COMS22202: 2014/15

Language Engineering

Dr Oliver Ray
(csxor@Bristol.ac.uk)

Department of Computer Science
University of Bristol

Thursday 3rd March, 2016

Axiomatic Semantics: cf. chap 6

- While the denotational semantics fully defines a program's behaviour, we are usually more interested in certain properties (regarding the program's intended purpose) than we are in others (implementation details)
- E.g., the denotational semantics of **while ¬(x=1) do (y:=y*x; x:=x-1)**, which can be directly characterised as follows, captures each and every effect of the program on all variables in all possible states:

$$\mathcal{S}_{\mathrm{ds}}[\![\, \textbf{while } \neg(x=1) \textbf{ do } (y:=y*x;\ x:=x-1) \,]\!]\ s = \begin{cases} s\,[\,x \mapsto 1\,]\,[\,y \mapsto (s\ y) * (s\ x)!\,] & \textbf{if } s\ x \geq 1 \\ \underline{\textbf{undef}} & \textbf{otherwise} \end{cases}$$

- By contrast, the axiomatic semantics allows to focus on the key property; which might be that, if **y** is initially 1 and **x** is a strictly positive integer **n**, then **y** will become **n**! (and we don't really care about the final value of **x**)
- This can be asserted in the axiomatic semantics by annotating the program with a precondition and a postcondition as follows:

{y=1 & x=n & n>0} while ¬(x=1) do (y:=y*x; x:=x-1) {y=n!}

Preconditions and Postconditions: p176

- The axiomatic semantics represents interesting properties of a program using *preconditions* (which are true in the state prior to executing the program) and *postconditions* (which are true in the state right afterwards)
- Such properties are denoted by *assertions* in the form of triples known as *Hoare triples* (named after the computer scientist Tony Hoare) which are written (either horizontally or vertically) as follows:

  Precondition Program Postcondition

- The axiomatic semantics is a calculus for deriving correct triples of this form. In fact, there are two calculi, of *total* and *partial* correctness, which differ in that the latter proves termination while the former doesn't (p.169)
- Under the partial correctness semantics, assertions are written **{P} S {R}** and mean that, if **P** holds before running **S** and, *if* **S** terminates, then **R** will hold immediately afterwards
- Under the total correctness semantics, assertions are written **{P} S {⇓R}** (or more usually **[P] S [R]**) and mean that, if **P** holds before running **S**, then **S** *will* terminate and **R** will hold immediately afterwards

The Language of Assertions: cf. p176-7

- The precise formalisation of the assertion language for preconditions and postconditions is discussed at length in the book, which compares two approaches called the *extensional* approach and the *intentional* approach
- But both of these approaches are too technical for our purposes (as they assume a prior knowledge of certain concepts - albeit in a more logical context - that this course aims to introduce in this more applied context)
- Therefore we take a simpler and more common sense approach that is closer to Hoare's original formulation and which uses a more conventional notation (like that described at http://en.wikipedia.org/wiki/Hoare_logic)
- We simply assume the assertion language contains standard arithmetic and logical operators and these include all of the arithmetic and Boolean expressions representable in the programming language **While**
- We also assume the assertion language contains a variable symbol for each program variable, in addition so-called logical variables (that we can use to remember the past values of program variables)
- So, while we certainly need to distinguish logical and program variables (p.176) we are *not* concerned with with distinction between extensional and intensional languages (p.177)

Inference System: Axioms and Rules

- An axiomatic semantics consists of a set of *axioms* and a set of *rules* that provide a (sound and complete) method for finding all true assertions
- A rule consists of an assertion a, called the *conclusion*, and set of one or more assertions $a_1 \ldots a_n$ called the *premises* – written as follows:

$$\frac{a_1 \quad \ldots \quad a_n}{a}$$

A rule is understood as meaning that, to show the conclusion a, it is sufficient to show all of the premises $a_1 \ldots a_n$

- An axiom is essentially a rule with no premises (which means that its conclusion a can simply be assumed at any time) and is written

$$a$$

- It is usual to express the individual axioms and rules using *schemata* which contain meta-variables that may be instantiated for particular programs or pre- and post-conditions (possibly subject to restrictions)

