else}\ c_2\ \underline{fi}, B) \wedge \neg b\}\ c_2\ \{B\}.$$

By Rule (if):

$$\vdash \{pred(\underline{if}\ b\ \underline{then}\ c_1\ \underline{else}\ c_2\ \underline{fi}, B)\}\ \underline{if}\ b\ \underline{then}\ c_1\ \underline{else}\ c_2\ \underline{fi}\ \{B\}.$$

$w = \underline{while}\ b\ \underline{do}\ c\ \underline{od}$

Take $pred(w, B)$ that exists by Dijkstra's theorem.

We show

(1) $\vdash \{pred(w, B) \wedge b\}\ c\ \{pred(w, B)\}$.

(2) $pred(w, B) \wedge \neg b \Rightarrow B$.

By Rule (while), (1) yields:

$$\vdash \{pred(w, B)\}\ w\ \{pred(w, B) \wedge \neg b\}.$$

By Rule (consequence) and (2):

$$\vdash \{pred(w, B)\}\ w\ \{B\}.$$

Concerning (1):

By the induction hypothesis:

$$\vdash \{pred(c, pred(w, B))\}\ c\ \{pred(w, B)\}.$$

By definition of $pred(c; w, B)$, this means

$$\vdash \{pred(c; w, B)\}\ c\ \{pred(w, B)\}.$$

Since

$$C[\![w]\!] = C[\![\underline{if}\ b\ \underline{then}\ c; w\ \underline{else}\ skip\ \underline{fi}]\!],$$

we moreover have

$$pred(w, B) \Leftrightarrow pred(\underline{if}\ b\ \underline{then}\ c; w\ \underline{else}\ skip\ \underline{fi}, B).$$

If we add b, this yields

$$pred(w, B) \wedge b$$

(pred for if) $\Leftrightarrow [[b \wedge pred(c;w, B)) \vee (\neg b \wedge pred(skip, B))] \wedge b$

(Distributivity) $\Leftrightarrow b \wedge pred(c;w, B).$

From $\vdash \{pred(c;w, B)\}\, c\, \{pred(w, B)\}$

and $pred(w, B) \wedge b \Rightarrow pred(c;w, B)$,

we conclude with Rule (consequence):

$$\vdash \{pred(w, B) \wedge b\}\, c\, \{pred(w, B)\}.$$

Concerning (2):

Consider $\sigma \models pred(w, B) \wedge \neg b$.

Then for all $\sigma' \in State$ with $(w, \sigma) \Downarrow \sigma'$ we have $\sigma' \models B$.

We have $C[\![w]\!] = C[\![$ if b then $c;w$ else skip fi$]\!]$

Since $\sigma \models \neg b$, this means $(skip, \sigma) \Downarrow \sigma'$.

This in turn means $\sigma' = \sigma$ and therefore $\sigma \models B$. □

Theorem (Completeness, Cook '74, Steven Cook, *1939, Turing award 1982)

$\models \{A\}\, c\, \{B\}$ implies $\vdash \{A\}\, c\, \{B\}$.

Proof:

We have $\vdash \{pred(c, B)\}\, c\, \{B\}$ by the Soundness lemma.

Moreover, $A \Rightarrow pred(c, B)$ by the lemma on weakest liberal preconditions.

Hence, by Rule (consequence):

$$\vdash \{A\}\, c\, \{B\}.$$

□

Soundness of the Hoare rules states that every theorem $\vdash \{A\}\, c\, \{B\}$ (derived with the proof rules) is a valid Hoare triple.

Theorem (Soundness):

$\vdash \{A\}\, c\, \{B\}$ implies $\models \{A\}\, c\, \{B\}$.

The proof is a routine induction on the height of the proof trees.

20.3 Relative Completeness

We say that a proof system is *effective*,
if there is a semi-decision procedure for rule instantiation:

- There is an algorithm that, given a potential rule instance, terminates and returns "yes" for *actual* rule instances.
- The method does not have to terminate or it should return "no" for *non*-instances (negative cases).

- Is the above proof system effective?

The answer is *no*!
The reason is *Gödel's incompleteness theorem* (see Logic):
There is no effective proof system the theorems provable in which are precisely (sound and complete) the valid assertions. (*)

Nowadays, Gödels result (*) follows from computability theory:
There is no semi-decider for validity of assertions (**)
(and an effective proof system would yield a semi-decider).

Statement (**) is not too difficult:
Since the structure the program is interpreted over is fixed to be arithmetic $\mathbb{S}$, we have $\neg\, \mathbb{S} \models A$ iff $\mathbb{S} \models \neg A$.

So a semi-decider for valid assertions would already be a decision procedure.

From (**) and the fact that

$$\models B \quad \text{iff} \quad \models \text{true} \to B,$$

we can conclude that the applications of Rule (consequence) are <u>not effective</u>.

- One could hope that there is a <u>better proof system</u> than the one of Hoare that is effective, sound, and complete. But due to

$$\models B \quad \text{iff} \quad \models \{\text{true}\}\ \text{skip}\ \{B\},$$

this would lead to an effective, sound, and complete proof system for arithmetic, contradicting Gödel's theorem (*).

- There is a different proof for the fact that there is no effective, sound, and complete proof system for the valid Hoare triples that uses computability. If there was such a proof system, we would have a semi-decider for the valid $\{A\}\, c\, \{B\}$. But

$$c \text{ diverges on all inputs} \quad \text{iff} \quad \models \{\text{true}\}\, c\, \{\text{false}\}.$$

Since divergence on all inputs is <u>not semi-decidable</u> (this is a theorem) there cannot be an effective, sound, and complete proof system for the valid Hoare triples.

- In this sense, our proof system is the best we can hope for: It is complete – <u>relative to</u> (assuming an oracle for) reasoning about arithmetic.