

Programming Language Concepts and Semantics Part I

Lecture 8 Axiomatic Semantics

Lecture IIX introduces *axiomatic semantics*, a logic-based semantic formalism that works with inference rules on pre and post conditions of statements and program fragments.

Lecture IIX follows Winskel, Chapter 6 and 7.

Axiomatic Semantics

In *Axiomatic Semantics* we specify *conditions* and *invariants* that hold during the execution of a program.

Axiomatic Semantics

- uses pre and postconditions to specify properties of programs
- uses first order logic to reason about the semantics of program fragments

First approaches to this kind of reasoning (on flowcharts) were described by R.W. Floyd.

The foundation of modern axiomatic semantics were laid in C.A.R. Hoare. *An axiomatic basis for computer programming*. Communications of the ACM 12(10):576-585, October 1969.

Assertions

The basic ingredient of Hoare Logic are assertions, i.e. logical statements about execution states of a program.

Recall the earlier example

$$\{Y=a\} Y := Y + 1 \{Y=a + 1\} \quad \{Y \neq 0, Y=b\} X := 1 / Y \{X=1/b\}$$
$$\begin{aligned} & \{Y=a, a > 0\} \\ & Y:=Y+1 ; \\ & \{Y > 0, Y=a + 1\} \\ & X := 1 / Y ; \\ & \{X=\frac{1}{a+1}\} \end{aligned}$$

An assertion

$$\{A\}c\{B\}$$

means that if the logical condition A holds before the execution of statement c then B will hold after c has been executed.

A similar style of assertions also forms the basis of programming by contracts (e.g. in the Eiffel language) and you may know very simple forms of assertions from debugging languages like Java. However, in debuggers assertions are only used for run-time checking, whereas we want to be able to abstractly derive program properties without executing a program.

Invariants

Let us consider the analysis of repeat statements, such as loops. Obviously it is more tricky to use assertions $\{A\}C\{B\}$ in the above style to describe the changes between two iterations of a loop.

```
while c do
  {A}
  ... statements
  {B}
end
```

The conditions A and B that we are “passing” in each iteration are the same, yet we want to describe what is changing.

This can be done formulating the execution conditions as *invariants*.

```
S := 0; N := 1;
{S = 0 ∧ N = 1}
while N ≤ 100 do
  (*) {S = ∑i=1...N-1 i}
  S := S + N;
  {S = ∑i=1...N i}
  N := N + 1;
  (*) {S = ∑i=1...N-1 i}
end
{S = ∑1 ≤ i ≤ 100 i ∧ N = 101}
```

The assertion (*) is an *invariant*: it holds at the beginning and at the end of each iteration (and also upon entry into and exit out of the loop).

Partial and Total Correctness

It is important to note that

$$\{A\}c\{B\}$$

for B to be valid requires

- validity of A
- *termination* of c .

This means (informally) that

$$A \wedge \textit{terminates}(c) \Rightarrow B$$

Which leads to non-intuitive correct assertions, such as

$$\{\mathbf{true}\} \mathbf{while\ true\ do\ skip}\ \{\mathbf{false}\}$$

If c does not terminate, we know nothing about the validity of B .

This is called *partial correctness*.

In contrast to this *total correctness* asserts that c will always terminate if A holds and that subsequently B will hold. This is often written as

$$[A]c[B]$$

Language of Assertions

We need to define what is admissible as an assertion.

Assertions are a subset of predicate logic expressions. They contain

- **Aexp**: arithmetic expressions as in IMP
- **Bexp**: boolean expressions as in IMP
- quantification

To be able to use quantification, we need to introduce quantifiers \forall, \exists and *assertion variables* that are only used in the assertions themselves (i.e. are not variables of the program). In our case these will be integer variables $\mathbf{aVar} = \{i, j, k, \dots\}$.

We extend the arithmetic expressions

$$a ::= n \mid X \mid i \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1$$

with $n \in \mathbf{N}, X \in \mathbf{Var}, i \in \mathbf{aVar}$.

We can then define the language of assertions **Assn** as the boolean expressions over these arithmetic expressions:

$$\begin{aligned} \mathit{assn} ::= & \mathbf{true} \mid \mathbf{false} \mid \\ & a_0 = a_1 \mid a_0 \leq a_1 \mid \\ & \mathit{assn}_0 \wedge \mathit{assn}_1 \mid \mathit{assn}_0 \vee \mathit{assn}_1 \mid \neg \mathit{assn} \mid \mathit{assn}_0 \Rightarrow \mathit{assn}_1 \mid \\ & \forall i. \mathit{assn} \mid \exists i. \mathit{assn} \end{aligned}$$

Substitutions

Similarly to substitutions on states we will need to define substitutions on assertions. However, here we do not substitute a value for a program variable, but instead we substitute assertion variables with values (or more generally, with expressions).