Partial Correctness Schemata: cf. p178

$$\{ P \}\ \text{skip}\ \{ P \}$$ **SKIP**

$$\{ P(a) \}\ x := a\ \{ P(x) \}$$ **ASS**

$$\frac{\{ P \}\ S_1\ \{ Q \} \qquad \{ Q \}\ S_2\ \{ R \}}{\{ P \}\ S_1 ; S_2\ \{ R \}}$$ **SEQ**

$$\frac{\{ P \wedge b \}\ S_1\ \{ Q \} \qquad \{P \wedge \neg b \}\ S_2\ \{ Q \}}{\{ P \}\ \text{if}\ \ b\ \text{then}\ \ S_1\ \ \text{else}\ \ S_2\ \{ Q \}}$$ **COND**

$$\frac{\{ P \wedge b \}\ S\ \{ P \}}{\{ P \}\ \text{while}\ b\ \text{do}\ S\ \{P \wedge \neg b \}}$$ **LOOP**

$$\frac{\{ P' \}\ S\ \ \{ Q' \}}{\{ P \}\ S\ \ \{ Q \}} \quad \text{where}\ \ P\ \models P' \quad Q' \models Q$$ **CONS**

Axiom Schema for Skip: cf. p178

{ P } skip { P }

- If a statement is true before running skip then it must be true after
- If a statement is true after running skip then it must have been true before
- **P** is any statement in the assertion language
- For example: {x>1} skip {x>1}

Axiom Schema for Assign: cf. p178

{ P(a) } x := a { P(x) }

- If a statement **P(x)** (that possibly involves the program variable **x**) is true after assigning **x** to **a**, then the statement **P(a)** (where all occurrences of **x** are replaced by **a**) must have been true before
- **P** is any statement in the assertion language, **x** is any program variable, and **a** is any arithmetic expression in **While**
- For example: {x-1>1} x:=x-1 {x>1}
- Note: on this course we are just syntactically replacing every original occurrence of **x** by **a** in **P** (and it doesn't matter if **a** contains **x** or not); this is often denoted **P[x/a]** or **P[x |-> a]** and it provides a simpler but equivalent approach to the one used in the book (which makes a point of embedding the denotational semantics within the axiomatic semantics)

Rule Schema for Sequence: cf. p179

$$\frac{\{\ P\ \}\ S_1\ \{\ Q\ \} \qquad \{\ Q\ \}\ S_2\ \{\ R\ \}}{\{\ P\ \}\ S_1\ ;\ S_2\ \{\ R\ \}}$$

- To prove an assertion about a composition of two program statements it suffices to find an intermediate property (sometimes called a *midcondition*) that is a postcondition of the former and a precondition of the latter

- **P Q R** are any statements in the assertion language and **S_1 S_2** are any program statements in **While**

- For example: $$\frac{\{x=1\}\ x:=x-1\ \{x=0\} \qquad \{x=0\}\ y:=y*x\ \{y=0\}}{\{x=1\}\ x:=x-1\ ;\ y:=y*x\ \{y=0\}}$$

- Note: we can't show the premises directly using the assignment axiom as we would only strictly get **{x-1=0} x:=x-1 {x=0}** and **{y*x=0} y:=y*x {y=0}** (we need the later consequence rule to link them together in a real proof)

Rule Schema for Conditionals: cf. p179

$$\frac{\{ P \wedge b \}\ S_1\ \{ Q \} \qquad \{ P \wedge \neg b \}\ S_2\ \{ Q \}}{\{ P \}\ \textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \{ Q \}}$$

- To prove an assertion about a conditional it suffices to show the property holds whether the condition is satisfied or not

- **P Q** are any statements in the assertion language, **S_1 S_2** are any program statements and **b** is any Boolean expression of **While**

- For example:

$$\frac{\{x=n \,\&\, x\leq 0\}\ x:=x*(-1)\ \{x=|n|\} \qquad \{x=n \,\&\, \neg(x\leq 0)\}\ \text{skip}\ \{x=|n|\}}{\{x=n\}\ \text{if } (x\leq 0) \text{ then } x:=x*(-1) \text{ else skip}\ \{x=|n|\}}$$

- Note: we can't show the premises directly using the assignment and skip axioms as we would only strictly get **$\{x*(-1)=|n|\}$ $x:=x*(-1)$ $\{x=|n|\}$** and **$\{x=|n|\}$ skip $\{x=|n|\}$** (once again we'd need the consequence rule here)

Rule Schema for Loop: cf. p179

$$\frac{\{ P \wedge b \}\ S\ \{ P \}}{\{ P \}\ \textbf{while } b \textbf{ do } S\ \{ P \wedge \neg b \}}$$

- In the rule for loops, the precondition **P** is also known a loop invariant (as showing the premise is in fact equivalent to proving that **P** is maintained on each iteration of the loop)
- On exiting the loop (assuming that it terminates) we know that the invariant must be true along with the negation of the loop condition (because, if the loop condition were true, then the loop wouldn't have finished yet)
- **P** is any statement in the assertion language, **S** is any program statement and **b** is any Boolean expression of **While**
- For example:

$$\frac{\{x\%2=n\%2\ \&\ x\geq0\leftrightarrow n\geq0\ \&\ 2\leq x\}\ \ x:=x-2\ \ \{x\%2=n\%2\ \&\ x\geq0\leftrightarrow n\geq0\}}{\{x\%2=n\%2\ \&\ x\geq0\leftrightarrow n\geq0\}\ \textbf{while } 2\leq x \textbf{ do } x:=x-2\ \{x\%2=n\%2\ \&\ x\geq0\leftrightarrow n\geq0\ \&\ \neg(2\leq x)\}}$$

