Chapter 1

Classical Program Logics: Hoare Logic, Weakest Liberal Preconditions

1.1 The IMP Language

IMP is a programming language with an extensible syntax that was developed in the late 1960s. We will consider only a subset of this simple imperative language. Its purpose is to present the fundamental notions of deductive program verification.

1.1.1 Syntax

The language provides global variables of type integer, integer expressions, assignments, and standard structured statements (sequence, conditional, and loop). The grammar of expressions and statements is as follows, where n denotes an integer constant and x a variable identifier.

$$\begin{array}{rcl} e & ::= & n \mid x \mid e\ op\ e \\ op & ::= & + \mid - \mid * \mid = \mid \neq \mid < \mid > \mid \leq \mid \geq \mid and \mid or \\ s & ::= & \texttt{skip} \mid x := e \mid s; s \mid \texttt{if}\ e\ \texttt{then}\ s\ \texttt{else}\ s \mid \texttt{while}\ e\ \texttt{do}\ s \end{array}$$

Remarks:

- There is only one data type: integers. They will have their mathematical meaning, that is, they are unbounded, unlike machine integers.
- The relational operators return an integer: 0 meaning "false" and -1 meaning "true".
- The condition in if and while statements interprets 0 as "false" and non-zero as "true"
- There is no division operator.
- A conditional without "else" branch is syntactic sugar for an "else skip".

Consequence of these remarks:

- Expressions always evaluate without error.
- Expressions have no side effect; statements do.
- Since there is only one type, all programs are well-typed.

- There is no possible runtime error: all programs execute until their end or infinitely (see the semantics below).

Example 1.1.1 *The following program ISQRT operates on three global variables n, $count$, and sum:*

```
count := 0; sum := 1;
while sum <= n do count := count + 1; sum := sum + 2 * count + 1 done
```

A property that we would like to formally establish is that, at the end of execution of this program, $count$ contains the square root of n, rounded downward, e.g. for $n = 42$, the final value of $count$ is 6.

1.1.2 Operational Semantics

We formalize the operational semantics of our language using a standard small-step semantics. This follows a weel-known scheme known as structural operational semantics (SOS) originality due to Plotkin in 1980 [6].

A *program state* describes the content of global variables at a given time. It can be modeled by a finite map Σ associating to each variable x its current value denoted $\Sigma(x)$. The value of an expression e in some state Σ, denoted $[\![e]\!]_\Sigma$, is always defined, by the following recursive equations.

$$\begin{aligned} [\![n]\!]_\Sigma &= n \\ [\![x]\!]_\Sigma &= \Sigma(x) \\ [\![e_1 \ op \ e_2]\!]_\Sigma &= [\![e_1]\!]_\Sigma \ [\![op]\!] \ [\![e_2]\!]_\Sigma \end{aligned}$$

where $[\![op]\!]$ is the natural semantic of operator op on integers (with relational operators returning 0 for false and -1 for true).

The semantics of statements is defined via the judgment $\Sigma, s \rightsquigarrow \Sigma', s'$ meaning that, in state Σ, executing one step of statement s leads to the state Σ' and the remaining statement to execute is s'. The semantics is defined by the following rules.

$$\frac{}{\Sigma, x := e \rightsquigarrow \Sigma\{x \leftarrow [\![e]\!]_\Sigma\}, \texttt{skip}}$$

$$\frac{}{\Sigma, (\texttt{skip}; s) \rightsquigarrow \Sigma, s}$$

$$\frac{\Sigma, s_1 \rightsquigarrow \Sigma', s_1'}{\Sigma, (s_1; s_2) \rightsquigarrow \Sigma', (s_1'; s_2)}$$

$$\frac{[\![e]\!]_\Sigma \neq 0}{\Sigma, \texttt{if } e \texttt{ then } s_1 \texttt{ else } s_2 \rightsquigarrow \Sigma, s_1} \qquad \frac{[\![e]\!]_\Sigma = 0}{\Sigma, \texttt{if } e \texttt{ then } s_1 \texttt{ else } s_2 \rightsquigarrow \Sigma, s_2}$$

$$\frac{[\![e]\!]_\Sigma \neq 0}{\Sigma, \texttt{while } e \texttt{ do } s \rightsquigarrow \Sigma, (s; \texttt{while } e \texttt{ do } s)} \qquad \frac{[\![e]\!]_\Sigma = 0}{\Sigma, \texttt{while } e \texttt{ do } s \rightsquigarrow \Sigma, \texttt{skip}}$$

Remark that, with these rules, any statement different from `skip` can execute in any state. In other words, the statement `skip` alone represents the final step of execution of a program, and there is no possible *runtime error*.