$$\begin{aligned}n[a/i] &= n \\X[a/i] &= X \\j[a/i] &= j \text{ where } i \neq j \\i[a/i] &= a \\(a_0 + a_1)[a/i] &= a_0[a/i] + a_1[a/i] \\(a_0 - a_1)[a/i] &= a_0[a/i] - a_1[a/i] \\(a_0 * a_1)[a/i] &= a_0[a/i] * a_1[a/i] **true**[a/i] &= true **false**[a/i] &= false \\(a_0 = a_1)[a/i] &= (a_0[a/i] = a_1[a/i]) \\&\dots \text{ and likewise for all other boolean operators} \\(\forall j. a)[a/i] &= (\forall j. (a[a/i])) \text{ where } i \neq j \\(\forall i. a)[a/i] &= (\forall i. a)\end{aligned}$$

Note: We can only substitute free variables in an expression and assume that all bound variables in the scope of quantifiers have been appropriately renamed to unique variables before the substitution happens. For example

$$\begin{array}{lll}(i = k \wedge \exists i. k \leq i)[a/i] & & (i = k \wedge \exists i. k \leq i)[a/k] \\ \text{results in} & \text{but} & \text{results in} \\ (a = k \wedge \exists i_0. k \leq i_0) & & (i = a \wedge \exists i_0. a \leq i_0)\end{array}$$

Coupling Assertions to States

Let us write

$$\sigma \models A$$

for assertion A holds in state σ .

Recall from denotational semantics that a command denotes a partial function from states to states:

$$\mathcal{C} : \Sigma \rightarrow \Sigma$$

We modify the definition of \mathcal{C} to a total function

$$\mathcal{C}[[c]]\sigma = \perp \text{ where } \mathcal{C}[[c]]\sigma \text{ is } \textit{undefined}$$

\perp denotes an undefined state. In such a state any assertion could hold. So we define

$$\forall A. \perp \models A$$

We can now give a preliminary definition of what the assertion $\{A\}c\{B\}$ means:

$$\forall \sigma \in \Sigma. \sigma \models A \Rightarrow \mathcal{C}[[c]]\sigma \models B$$

Extending Expression Semantics

We can now define the meaning of assertions. This will involve semantic functions as we have used them in denotational semantics.

For expressions, we clearly need to extend the previous definition of the semantic function $\mathcal{A}[[a]]$ as we now have to take assertion variables into account. This is done in such a way that the new semantic function \mathcal{AV} for assertions is consistent with $\mathcal{A}[[a]]$ if a does not involve any assertion variables.

We use an *Interpretation* I that maps assertion variables to integer values

$$I : \mathbf{aVar} \rightarrow \mathbf{N}$$

and write $\mathcal{AV}[[a]]I\sigma$ to denote the value of an assertion expression a under interpretation I in state σ .

$$\begin{aligned}\mathcal{AV}[[n]]I\sigma &= n \\ \mathcal{AV}[[X]]I\sigma &= \sigma(X) \\ \mathcal{AV}[[i]]I\sigma &= I(i) \\ \mathcal{AV}[[a_0 + a_1]]I\sigma &= \mathcal{AV}[[a_0]]I\sigma + \mathcal{AV}[[a_1]]I\sigma \\ \mathcal{AV}[[a_0 - a_1]]I\sigma &= \mathcal{AV}[[a_0]]I\sigma - \mathcal{AV}[[a_1]]I\sigma \\ \mathcal{AV}[[a_0 * a_1]]I\sigma &= \mathcal{AV}[[a_0]]I\sigma * \mathcal{AV}[[a_1]]I\sigma\end{aligned}$$

Substitutions on Interpretations

We extend the notion of substitutions so that they are applicable to the interpretation function on assertion variables:

$$I[n/i](j) = \begin{cases} n & \text{if } j \equiv i \\ I(j) & \text{otherwise} \end{cases}$$

Semantics of Assertions

The meaning of an assertion is specified by defining in which states σ an assertion A holds under a given interpretation I . We write this as

$$\sigma \models^I A$$

and define by structural induction using the extended state set $\Sigma_{\perp} = \Sigma \cup \{\perp\}$.