Rule Schema for Consequence: cf. p179

$$\frac{\{ P' \}\ S\ \{ Q' \}}{\{ P \}\ S\ \{ Q \}} \qquad \text{where } P \models P' \quad Q' \models Q$$

- This rule is frequently necessary to glue help glue the results of the other rules together (and is sometimes presented as two separate rules)
- If we can show **{ P' } S { Q' }** then we can *strengthen* the precondition to **P** and *weaken* the postcondition to **Q**
- **S** is any program statement of **While** and **P P' Q Q'** are any statements in the assertion language such that **P |= P'** and **Q' |= Q**
- In order to use this rule it is necessary to demonstrate (by a proof) that the two requirements **P |= P'** and **Q' |= Q** are both satisfied
- For example: $$\frac{\{x-1=0\}\ x:=x-1\ \{x=0\}}{\{x=1\}\ x:=x-1\ \{x=0\}}$$

as **x=0 |= x=0 (**trivially, by reflexivity of logical entailment)

and **x=1 |= x-1=0** since from **x=1** we have **x-1=1-1** (subtracting 1 from both sides and **1-1=0** (by definition of subtraction) and so **x-1=0**

Axiomatic Proofs

- Proving an assertion in the axiomatic semantics amounts to chaining rules together (in the form of a proof tree) such the premises of one rule in the tree are the conclusions of the rules immediately above it.
- For clarity we will often label each rule with the schema it came from and we will sometimes draw dotted lines above the leaves (which will all be instances of the SKIP and ASS axioms).
- Each application of the consequence rule will be labelled with a number that we can use to identify the associated entailment proofs. For example:

$$\dfrac{\dfrac{\dfrac{}{\{x-1=0\}\ x:=x-1\ \{x=0\}}\ \text{ASS}}{\{x=1\}\ x:=x-1\ \{x=0\}}\ \text{CONS-1} \qquad \dfrac{\dfrac{}{\{y*x=0\}\ y:=y*x\ \{y=0\}}\ \text{ASS}}{\{x=0\}\ y:=y*x\ \{y=0}\ \text{CONS-2}}{\{x=1\}\ x:=x-1\ ;\ y:=y*x\ \{y=0\}}\ \text{SEQ}$$

- The proofs for CONS-1 are trivial (as explained on the previous slide) since **x=0 |= x=0** and **x=1 |= x-1=1-1=0**; and the proofs for CONS-2 are also trivial as **y=0 |= y=0** and **x=0 |= y*x=y*0=0**

Example: Proof Tree (First Try)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\{n!=(y*x)(x-1)!\}\ y:=y*x\ \{n!=y(x-1)!\}}\,\text{ASS}}{\{n!=yx!\ \&\ \neg(x=1)\}\ y:=y*x\ \{n!=y(x-1)!\}}\,\text{CONS-2} \qquad \dfrac{}{\{n!=y(x-1)!\}\ x:=x-1\ \{n!=yx!\}}\,\text{ASS}}{\{n!=yx!\ \&\ \neg(x=1)\}\ y:=y*x;\ x:=x-1\ \ \{n!=yx!\}}\,\text{SEQ}}{\{n!=yx!\}\ \ \text{while}\ \neg(x=1)\ \text{do}\ (y:=y*x;\ x:=x-1)\ \ \{n!=yx!\ \&\ \neg\neg(x=1)\}}\,\text{LOOP}}{\{y=1\ \&\ x=n\ \&\ n>0\}\ \ \text{while}\ \neg(x=1)\ \text{do}\ (y:=y*x;\ x:=x-1)\ \ \{y=n!\}}\,\text{CONS-1}$$

- As expected, guessing the loop invariant is the hardest part of the proof
- n!=yx! seems a good first choice based on observing some test values

#	0	1	2	3
x	4	3	2	1
y	1	4	12	24

#	0	1	2	...
x	0	-1	-2	...
y	1	0	0	...

#	0	1	2	...
x	-4	-5	-6	...
y	1	-4	20	...

- But we do still need to discharge our obligations for the consequence rules

Example: Proof Obligations (First Try)

(A) CONS-1-Postcondition (OK)

n!=yx! & ¬¬(x=1) |= y=n!