Since $\rightsquigarrow$ is a relation between pairs of state and statement, there is a transitive closure $\rightsquigarrow^+$ and a reflexive-transitive closure $\rightsquigarrow^*$. In other words, $\Sigma, s \rightsquigarrow^* \Sigma', s'$ means that statement s, in state Σ, reaches state Σ' with remaining statement s' after executing some finite number of steps.

We say that the execution of a statement s in some state Σ *terminates* if there is a state Σ' such that $\Sigma, s \rightsquigarrow^* \Sigma', \texttt{skip}$. Notice that, since there are no possible runtime errors, if there is no such Σ', then s executes infinitely.

Lemma 1.1.2 (Sequence execution) *For any terminating execution $\Sigma, (s_1; s_2) \rightsquigarrow^* \Sigma', \texttt{skip}$ of a sequence, there exists an* intermediate state Σ'' *such that $\Sigma, s_1 \rightsquigarrow^* \Sigma'', \texttt{skip}$ and $\Sigma'', s_2 \rightsquigarrow^* \Sigma', \texttt{skip}$.*

Proof. Straightforward induction on the number of steps of the sequence.

1.2 Program Specifications, Hoare Logic

Historically, the notions of this section were introduced by Floyd [4] and Hoare [5].

1.2.1 Propositions about Programs

To formally express properties of programs, we need a logic language. We use standard first-order logic for this purpose. One specific point, however, is that the propositions of the logic can talk about program variables. More formally, propositions are interpreted with respect to a given program state.

Our syntax for propositions is

$$p \quad ::= \quad e \mid p \wedge p \mid p \vee p \mid \neg p \mid p \Rightarrow p \mid \forall v,\ p \mid \exists v,\ p$$

where v denotes logical variable identifiers, and e denotes program expressions defined as before, augmented with these logical variables.

The semantics of a proposition p in a program state Σ is denoted $[\![p]\!]_\Sigma$. It is a logic formula where no program variables appear anymore: they have been replaced by their values in Σ. It is defined recursively as follows.

$$\begin{array}{rcl} [\![e]\!]_\Sigma & = & [\![e]\!]_\Sigma \neq 0 \\ [\![p_1 \ \wedge \ p_2]\!]_\Sigma & = & [\![p_1]\!]_\Sigma \ \wedge \ [\![p_2]\!]_\Sigma \\ & \vdots & \end{array}$$

where semantics of expressions is augmented with

$$[\![v]\!]_\Sigma \quad = \quad v$$

We denote by $\Sigma \models p$ the fact the formula $[\![p]\!]_\Sigma$ is valid, and we denote $\models p$ when $\Sigma \models p$ holds for any state Σ.

1.2.2 Hoare Triples

A *Hoare triple* is a triple denoted $\{P\}s\{Q\}$ where P and Q are logic propositions and s a statement. P is called the *precondition* and Q the *postcondition*.

Definition 1.2.1 (Partial correctness of a program) *A Hoare triple $\{P\}s\{Q\}$ is said* valid *if for any states Σ, Σ' such that $\Sigma, s \rightsquigarrow^* \Sigma', \texttt{skip}$ and $[\![P]\!]_\Sigma$ holds, then $[\![Q]\!]_{\Sigma'}$ holds. In other words, if s is executed in a state satisfying its precondition, then* if it terminates*, the resulting state satisfies its postcondition.*

Example 1.2.2 *Examples of valid triples for partial correctness:*

- $\{x = 1\}x := x + 2\{x = 3\}$
- $\{x = y\}x := x + y\{x = 2 * y\}$
- $\{\exists v,\ x = 4 * v\}x := x + 42\{\exists w,\ x = 2 * w\}$
- $\{true\}\texttt{while}\ 1\ \texttt{do}\ \texttt{skip}\{false\}$.
- $\{n \geq 0\}ISQRT\{count * count \leq n \wedge n < (count + 1) * (count + 1)\}$

1.2.3 Hoare Logic

Hoare logic is defined by a set of inference rules producing triples.

$$\frac{}{\{P\}\texttt{skip}\{P\}} \qquad \frac{\{P \wedge e \neq 0\}s_1\{Q\} \qquad \{P \wedge e = 0\}s_2\{Q\}}{\{P\}\texttt{if } e \texttt{ then } s_1 \texttt{ else } s_2\{Q\}}$$

$$\frac{}{\{P[x \leftarrow e]\}x := e\{P\}} \qquad \frac{\{I \wedge e \neq 0\}s\{I\}}{\{I\}\texttt{while } e \texttt{ do } s\{I \wedge e = 0\}}$$

$$\frac{\{P\}s_1\{Q\} \qquad \{Q\}s_2\{R\}}{\{P\}s_1; s_2\{R\}} \qquad \frac{\{P'\}s\{Q'\} \qquad \models P \Rightarrow P' \qquad \models Q' \Rightarrow Q}{\{P\}s\{Q\}}$$

where $P[x \leftarrow e]$ denotes the formula obtained by syntactically replacing all occurrences of the program variable x by e. In the rule for the while loop, I is traditionally called a *loop invariant*.