$$\forall \sigma \in \Sigma_{\perp}$$

$$\begin{aligned}
 \sigma &\models^I \mathbf{true} \\
 \sigma &\models^I (a_0 = a_1) \text{ if } \mathcal{AV}[[a_0]]I\sigma = \mathcal{AV}[[a_1]]I\sigma \\
 \sigma &\models^I (a_0 \leq a_1) \text{ if } \mathcal{AV}[[a_0]]I\sigma \leq \mathcal{AV}[[a_1]]I\sigma \\
 \sigma &\models^I (A_0 \wedge A_1) \text{ if } \sigma \models^I A_0 \wedge_T \sigma \models^I A_1 \\
 \sigma &\models^I (A_0 \vee A_1) \text{ if } \sigma \models^I A_0 \vee_T \sigma \models^I A_1 \\
 \sigma &\models^I (\neg A) \text{ if } \sigma \not\models^I A \\
 \sigma &\models^I (A \Rightarrow B) \text{ if } \sigma \not\models^I A \vee_T \sigma \models^I B \\
 \sigma &\models^I (\forall i.A) \text{ if } \forall n \in \mathbf{N}. \sigma \models^{I[n/i]} A \\
 \sigma &\models^I (\exists i.A) \text{ if } \exists n \in \mathbf{N}. \sigma \models^{I[n/i]} A \\
 \perp &\models^I A
 \end{aligned}$$

Note

- the negation of an assertion A holds if we cannot derive a contradiction $\neg A$
- Any assertion holds in the undefined state \perp .

Partial Correctness Assertions

A program fragment with pre and post conditions is a *partial correctness assertion*.

We need extend our definition of the meaning of an assertion to partial correctness assertions:

If a precondition holds in a state σ under interpretation I and we execute c in σ then B will hold in the follow-up state $\mathcal{C}[[c]]\sigma$.

$$\forall \sigma \in \Sigma. \sigma \models^I \{A\}c\{B\} \text{ iff } \sigma \models^I A \Rightarrow \mathcal{C}[[c]]\sigma \models^I B$$

If we state pre and postconditions of a program fragment we are in general, however, interested in conditions that are valid independently of a particular state or interpretation.

Only this allows us to combine program fragments with assertions in a modular (i.e. independent) fashion.

Validity

Validity captures the independence of a partial correctness assertion from state and interpretation: An assertion $\{A\}c\{B\}$ is *valid*, written $\models \{A\}c\{B\}$ if it holds independently of state and interpretation, i.e.

$$\forall I. \forall \sigma. \sigma \models^I \{A\}c\{B\}$$

- Consider the partial correctness assertion

$$\{0 \leq X \wedge X \leq 1\}Y := 1/X; \{1 \leq Y\}$$

We do not want this to depend on the state (i.e. the value of X).

- Consider the partial correctness assertion

$$\{i \leq X\}X := X + 1; \{i \leq X\}$$

We do not want this to depend on the interpretation of i .

Proof Rules for IMP

We can now give proof rules for IMP. These are called *Hoare Rules*. In conjunction with standard predicate calculus they form the basis of abstract logical reason about a program execution by allowing us to prove partial correctness assertions. We define by structural induction:

- Empty Statements:

$$\{A\}\mathbf{skip}\{A\}$$

- Assignments:

$$\{B[a/X]\}X := a\{B\}$$

- Sequencing:

$$\frac{\{A\}c_0\{C\} \quad \{C\}c_1\{B\}}{\{A\}c_0; c_1\{B\}}$$

- Conditionals:

$$\frac{\{A \wedge b\}c_0\{B\} \quad \{A \wedge \neg b\}c_1\{B\}}{\{A\}\mathbf{if } b \mathbf{ then } c_0 \mathbf{ else } c_1\{B\}}$$

- Iterations:

$$\frac{\{A \wedge b\}c\{A\}}{\{A\}\mathbf{while } b \mathbf{ do } c\{A \wedge \neg b\}}$$

- Consequence:

$$\frac{\models (A \Rightarrow A') \quad \{A'\}c\{B'\} \quad \models (B' \Rightarrow B)}{\{A\}c\{B\}}$$

Interpretation of Hoare Rules

Some of the above Hoare rules are straight forward (empty statements, sequencing). The remaining rules can be interpreted as follows:

- Assignments:

$$\{B[a/X]\} X := a \{B\}$$

To prove that B is valid after the assignment, we need to show that it would hold without the assignment under the additional assumption that X has the value a .

- Conditionals:

$$\frac{\{A \wedge b\}c_0\{B\} \quad \{A \wedge \neg b\}c_1\{B\}}{\{A\}\mathbf{if } b \mathbf{ then } c_0 \mathbf{ else } c_1\{B\}}$$

To prove the validity of $\{A\}\mathbf{if } b \mathbf{ then } c_0 \mathbf{ else } c_1\{B\}$ we need to show that B follows for both possible types of execution.

- Iterations:

$$\frac{\{A \wedge b\}c\{A\}}{\{A\}\mathbf{while } b \mathbf{ do } c\{A \wedge \neg b\}}$$

$\neg b$ must hold when the loop terminates. To prove that A still holds we need to show that c does not change A , i.e. that A is an *invariant*.

- Consequence:

$$\frac{\models (A \Rightarrow A') \quad \{A'\}c\{B'\} \quad \models (B' \Rightarrow B)}{\{A\}c\{B\}}$$

Note that the rule of consequence is not tied to a particular syntactic form by structural induction. Its function is to link the Hoare rules to standard logical inference.