1.	n!=yx!	given
2.	¬¬(x=1)	given
3.	x=1	¬¬ elimination on 2
4.	x!=1!=1	by defn of fac using 3
5.	n!=y1	sub 4 in rhs of 1
6.	n!=y	by defn of mult
7.	y=n!	by symmetry of =

(B) CONS-1-Precondition (OK)

y=1 & x=n & n>0 |= n!=yx!

1.	y=1	given
2.	x=n	given
3.	n>0	given
4.	n!=n!	by reflexivity of =
5.	n!=x!	sub 2 in rhs of 4
6.	x!=1x!	by defn of mult
7.	x!=yx!	sub 1 in rhs of 6
8.	n!=yx!	sub 7 in rhs of 5

(C) CONS-2-Postcondition (OK)

n!=y(x-1)! |= n!=y(x-1)!

1. n!=y(x-1)! given

(D) CONS-2-Precondition (FAIL!)

n!=yx! & ¬(x=1) |= n!=(y*x)(x-1)!

**now we want to try and prove that
n! = yx! = y(x(x-1)!) = (y*x)(x-1)!
using the definition of factorial and
the associativity of multiplication;
but we can't - as it isn't t true if x=0**

**so we must include x≠0 in the invariant;
or even x>0 (to also avoid working with
undefined factorials of negative ints)**

Example: Proof Tree (Final)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\begin{Bmatrix} n!=(y*x)(x-1)! \\ x-1>0 \end{Bmatrix} \; y:=y*x \; \begin{Bmatrix} n!=y(x-1)! \\ x-1>0 \end{Bmatrix}}\ \text{ASS}}{\begin{Bmatrix} n!=yx! \\ x>0 \\ \neg(x=1) \end{Bmatrix} \; y:=y*x \; \begin{Bmatrix} n!=y(x-1)! \\ x-1>0 \end{Bmatrix}}\ \text{CONS-2} \qquad \dfrac{}{\begin{Bmatrix} n!=y(x-1)! \\ x-1>0 \end{Bmatrix} \; x:=x-1 \; \begin{Bmatrix} n!=yx! \\ x>0 \end{Bmatrix}}\ \text{ASS}}{\{n!=yx!\ \&\ x>0\ \&\ \neg(x=1)\}\ y:=y*x;\ x:=x-1\ \ \{n!=yx!\ \&\ x>0\}}\ \text{SEQ}}{\{n!=yx!\ \&\ x>0\}\ \ \text{while}\ \neg(x=1)\ \text{do}\ (y:=y*x;\ x:=x-1)\ \ \{n!=yx!\ \&\ x>0\ \&\ \neg\neg(x=1)\}}\ \text{LOOP}}{\{y=1\ \&\ x=n\ \&\ n>0\}\ \ \text{while}\ \neg(x=1)\ \text{do}\ (y:=y*x;\ x:=x-1)\ \ \{y=n!\}}\ \text{CONS-1}$$

Example: Proof Obligations (Final)

(A) CONS-1-Postcondition (OK)

$n!=yx! \ \& \ x>0 \ \& \ \neg\neg(x=1) \models y=n!$

1.	$n!=yx!$	given
2.	$x>0$	given
3.	$\neg\neg(x=1)$	given
4.	$x=1$	$\neg\neg$ elimination on 3
5.	$x!=1!=1$	by defn of fac using 4
6.	$n!=y1$	sub 5 in rhs of 1
7.	$n!=y$	by defn of mult
8.	$y=n!$	by symmetry of =

(B) CONS-1-Precondition (OK)

$y=1 \ \& \ x=n \ \& \ n>0 \models n!=yx! \ \& \ x>0$

1.	$y=1$	given
2.	$x=n$	given
3.	$n>0$	given
4.	$n!=n!$	by reflexivity of =
5.	$n!=x!$	sub 2 in rhs of 4
6.	$x!=1x!$	by defn of mult
7.	$x!=yx!$	sub 1 in rhs of 6
8.	$n!=yx!$	sub 7 in rhs of 5
9.	$x>0$	sub 2 in lhs of 3

(C) CONS-2-Postcondition (OK)

$n!=y(x-1)! \ \& \ x-1>0 \models n!=y(x-1)! \ \& \ x-1>0$

1.	$n!=y(x-1)!$	given
2.	$x-1>0$	given

(D) CONS-2-Precondition (OK)

$n!=yx! \ \& \ x>0 \ \& \ \neg(x=1) \models n!=(y*x)(x-1)! \ \& \ x-1$

1.	$n!=yx!$	given
2.	$x>0$	given
3.	$\neg(x=1)$	given
4.	$x>1$	from 2 and 3
5.	$x!=x(x-1)!$	by defn of fact using 2
6.	$n!=y(x(x-1)!)$	sub 5 in rhs of 1
7.	$n!=(y*x)(x-1)!$	buy associativity of mult
8.	$x-1>0$	dec both sides of 4