Theorem 1.2.3 (Soundness of Hoare logic) *This set of rules is correct: any derivable triple is valid.*

Proof. This is proved by induction on the derivation tree of the considered triple. Thus, for each rule, assuming that the triples in premises are valid, we show that the triple in conclusion is valid too. The proofs are straightforward except for the sequence and while rules, that we detail now.

For the sequence: let us assume that the triples $\{P\}s_1\{Q\}$ and $\{Q\}s_2\{R\}$ are valid. To show that $\{P\}s_1; s_2\{R\}$ is valid, let us consider some state Σ such that $[\![P]\!]_\Sigma$ holds and some execution $\Sigma, (s_1; s_2) \rightsquigarrow^* \Sigma', \texttt{skip}$. By Lemma 1.1.2 (Sequence execution), we have an intermediate step Σ'' such that $\Sigma, s_1 \rightsquigarrow^* \Sigma'', \texttt{skip}$ and $\Sigma'', s_2 \rightsquigarrow^* \Sigma', \texttt{skip}$. Since $[\![P]\!]_\Sigma$ holds and $\{P\}s_1\{Q\}$ is valid, $[\![Q]\!]_{\Sigma''}$ holds, and then since $\{Q\}s_2\{R\}$ is valid, $[\![R]\!]_{\Sigma'}$ holds, q.e.d.

For the while loop: let us assume that $\{I \wedge e \neq 0\}s\{I\}$ is valid. To show that $\{I\}\texttt{while } e \texttt{ do } s\{I \wedge e = 0\}$ is valid, let us consider some state Σ such that $[\![I]\!]_\Sigma$ holds and some execution $\Sigma, \texttt{while } e \texttt{ do } s \rightsquigarrow^* \Sigma', \texttt{skip}$. We proceed by induction on the number of steps of this execution. We have two cases depending on whether the condition $[\![e]\!]_\Sigma$ is 0 or not. If it is 0 then the execution terminates in just one step, and $\Sigma' = \Sigma$, hence $[\![I \wedge e = 0]\!]_{\Sigma'}$ holds. If the condition is not 0, then the execution has the form $\Sigma, \texttt{while } e \texttt{ do } s \rightsquigarrow \Sigma, (s; \texttt{while } e \texttt{ do } s) \rightsquigarrow^* \Sigma', \texttt{skip}$. Again using the Sequence execution lemma, there is a state Σ'' such that $\Sigma, s \rightsquigarrow^* \Sigma'', \texttt{skip}$ and $\Sigma'', \texttt{while } e \texttt{ do } s \rightsquigarrow^* \Sigma', \texttt{skip}$. Since $[\![I \wedge e \neq 0]\!]_\Sigma$ holds and $\{I \wedge e \neq 0\}s\{I\}$ is valid, $[\![I]\!]_{\Sigma''}$ holds. By induction, since $\Sigma'', \texttt{while } e \texttt{ do } s \rightsquigarrow^* \Sigma', \texttt{skip}$ has fewer steps than the original execution, we get that $[\![I \wedge e = 0]\!]_{\Sigma'}$ holds, q.e.d.

1.2.4 Completeness

A major difficulty when trying to prove a program using Hoare logic rules is the need to guess the appropriate intermediate predicates, for example the intermediate predicate of a sequence, or the loop invariant for the while rule. For instance, our program ISQRT cannot be proved without a bit of thinking: one needs to discover a suitable loop invariant.

On a theoretical point of view, the question is the completeness of Hoare logic: are all valid triples derivable from the rules? The answer is given by the following theorem.

Theorem 1.2.4 (Completeness of Hoare logic) *The set of rules of Hoare logic is* relatively *complete: if the logic language is expressive enough, then any valid triple $\{P\}s\{Q\}$ can be derived using the rules.*

The logic in which annotations are written needs to be expressive enough, so that the loop invariants needed can be obtained, in theory. It is the case here since we have multiplication operator, hence Peano arithmetic (non-linear integer arithmetic). It is known that this logic has the expressive power of Turing

machines, hence whatever is computed by an IMP program, or an IMP loop, can be represented by a predicate of our logic [1].

Remark that the knowledge of the completeness gives only hints on how to effectively determine a suitable loop invariant when needed (see the theory of abstract interpretation [2]).

1.2.5 Frame Rule

By induction on a statement s, one can easily define the set of variables that are assigned in s, that is to say, they appear on the left of an assignment operator. One can then prove that if $\Sigma, s \rightsquigarrow^* \Sigma', s'$ and v is not assigned in s, then $[\![v]\!]_\Sigma = [\![v]\!]_{\Sigma'}$.