Soundness and Completeness

Whenever we establish a system of proof rules, we are interested in two properties: soundness and completeness.

- Soundness establishes that only valid conclusions are derived from valid assumptions, i.e. the proof system does not allow us to prove something that “is not true”.
- Completeness establishes that the proof systems allows us to prove everything that is the fact (in accordance with some other definition of *what is the fact*).

Obviously soundness is the more fundamental property. A proof system that is not sound is useless, because we can never rely on the fact that our conclusions are valid.

A proof system that is not complete is still useful, even though it is sometimes not *strong* enough to allow us to prove some theorem.

We proceed to show the soundness of the Hoare rules for IMP. This can be done by showing that every rule is sound in itself. It follows that the whole system of rules is sound.

We write $\vdash\{A\}c\{B\}$ to denote that $\{A\}c\{B\}$ can be proven using the Hoare rules.

Definition: (*Soundness*) $\vdash\{A\}c\{B\} . \Rightarrow . \models \{A\}c\{B\}$.

Soundness of the Assignment Rule

Lemma: $\sigma \models^I B[a/X]$ iff $\sigma[\mathcal{A}[[a]]\sigma/X] \models^I B$

This essentially states that instead of substituting a variable X by an expression a in a partial correctness assertion, we can substitute the variable by the value of the expression a in the environment from which we are trying to prove B . This can be proven by structural induction on B .

To show soundness of the assignment rule:

$$(1) \vdash \{B[a/X]\}X := a\{B\}$$

$$(2) \sigma \models^I B[a/X] \text{ iff } \sigma[\mathcal{A}[[a]]\sigma/X] \models^I B \quad \text{by the above lemma}$$

$$(3) \mathcal{C}[[X := a]]\sigma = \sigma[\mathcal{A}[[a]]\sigma/X] \quad \text{from the definition of } \mathcal{C}$$

$$(4) \sigma \models^I B[a/X] \Rightarrow \mathcal{C}[[X := a]]\sigma \models^I B \quad \text{by (2) and (3)}$$

$$(5) \models \{B[a/X]\}X := a\{B\} \quad \text{by (4) and def. of } \models^I$$

Soundness of the Sequencing Rule

$$\frac{\{A\}_{c_0}\{C\} \quad \{C\}_{c_1}\{B\}}{\{A\}_{c_0; c_1}\{B\}}$$

From the precondition we have

$$\{A\}_{c_0}\{C\} \wedge \{C\}_{c_1}\{B\}$$

Take some state σ with $\sigma \models^I A$.

- (1) $\mathcal{C}[[c_0]]\sigma \models^I C$ *from def. of \models^I and $\{A\}_{c_0}\{C\}$*
- (2) $\mathcal{C}[[c_1]](\mathcal{C}[[c_0]]\sigma) \models^I B$ *from def. of \models^I , $\{C\}_{c_1}\{B\}$, and (1).*
- (3) $\mathcal{C}[[c_0; c_1]]\sigma \models^I B$ *from $\mathcal{C}[[c_0; c_1]]\sigma = \mathcal{C}[[c_1]](\mathcal{C}[[c_0]]\sigma)$, and (2).*
- (4) $\models \{A\}_{c_0; c_1}\{B\}$ *from def. of \models^I and the above.*

Soundness of the Conditional Rule

$$\frac{\{A \wedge b\}c_0\{B\} \quad \{A \wedge \neg b\}c_1\{B\}}{\{A\}\mathbf{if } b \mathbf{ then } c_0 \mathbf{ else } c_1\{B\}}$$

From the precondition we have

$$\{A \wedge b\}c_0\{B\} \wedge \{A \wedge \neg b\}c_1\{B\}$$

Take some state σ with $\sigma \models^I A$.

- **Case** $\sigma \models^I b$:

We have $\mathcal{C}[\mathbf{if } b \mathbf{ then } c_0 \mathbf{ else } c_1] = \mathcal{C}[[c_0]]$ if $b = \mathbf{true}$.

From the precondition we have $\{A \wedge b\}c_0\{B\}$.

From the definition of validity we obtain $\mathcal{C}[[c_0]]\sigma \models^I B$.

- **Case** $\sigma \models^I \neg b$:

We have $\mathcal{C}[\mathbf{if } b \mathbf{ then } c_0 \mathbf{ else } c_1] = \mathcal{C}[[c_1]]$ if $b = \mathbf{false}$.

From the precondition we have $\{A \wedge \neg b\}c_1\{B\}$.

From the definition of validity we obtain $\mathcal{C}[[c_1]]\sigma \models^I B$.

- In both cases $\mathcal{C}[\mathbf{if } b \mathbf{ then } c_0 \mathbf{ else } c_1]\sigma \models^I B$, we substitute into the definition of validity to obtain $\models \{A\}\mathbf{if } b \mathbf{ then } c_0 \mathbf{ else } c_1\{B\}$.