As a consequence, adding the following inference rule does not invalidate the soundness of Hoare logic:

$$\frac{\{P\}s\{Q\}}{\{P \wedge R\}s\{Q \wedge R\}}$$

with R a formula where no variables assigned in s occur.

This rule is not necessary since Hoare logic was already complete. In Section 1.3, we will, however, modify one of the rules in a lossy way: soundness will be preserved, but completeness will not. The frame rule will help to alleviate this loss.

Lemma 1.2.5 (Compressing consequence and frame rules) *For any derivable triple $\{P\}s\{Q\}$, there are some formula P', Q', and R, and a derivation tree that ends with*

$$\dfrac{\dfrac{\dfrac{\cdots}{\{P'\}s\{Q'\}}\ (\text{skip, assign, while, or seq})}{\{P' \wedge R\}s\{Q' \wedge R\}}\ (\text{frame}) \qquad \models P \Rightarrow (P' \wedge R) \quad \models (Q' \wedge R) \Rightarrow Q}{\{P\}s\{Q\}}\ (\text{consequence})$$

Proof. This lemma is proved by construction: the derivation tree of $\{P\}s\{Q\}$ is modified until it has the expected structure. First, notice that one can add the following derivation steps to deal with degenerate trees that would end with no frame step or no consequence step:

$$\dfrac{\dfrac{\{P\}s\{Q\}}{\{P \wedge \mathit{true}\}s\{Q \wedge \mathit{true}\}} \quad \models P \Rightarrow (P \wedge \mathit{true}) \quad \models (Q \wedge \mathit{true}) \Rightarrow Q}{\{P\}s\{Q\}}$$

Second, notice that two consecutive consequence steps can be merged into one:

$$\dfrac{\dfrac{\{P''\}s\{Q''\} \quad \models P' \Rightarrow P'' \quad \models Q'' \Rightarrow Q'}{\{P'\}s\{Q'\}} \qquad \models P \Rightarrow P' \quad \models Q' \Rightarrow Q}{\{P\}s\{Q\}}$$

becomes

$$\dfrac{\{P''\}s\{Q''\} \quad \models P \Rightarrow P'' \quad \models Q'' \Rightarrow Q}{\{P\}s\{Q\}}$$

There is a similar transformation for two consecutive frame steps.

Finally, notice that a consequence step can be moved after a frame step:

$$\dfrac{\dfrac{\{P'\}s\{Q'\} \quad \models P \Rightarrow P' \quad \models Q' \Rightarrow Q}{\{P\}s\{Q\}}}{\{P \wedge R\}s\{Q \wedge R\}}$$

becomes

$$\frac{\dfrac{\{P'\}s\{Q'\}}{\{P' \wedge R\}s\{Q' \wedge R\}} \quad \models (P \wedge R) \Rightarrow (P' \wedge R) \quad \models (Q' \wedge R) \Rightarrow (Q \wedge R)}{\{P \wedge R\}s\{Q \wedge R\}}$$

By applying all these transformations, one can transform the original deduction tree until left with a single frame step followed by a single consequence step at the bottom of the tree.

1.3 Weakest Liberal Preconditions

The notion of weakest precondition calculus was originally proposed by Dijkstra [3].

1.3.1 Annotated IMP Programs

Our aim is now to add more automation to the principles of Hoare logic, so that we can perform a proof of a specified program in a more systematic manner. Since we know that loop invariants must be discovered at some point, we augment our IMP language so that while statements contain a loop invariant as an annotation.

$$s \quad ::= \quad \texttt{skip} \mid x := e \mid s; s \mid \texttt{if}\ e\ \texttt{then}\ s\ \texttt{else}\ s \mid \texttt{while}\ e\ \texttt{invariant}\ I\ \texttt{do}\ s$$

The operational semantics is unchanged.

The Hoare logic rule for the loop is modified accordingly:

$$\frac{\{I \wedge e \neq 0\}s\{I\}}{\{I\}\texttt{while}\ e\ \texttt{invariant}\ I\ \texttt{do}\ s\{I \wedge e = 0\}}$$

Beware that by enforcing a fixed loop invariant, we lose completeness: if we choose a property I that poorly tracks the content of variables assigned in $\texttt{while}\ e\ \texttt{invariant}\ I\ \texttt{do}\ s$, some valid triples might not be derivable.

1.3.2 Weakest Liberal Preconditions Computation

We define a function $\mathrm{WLP}(s, Q)$ where s is a statement and Q a formula, using the following structurally recursive equations.

$$\begin{aligned}
\mathrm{WLP}(\texttt{skip}, Q) &= Q \\
\mathrm{WLP}(x := e, Q) &= Q[x \leftarrow e] \\
\mathrm{WLP}(s_1; s_2, Q) &= \mathrm{WLP}(s_1, \mathrm{WLP}(s_2, Q)) \\
\mathrm{WLP}(\texttt{if}\ e\ \texttt{then}\ s_1\ \texttt{else}\ s_2, Q) &= (e \neq 0 \Rightarrow \mathrm{WLP}(s_1, Q)) \wedge (e = 0 \Rightarrow \mathrm{WLP}(s_2, Q)) \\
\mathrm{WLP}(\texttt{while}\ e\ \texttt{invariant}\ I\ \texttt{do}\ s, Q) &= I \wedge \\
&\quad \forall x_1, \ldots, x_k, \\
&\quad (((e \neq 0 \wedge I) \Rightarrow \mathrm{WLP}(s, I)) \wedge ((e = 0 \wedge I) \Rightarrow Q))[w_i \leftarrow x_i] \\
&\quad \text{where } w_1, \ldots, w_k \text{ is the set of assigned variables in} \\
&\quad \text{statement } s \text{ and } x_1, \ldots, x_k \text{ are fresh logic variables.}
\end{aligned}$$

Example 1.3.1

$$\mathrm{WLP}(x := x + y, x = 2y) \equiv x + y = 2y$$
$$\mathrm{WLP}(\texttt{while } y > 0 \texttt{ invariant } even(y) \texttt{ do } y := y - 2, even(y)) \equiv$$
$$even(y) \wedge \forall x, ((x > 0 \wedge even(x)) \Rightarrow even(x - 2)) \wedge ((x \leq 0 \wedge even(x)) \Rightarrow even(x))$$

Theorem 1.3.2 (Soundness) *For all statement s and formula Q, $\{\mathrm{WLP}(s, Q)\}s\{Q\}$ is valid for partial correctness.*

Proof. We prove this theorem by directly considering the definition of triples, in terms of operational semantics. It would also be possible to prove the validity of the triple using Hoare logic rules, but that would need some auxiliary results. The proof is performed by induction on the structure of statement s. We detail the proof only for the case of the while loop; the other cases are straightforward.

A preliminary remark is that for any formula ϕ and any state Σ, the interpretation of the formula $\forall x_1, \ldots, x_k, \phi[w_i \leftarrow x_i]$ in Σ does not depend on the values of the variables w_i, and thus if $[\![\forall x_1, \ldots, x_k, \phi[w_i \leftarrow x_i]]\!]_\Sigma$ holds then $[\![\forall x_1, \ldots, x_k, \phi[w_i \leftarrow x_i]]\!]_{\Sigma'}$ also holds for any state Σ' that differs from Σ only for the values of the variables w_i.

Let us assume a state Σ such that $[\![\mathrm{WLP}(s, Q)]\!]_\Sigma$ holds, with $s = \texttt{while } e \texttt{ invariant } I \texttt{ do } b$, and s executes on Σ and terminates: $\Sigma, s \rightsquigarrow^* \Sigma', \texttt{skip}$ to a state Σ'. We want to show that Q holds in Σ'. As for soundness of the Hoare logic rule for while, we proceed by induction on the length of this execution.

The first case is when $[\![e]\!]_\Sigma = 0$: the loop ends immediately and $\Sigma' = \Sigma$. From the definition of $\mathrm{WLP}(s, Q)$ when s is a while loop, we know that both $[\![I]\!]_\Sigma$ and $[\![\forall x_1, \ldots, x_k, ((e = 0 \wedge I) \Rightarrow Q)[w_i \leftarrow x_i]]\!]_\Sigma$ hold. If we simply instantiate the variables of this second part by the values of each w_i in state Σ, we get directly $[\![(e = 0 \wedge I) \Rightarrow Q]\!]_\Sigma$. Then from $[\![e]\!]_\Sigma = 0$ and $[\![I]\!]_\Sigma$ we get $[\![Q]\!]_\Sigma$.