Soundness of the Consequence Rule

$$\frac{\models (A \Rightarrow A') \quad \{A'\}c\{B'\} \quad \models (B' \Rightarrow B)}{\{A\}c\{B\}}$$

Take some state σ with $\sigma \models^I A$.

From the precondition we have $A \Rightarrow A'$.

Hence $\sigma \models^I A'$.

From the precondition $\{A'\}c\{B'\}$ and the definition of validity

$$\sigma \models^I B'.$$

From the precondition $B' \Rightarrow B$ and hence $\sigma \models^I B$.

Substituting into the definition of validity, we obtain $\{A\}c\{B\}$.

Soundness of the While Rule

$$\frac{\{A \wedge b\}c\{A\}}{\{A\}\mathbf{while\ } b \mathbf{ do\ } c\{A \wedge \neg b\}}$$

As could have been expected, this is the property that is the most complicated to establish. We need to use induction to prove the soundness of while.

Let $w \equiv \mathbf{while\ } b \mathbf{ do\ } c$.

Recall from the fixpoint construction of the loop semantics that

$$\mathcal{C}[[w]] = \bigcup_{n=1.. \infty} \gamma_n$$

where $\gamma_n = \Gamma^n(\emptyset)$ and

$$\begin{aligned} \gamma_0 &= \emptyset \\ \gamma_{n+1} &= \{(\sigma, \sigma') \mid (\sigma, \mathbf{true}) \in \mathcal{B}[[b]] \wedge_T (\sigma, \sigma') \in \gamma_n \circ \mathcal{C}[[c]]\} \\ &\quad \cup \{(\sigma, \sigma) \mid (\sigma, \mathbf{false}) \in \mathcal{B}[[b]]\} \end{aligned}$$

Using ordinary induction on the integers we show the property $P(n)$:

$$\begin{aligned} \forall n \in \{1, \dots, \infty\}. \forall \sigma, \sigma' \in \Sigma . \\ ((\sigma, \sigma') \in \gamma_n \wedge \sigma \models^I A . \Rightarrow . \sigma' \models^I A \wedge \neg b) \end{aligned}$$

Effectively we prove soundness for increasing levels of the fixpoint construction, which—in a sense—corresponds to showing it for increasing lengths of loop executions.

Soundness of the While Rule (cont.)

Take some state σ with $\sigma \models^I A$.

Assume A is an invariant

$$\models \{A \wedge b\}c\{A\}$$

- *Base Case:* $n = 0$: $\gamma_0 = \emptyset$, so the hypothesis $P(0)$ trivially holds.
- *Induction:* We assume $P(n)$ and show $P(n + 1)$. We distinguish two cases:
 - $(\sigma, \mathbf{true}) \in \mathcal{B}[[b]]$: We have $\mathcal{B}[[b]] = \mathbf{true}$ and thus $\sigma \models^I b$. As $\sigma \models^I A$ it follows that $\sigma \models^I A \wedge b$. From the definition of \mathcal{C} we know there is some state σ'' such that $\mathcal{C}[[c]]\sigma = \sigma''$. From this, $\sigma \models^I A$ and the invariant condition we obtain $\sigma'' \models^I A$. From the definition of γ_{n+1} we see that for this state σ'' we have $(\sigma'', \sigma') \in \gamma_n$. We assume the induction hypothesis $P(n)$. As $(\sigma'', \sigma') \in \gamma_n \wedge \sigma'' \models^I A$ it follows that $\sigma' \models^I A \wedge \neg b$. This proves $P(n + 1)$.
 - $(\sigma, \mathbf{false}) \in \mathcal{B}[[b]]$: We have $\mathcal{B}[[b]] = \mathbf{false}$ and thus $\sigma \models^I \neg b$. As $\sigma \models^I A$ it follows that $\sigma \models^I A \wedge \neg b$. From the definition of \mathcal{C} we know that $\sigma = \sigma'$. Thus $\sigma' \models^I A \wedge \neg b$.

This establishes $P(n)$ for all n . This the rule for while is sound.

Proving Program Properties in Hoare Logic

We want to prove that our implementation of the factorial does indeed compute the factorial

$$n! = \prod_{i=1}^n i$$

Let the program fragment w be:

```
while ( X > 0 ) do
  begin
    Y := X * Y ;
    X := X - 1 ;
  end
```

Then we have to show that

$$\{X = n \wedge n \geq 0 \wedge Y = 1\} w \{Y = n!\}$$

We need to invoke the **while** inference rule

$$\frac{\{A \wedge b\}c\{A\}}{\{A\}\mathbf{while} \ b \ \mathbf{do} \ c\{A \wedge \neg b\}}$$

and choose the invariant A to be:

$$A \equiv Y \times X! = n! \wedge X \geq 0$$

Proving Program Properties in Hoare Logic (cont.)