The second case is when $[\![e]\!]_\Sigma \neq 0$. We thus have $\Sigma, s \rightsquigarrow \Sigma, b; s \rightsquigarrow^* \Sigma'', s \rightsquigarrow^* \Sigma', \texttt{skip}$. Since $[\![\mathrm{WLP}(s, Q)]\!]_\Sigma$ holds, we have $[\![\forall x_1, \ldots, x_k, ((e \neq 0 \wedge I) \Rightarrow \mathrm{WLP}(b, I))[w_i \leftarrow x_i]]\!]_\Sigma$. If we instantiate each x_i by the value of each w_i in state Σ, we get that $[\![(e \neq 0 \wedge I) \Rightarrow \mathrm{WLP}(b, I)]\!]_\Sigma$ holds, and thus $[\![\mathrm{WLP}(b, I)]\!]_\Sigma$ holds. By our structural induction, we know that the triple $\{\mathrm{WLP}(b, I)\}b\{I\}$ is valid, hence $[\![I]\!]_{\Sigma''}$ holds. The state Σ'' differs from Σ only for the values of the variables w_i, and thus by our preliminary remarks we know that $[\![\forall x_1, \ldots, x_k, (((e \neq 0 \wedge I) \Rightarrow \mathrm{WLP}(s, I)) \wedge ((e = 0 \wedge I) \Rightarrow Q))[w_i \leftarrow x_i]]\!]_{\Sigma''}$ holds, and thus $[\![\mathrm{WLP}(s, Q)]\!]_{\Sigma''}$ holds. By our induction on the length of the derivation, we get that $[\![Q]\!]_{\Sigma'}$ holds.

As a consequence, for proving that a triple $\{P\}s\{Q\}$ is valid, it suffices to prove the formula $\models P \Rightarrow \mathrm{WLP}(s, Q)$. We justify that WLP is the *weakest* precondition by the following property.

Theorem 1.3.3 (Weakest precondition property) *For any triple $\{P\}s\{Q\}$ that is derivable using our (modified) rules, we have $\models P \Rightarrow \mathrm{WLP}(s, Q)$.*

Proof. The proof is by induction on the structure of the statement s.

Let us consider the sequence and suppose that $\{P\}s_1; s_2\{Q\}$ is derivable. According to Lemma 1.2.5 and the available Hoare rules for the language constructs, there is a derivation tree that ends by the sequence rule followed by the frame rule and finally the consequence rule, which we represent by the following compressed inference rule where variables assigned in either s_1 or s_2 do not occur in R:

$$\frac{\dfrac{\{P'\}s_1\{T\} \quad \{T\}s_2\{Q'\}}{\{P'\}s_1; s_2\{Q'\}} \qquad \models P \Rightarrow (P' \wedge R) \quad \models (Q' \wedge R) \Rightarrow Q}{\{P\}s_1; s_2\{Q\}}$$

First, notice that, since $\{T\}s_2\{Q'\}$ is derivable, $\{T \wedge R\}s_2\{Q\}$ is derivable too by using the frame rule then weakening the postcondition by the consequence rule. So $\models (T \wedge R) \Rightarrow \mathrm{WLP}(s_2, Q)$ holds by

induction. As a consequence, the triple $\{P' \wedge R\} s_1 \{\mathrm{WLP}(s_2, Q)\}$ is derivable by using the frame rule on $\{P'\} s_1 \{T\}$ and then weakening the resulting postcondition. So $\models (P' \wedge R) \Rightarrow \mathrm{WLP}(s_1, \mathrm{WLP}(s_2, Q))$ holds by induction. Therefore, $\models P \Rightarrow \mathrm{WLP}(s_1, \mathrm{WLP}(s_2, Q))$ holds, which concludes the proof for the sequence.

The proofs for the conditional and for the assignment are similar.

Let us consider the case of the loop now. Let us suppose that $\{P\}\texttt{while}\ e\ \texttt{invariant}\ I\ \texttt{do}\ s\{Q\}$ is derivable, which means we have the following pseudo-inference rule:

$$\frac{\{I \wedge e \neq 0\} s \{I\} \qquad \models P \Rightarrow I \wedge R \qquad \models I \wedge e = 0 \wedge R \Rightarrow Q}{\{P\}\texttt{while}\ e\ \texttt{invariant}\ I\ \texttt{do}\ s\{Q\}}$$

with R a formula where no variables assigned in s occur.

By induction, $\models (I \wedge e \neq 0) \Rightarrow \mathrm{WLP}(s, I)$ holds. By definition of $\models$, this means that $\forall \Sigma,\ [\![(I \wedge e \neq 0) \Rightarrow \mathrm{WLP}(s, I)]\!]_\Sigma$ holds. There are finitely many assigned variables in s, so the quantification on these variables can be extracted from Σ: $\forall \Sigma,\ [\![\forall x_1, \ldots, x_k,\ ((I \wedge e \neq 0) \Rightarrow \mathrm{WLP}(s, I))[w_i \leftarrow x_i]]\!]_\Sigma$. Similarly, the quantification on these variables can be made explicit in $\models R \Rightarrow ((I \wedge e = 0) \Rightarrow Q)$. But since these variables do not occur in R, the quantifiers can be moved deeper: $\forall \Sigma,\ [\![R \Rightarrow \forall x_1, \ldots, x_k,\ ((I \wedge e = 0) \Rightarrow Q)[w_i \leftarrow x_i]]\!]_\Sigma$. As a consequence, $[\![(I \wedge R) \Rightarrow \mathrm{WLP}(\texttt{while}\ e\ \texttt{invariant}\ I\ \texttt{do}\ s, Q)]\!]_\Sigma$ holds for any Σ. This concludes the proof for the loop, since $\models P \Rightarrow I \wedge R$.

As a consequence, given a triple $\{P\}s\{Q\}$ that we want to prove valid, rather than exhibiting a derivation tree, we can *without loss of generality* look for a proof of the formula $\models P \Rightarrow \mathrm{WLP}(s, Q)$.

1.4 Proving Termination: Total Correctness

The techniques introduced before to prove a specification (pre- and post-condition) of a piece of code do not bring any useful information in case the program is not terminating.

To alleviate this problem, we can modified the meaning of the validity of a Hoare triple by requiring termination, this is the so-called *total correctness*

Definition 1.4.1 (Total correctness of a program) *A Hoare triple $\{P\}s\{Q\}$ is said* valid *(for total correctness) if for any state Σ such that $[\![P]\!]_\Sigma$ holds, there exists Σ' such that $\Sigma, s \rightsquigarrow^* \Sigma', \texttt{skip}$ and $[\![Q]\!]_{\Sigma'}$ holds. In other words, if s is executed in a state satisfying its precondition, then* it terminates *and the resulting state satisfies its post-condition.*

Note that the language is deterministic, so there is one and only one such Σ' if the program terminates.

1.4.1 Hoare Rules for Total Correctness

Most of our Hoare logic rules remain unchanged when dealing with total correctness. The one that needs some change is naturally the rule for the while loop:

$$\frac{\{I \wedge e \neq 0 \wedge v = \xi\} s \{I \wedge v \prec \xi\} \quad wf(\prec)}{\{I\}\texttt{while}\ e\ \texttt{do}\ s\{I \wedge e = 0\}}$$

with v being an expression and ξ a fresh logic variable. v is called the *variant* of the loop. $wf(\prec)$ means that $\prec$ is a *well-founded relation*, i.e. there is no infinite sequence $\xi_1 \succ \xi_2 \succ \xi_3 \succ \cdots$.

Beware that on our only data type of unbounded integers, the usual relation $<$ is not well-founded. To turn it into a well-founded relation, we need to ensure that we only compare numbers greater than some bound. A standard well-founded relation for loop termination is:

$$x \prec y \quad = \quad x < y \wedge 0 \leq y$$

Example 1.4.2 *A suitable variant for ISQRT is $n - sum$ with the above relation. Notice that this variant would not be suitable if the well-founded relation in use was*

$$x \prec y \quad = \quad 0 \leq x < y$$

because $n - sum$ becomes negative at the last iteration of the loop.

1.4.2 Loops Annotated with Variants

In order to prove termination using a weakest precondition calculus, we augment the syntax of our while-loop construct with an explicit variant.

$$s \quad ::= \quad \ldots \mid \texttt{while}\ e\ \texttt{invariant}\ I\ \texttt{variant}\ v, \prec\ \texttt{do}\ s$$

The Hoare logic rule for total correctness is modified accordingly:

$$\frac{\{I \wedge e \neq 0 \wedge v = \xi\} s \{I \wedge v \prec \xi\} \quad wf(\prec)}{\{I\}\texttt{while}\ e\ \texttt{invariant}\ I\ \texttt{variant}\ v, \prec\ \texttt{do}\ s\{I \wedge e = 0\}}$$

1.4.3 Weakest (Strict) Preconditions Computation

We define a function $\mathrm{WP}(s, Q)$ where s is a statement and Q a formula, using the structurally recursive equations.

$$\begin{aligned}
\mathrm{WP}(x := e, Q) &= Q[x \leftarrow e] \\
\mathrm{WP}(s_1; s_2, Q) &= \mathrm{WP}(s_1, \mathrm{WP}(s_2, Q)) \\
\mathrm{WP}(\texttt{if}\ e\ \texttt{then}\ s_1\ \texttt{else}\ s_2, Q) &= (e \neq 0 \Rightarrow \mathrm{WP}(s_1, Q)) \wedge (e = 0 \Rightarrow \mathrm{WP}(s_2, Q)) \\
\mathrm{WP}\left(\begin{array}{l}\texttt{while}\ e\ \texttt{invariant}\ I \\ \texttt{variant}\ v, \prec\ \texttt{do}\ s\end{array}, Q\right) &= I \wedge \\
&\quad \forall x_1, \ldots, x_k, \xi, \\
&\quad\quad (((e \neq 0 \wedge I \wedge \xi = v) \Rightarrow \mathrm{WP}(s, I \wedge v \prec \xi)) \wedge \\
&\quad\quad\quad ((e = 0 \wedge I) \Rightarrow Q))[w_i \leftarrow x_i] \\
&\quad \text{where } w_1, \ldots, w_k \text{ is the set of assigned variables in} \\
&\quad \text{statement } s \text{ and } x_1, \ldots, x_k, \xi \text{ are fresh logic variables.}
\end{aligned}$$