To show that A is an invariant we choose $b \equiv X > 0$ and prove

$$\{Y \times X! = n! \wedge X > 0\} Y := Y * X; X := X - 1 \{Y \times X! = n! \wedge X \geq 0\}$$

Using the sequencing rule this can be reduced to

$$\{Y \times X! = n! \wedge X > 0\} Y := Y * X; \{C\}$$

and

$$\{C\} X := X - 1 \{Y \times X! = n! \wedge X \geq 0\}$$

We invoke the assignment rule for the *second* part and obtain

$$\{A [(X - 1)/X]\} X := X - 1 \{A\}$$

where $C = A [(X - 1)/X]$ is $Y \times (X - 1)! = n! \wedge (X - 1) \geq 0$.

We again invoke assignment rule on the *first* part and obtain

$$\{C [(Y * X)/Y]\} Y := Y * X \{C\}$$

where $C [(Y * X)/Y]$ is $Y * X \times (X - 1)! = n! \wedge (X - 1) \geq 0$.

Clearly, $\models (A \wedge b \Rightarrow C [(Y * X)/Y])$ and trivially $\models A \Rightarrow A$.

Thus, by the consequence rule

$$\frac{\models (A \Rightarrow A') \quad \{A'\}c\{B'\} \quad \models (B' \Rightarrow B)}{\{A\}c\{B\}}$$

we establish that $\{A\}c\{A\}$, i.e. that A is an invariant.

With $b \equiv X > 0$, using consequence we derive $\{A \wedge b\}c\{A\}$.

From the **while** rule: $\{A \wedge b\} \mathbf{while} \ b \ \mathbf{do} \ c \{A \wedge X \neq 0\}$ and by consequence:

$$\{A\}w\{Y = n! \wedge X = 0\}.$$

□

Logical Incompleteness - Gödel's Theorem

We now ask, what are the limits of using Hoare logic? Of course, we would like to be able to feed a hoare logic to an automated theorem proving system that can make an automated decision about any property of a given program that we can state as a partial correctness assertion. Is this possible?

To answer this question, we must revisit one of the most famous and most important results in twentieth century mathematics.

In 1920 the German mathematician David Hilbert suggested that mathematics should be completely axiomatized and then proven as being consistent. In 1931, The Austrian Mathematician Kurt Gödel showed that this is quite simply not possible. His *second incompleteness theorem* states that *no formal system that is consistent can prove its own consistency*.

Gödel's argument is based on the construction of a paradox. Intuitively, consider a system S allows you to express a property P that expresses " P cannot be proven in S ". If S is complete, it must be able to prove P true or false. However, if S proves P to be true than P is also false (and vice versa). So S is not consistent. Thus no logical system which can support Peano's axioms can be both complete and consistent. This result is an extension of Gödel's First Incompleteness Theorem.

Gödel's First Incompleteness Theorem: (On formally undecidable propositions of principia mathematica and related systems, 1931) *Any consistent formal system that is expressive enough to define Peano Arithmetic (the natural numbers) is incomplete: It*

can express statements that can neither be proven true nor false within this system.

The Essence of Gödel's Proof

Gödel's proof is an instance of a *Diagonalization Argument*, a style of argument first conducted by Cantor to show that the set of real numbers is not countable.

Gödel's first trick was to show that in a sufficiently expressive system all formulas in a theory can be systematically enumerated, i.e. given unique numbers (*Gödel numbers*). Using this, a system that can reason about numbers can reason about formulas. Let $g(f)$ be the Gödel number of formula f .

The second trick is to use a self-referential paradoxical property. Let $P(x)$ say for the statement f with Gödel number x that $f(g(f))$ is not provable, i.e. that statement f applied to its own Gödel number is not provable.

Now let $G = P(g(P))$ and assume that the system is consistent.

If G is provable, then $P(g(P))$ is provable, but this is exactly the property that $P(g(P))$ is unprovable. Thus G cannot be provable.

If $\neg G$ is provable, then $\neg P(g(P))$ is provable, but this is exactly the property that $g(P)$ is *not* the number of a formula such that $f(g(f))$ is unprovable. Thus $P(g(P))$ must be provable and thus G must be provable. But this is not consistent with the assumption that $\neg G$ is provable.

Therefore, in any sufficiently expressive consistent system there are properties that can neither be proven nor disproven. \square

The Halting Problem

An central theorem of Computer Science due to Alonzo Church and Alan Turing is an extension of Gödel's incompleteness result and closely related to this. *A. Church. A note on the Entscheidungsproblem, 1936.*

“Entscheidungsproblem” (decision problem) Theorem:

There is no algorithm that can decide for every given first-order statement whether it is universally true or not.

Church's proof is phrased in terms of the λ -calculus, Turing's proof (somewhat easier to understand) reduces the proof to the Halting Problem for Turing Machines (*A. Turing. On computable numbers, with an application to the Entscheidungsproblem, 1937*).

Halting Problem Theorem: There is no program P that computes a function $p(n, i)$ that decides for a given program n and input i whether or not n applied to i halts.

The proof is closely related to Gödel's proof. Again, the core idea is that Turing Machines can be given unique numbers (Gödel numbers), too.

Halting Problem for Turing Machines (Proof Idea)

Let $TM(n)$ the Turing machine with Gödel number n . We construct a paradoxical Turing machine:

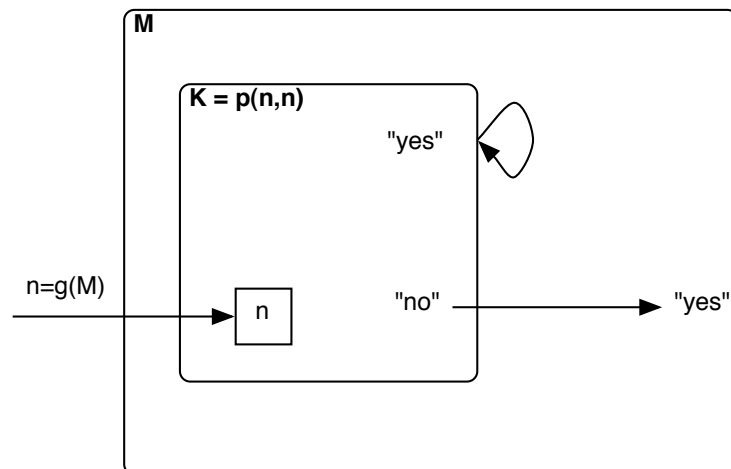
Assume there is a Turing machine K that computes property p from the Entscheidungsproblem and halts accordingly with output “yes” or “no”.

We can now construct a very strange Turing machine M : It takes as input a single number n , the Gödel number of a Turing machine. It first executes K to compute $p(TM(n), n)$. If K stops with “no”, M halts answering “yes”. If K stops with “no”, M goes into an infinite loop.

Now we start M on input $g(M)$, its own Gödel number. If M stops, then K had answered that M applied to $g(M)$ does not halt.

If M on input $g(M)$ does not halt, then K had answered that M applied to $g(M)$ does halt.

In both cases we have a contradiction, so K cannot exist. \square .



Incompleteness of Hoare Logic

As a consequence of the above theorem, Hoare Logic cannot be complete. In particular, we cannot construct a theorem prover that decides for a given assertion whether it is universally true.

The Hoare rules that we have given look very simple, so why is this not possible? The culprit is the *consequence* rule which allows us to include any first order implication in a proof.

$$\frac{\models (A \Rightarrow A') \quad \{A'\}c\{B'\} \quad \models (B' \Rightarrow B)}{\{A\}c\{B\}}$$

Therefore, whenever our assertions are expressive enough to allow us to define the natural numbers, Hoare Logic cannot be complete.

Proof: Assume we could decide any partial correctness assertion. Then we could particularly decide $\{\mathbf{true}\} \mathbf{skip} \{\mathbf{B}\}$.

As $\{\mathbf{true}\} \mathbf{skip} \{\mathbf{B}\} . \Leftrightarrow . \models \mathbf{B}$, this would give us an effective decision procedure for B . □

Relative Completeness

If we still want to be able to meaningfully talk about completeness of Hoare Logic, we must use a *weaker* notion of completeness.

Such a form of completeness was defined by Stephen Cook and is called *relative completeness*.

Relative Completeness of Hoare Logic: If a partial correctness assertion $\{A\}c\{B\}$ is valid then there is a proof in Hoare Logic for it, provided this proof is allowed to use certain assertions in **Assn** as axioms.

This essentially amounts to using an oracle in the proof. What prevents Hoare Logic from being complete is the fact that the assertion language can define the natural numbers and that the consequence rule requires provability for any statement in the assertion language.

If we allow the proof to use valid arithmetic statements “for free”, i.e. as axioms, without having to prove them, then the remaining proof obligations can be fulfilled mechanically using Hoare Logic.

Weakest Preconditions

To prove relative completeness, we need a new concept: the *weakest precondition*.

Consider an assertion $\{A\}c\{B\}$. Given B , we may be interested in the most general assertion $\{W\}$ that guarantees us $\{W\}c\{B\}$. For formal reasons, this must of course also include all states from which c diverges.

The **weakest precondition** of B with respect to c in I is:

$$wp^I[[c, B]] = \{\sigma \in \Sigma_{\perp} \mid \mathcal{C}[[c]]\sigma \models^I B\}$$

There are some important questions about the weakest precondition

- Does the weakest precondition always exist?
- Is the weakest precondition unique?
- Can the weakest precondition always be expressed in **Assn**?
- Can the weakest precondition always be proven in Hoare Logic?