Theorem 1.4.3 (Soundness) *For all statement s and formula Q, $\{\mathrm{WP}(s, Q)\}s\{Q\}$ is valid for total correctness.*

Theorem 1.4.4 (Weakest precondition property) *For any triple $\{P\}s\{Q\}$ that is derivable using our (modified) rules, we have $\models P \Rightarrow \mathrm{WP}(s, Q)$.*

As a consequence, for proving that a triple $\{P\}s\{Q\}$ is valid (for total correctness), we can *without loss of generality* prove the formula $\models P \Rightarrow \mathrm{WP}(s, Q)$.

1.5 Exercises

Exercise 1.5.1 *Consider the ISQRT program of Example 1.1.1.*

- *Use the Hoare rules for partial correctness to derive the triple for ISQRT from Example 1.2.2.*
- *Prove the validity of the same triple via WLP.*
- *Find a suitable variant for the loop in ISQRT and prove the total correctness of ISQRT using WP.*

Exercise 1.5.2 *Consider the following (inefficient) program for computing the sum $x + y$.*

```
while y > 0 do
   x := x + 1; y := y - 1
```

- *Propose a post-condition stating that the final value of x is the sum of the original values of x and y.*
- *Add a variant and an invariant to the loop.*
- *Prove the program.*

Exercise 1.5.3 *The following program is one of the original examples of Floyd [4].*

```
q := 0; r := x;
while r >= y do
   r := r - y; q := q + 1
```

- *Propose a formal pre-condition to express that x is assumed non-negative, y is assumed positive, and a formal post-condition expressing that q and r are respectively the quotient and the remainder of the Euclidean division of x by y.*
- *Find appropriate loop invariant and variant and prove the total correctness of the program using first Hoare logic rules, second the* WP *calculus.*

Exercise 1.5.4 *Let us assume given in the underlying logic the functions div2(x) and mod2(x) which respectively return the division of x by 2 and its remainder. The following program is supposed to compute, in variable r, the power x^n.*

```
r := 1; p := x; e := n;
while e > 0 do
  if mod2(e) <> 0 then r := r * p;
  p := p * p;
  e := div2(e);
```

- *Assuming that the power function exists in the logic, specify appropriate pre- and post-conditions for this program.*
- *Find appropriate loop invariant and variant, and prove the program using respectively Hoare rules and WP calculus.*

Exercise 1.5.5 *The Fibonacci sequence is defined recursively by $fib(0) = 0$, $fib(1) = 1$ and $fib(n+2) = fib(n+1) + fib(n)$. The following program is supposed to compute fib in linear time, the result being stored in y.*

```
y := 0; x := 1; i := 0;
while i < n do
   aux := y; y := x; x := x + aux; i := i + 1
```

- *Assuming fib exists in the logic, specify appropriate pre- and post-conditions.*
- *Prove the program.*

Exercise 1.5.6 *(Hoare rules and WP calculi for variants of "for" loops)*

- *Propose Hoare deduction rules for the C-style "for" loop*

  for(init; cond; incr) body

  with respect to partial correctness.

- *Same question for the Pascal or Caml-style "for" loop*

  for v=e1 to e2 do body

  Notice that v is not a mutable variable, and e1, e2 must be evaluated only once. Be careful about the case where e1 > e2.

- *If total correctness is considered, what should be modified in the rules above?*
- *Propose WP computation rules for the two cases of "for" loops, including termination checking.*

Bibliography

[1] S. A. Cook. Soundness and completeness of an axiom system for program verification. *SIAM Journal on Computing*, 7(1):70–90, 1978. doi:10.1137/0207005.

[2] P. Cousot. Methods and logics for proving programs. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 841–993. North-Holland, 1990.

[3] E. W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Commun. ACM*, 18:453–457, August 1975. ISSN 0001-0782. doi:10.1145/360933.360975.

[4] R. W. Floyd. Assigning meanings to programs. In J. T. Schwartz, editor, *Mathematical Aspects of Computer Science*, volume 19 of *Proceedings of Symposia in Applied Mathematics*, pages 19–32, Providence, Rhode Island, 1967. American Mathematical Society.

[5] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12 (10):576–580 and 583, Oct. 1969.

[6] G. D. Plotkin. The origins of structural operational semantics. *Journal of Logic and Algebraic Programming*, 60–61:3–15, 2004. doi:10.1016/j.jlap.2004.03.009.