Luckily the answer to all these questions is *yes*.¹

¹The weakest precondition is only unique up to equivalence of assertions.

Expressiveness of the Assertion Language

The weakest precondition always exists and is unique (up to equivalence of assertions).

The proof of *Expressiveness* (the weakest precondition can always be expressed in **Assn**) is rather involved (because of the while loop). We skip the details (see Winskel, Chapter 7, Lemma 7.5).

We give the general flavour only for two simple cases. We define a function $w[[c, B]]$ by structural induction to give us the weakest precondition for assertion B with respect to command c . We then use structural induction to show that w models the weakest precondition correctly, i.e. that

$$\sigma \models^I w[[c, B]] \ . \Leftrightarrow \ . \mathcal{C}[[c]]\sigma \models^I B$$

- **Skip:** $w[[\mathbf{skip}, B]] = B$.

$$\begin{aligned} \sigma \in wp^I[[\mathbf{skip}, B]] & \ . \Leftrightarrow \ . \text{ (by def. of } wp) \\ \mathcal{C}[[\mathbf{skip}]]\sigma \models^I B & \ . \Leftrightarrow \ . \text{ (by def. of } \mathcal{C}) \\ \sigma \models^I B & \ . \Leftrightarrow \ . \text{ (by def. of } w) \\ \sigma \models^I w[[\mathbf{skip}, B]] & \end{aligned}$$

- **Sequencing:** $w[[c_0; c_1, B]] = w[[c_0, w[[c_1, B]]]]$.

$$\begin{aligned} \sigma \in wp^I[[c_0; c_1, B]] & \ . \Leftrightarrow \ . \text{ (by def. of } wp) \\ \mathcal{C}[[c_0; c_1]]\sigma \models^I B & \ . \Leftrightarrow \ . \text{ (by def. of } \mathcal{C}) \\ \mathcal{C}[[c_1]](\mathcal{C}[[c_0]]\sigma) \models^I B & \ . \Leftrightarrow \ . \text{ (by induction)} \\ \mathcal{C}[[c_0]]\sigma \models^I w[[c_1, B]] & \ . \Leftrightarrow \ . \text{ (by induction)} \\ \sigma \models^I w[[c_0, w[[c_1, B]]]] & \ . \Leftrightarrow \ . \text{ (by def. of } w) \\ \sigma \models^I w[[c_0; c_1, B]] & \end{aligned}$$

Hoare Logic is Relative Complete

We skip the details of the proof that the weakest precondition can always be proved in Hoare Logic as it relies on the constructions in the proof of expressiveness (use structural induction on c). For details see Lemma 7.6 in Winskel, Chapter 7.

However, given the four lemmata

- (1) The weakest precondition always exists.
- (2) The weakest precondition is unique.
- (3) The weakest precondition can be expressed in **Assn**.
- (4) The weakest precondition can be proven in Hoare Logic.

we can easily see that

Theorem: Hoare Logic is *relative complete*.

Proof: For $\{A\}c\{B\}$ we can construct a proof using the consequence rule:

$$\frac{\begin{array}{c} (oracle) \\ \overline{A \Rightarrow wp[[c, B]]} \end{array} \quad \frac{\begin{array}{c} \vdots \\ \overline{\{wp[[c, B]]\}c\{B\}} \end{array} \quad \overline{B \Rightarrow B}}{\overline{\{A\}c\{B\}}}$$

As we have an oracle for the assertions in **Assn**, we can prove the left top branch. From (1-3) allow us to state $\{wp[[c, B]]\}c\{B\}$ and from (4) we know that we can complete the middle top branch. \square

Summary

We have discussed axiomatic semantics

- Denotational semantics uses logical pre and post conditions to establish program properties.
- Such conditions take the form of partial correctness assertions $\{A\}c\{B\}$.
- Reasoning is performed with a mixture of standard predicate calculus and special program inference rules, called Hoare rules.
- Reasoning with Hoare Logic is sound.
- Due to general logical incompleteness, there can be no effective proof procedure for Hoare Logic that is complete.
- We have introduced the concept of Relative Completeness and have demonstrated that Hoare Logic is relative complete.

We have also given the axiomatic semantics for IMP and have shown the soundness of its Hoare rules.

Homework

- Prove that the semantics of boolean expressions and the semantics of assertions (limited to boolean expressions) agree independently of the interpretation I

$$\begin{aligned}\mathcal{B}[[b]] &= \mathbf{true} \quad \text{iff} \quad \sigma \models b \\ \mathcal{B}[[b]] &= \mathbf{false} \quad \text{iff} \quad \sigma \not\models b\end{aligned}$$

Use structural induction.