COMP340-08B Lecture Notes

David Goodwin

September 20, 2008

This document was produced using LaTeX and typeset with TeX. Fonts are by `METAFONT` and $\mathcal{AMS}$-LaTeX.

All complaints, comments, corrections and suggestions to David Goodwin (Email: dgsoftnz@gmail.com).

Contents

Chapter 1

Introduction

1.1 Advantages of Formal Methods

1. Reduces the need for debugging
2. Forces a clear formulation of program objectives
3. Makes reusing software easier
4. Commercial imperatives

The subset of our analysis are Pieces of Code - a self contained program (here written as pseudo-code), which acts as a function on its inputs. We assume this pseudo-code is made up of:

1. Usual arithmetic operators and library functions (eg +, $*$, etc)
2. if-then and if-then-else constructs
3. The while construct

Note:

- No type declarations (use context)
- No input/output statements

1.2 State Vector

We assume a piece of code acts on a finite set of variables $x_1, x_2, ..., x_n$ (which may be numeric, Boolean, etc). We call $(x_1, x_2, ..., x_n)$ the Program State: it is transformed (many times usually) as the program runs. The x_i represent all variables used in the program.

Suppose x_i takes values from the dataset X_i, so the state is a "vector" in X_1 x X_2 x ... X_n.

1.3 Most basic code

The most basic piece of code is an assignment, eg $x := 3$. For this "atomic program", th state space as only one component, X ($= \mathbb{N}$ say). All inputs give output.

1.4 Examples

1.4.1 Example 1

$$y := x$$

Here the state space might be $\mathbb{N}$x$\mathbb{N}$If the input state vector is $(x, y) = (2, 3)$, then the output vector is $(2, 2)$.

To build up more complicated programs we need to use conditions or tests. These are predicates on the state space (assumed to be of some easily computed form).

1.4.2 Example 2

The test $(x < y)$ on the state space $\mathbb{N}$x$\mathbb{N}$gives the answer true if $(x, y) = (2, 3)$ and false if $(x, y) = (4, 4)$, etc. We assume such tests are part of the language for building programs. Being predicates, all the usual logical connectives can be applied to them.

Chapter 2

Pieces of Code

Fix the state space $X = X_1$ x X_2 x ... X_n. Then we can recursively define more complex programs as follows:

2.1 Assignment

As we have seen, assignment statements are pieces of code.

2.2 Concatentation

If P_1 and P_2 are two pieces of code, so is $P_1; P_2$, the concatentation (composition) of P_1 with P_2. This is “first P_1, then P_2”, which obviously corresponds to the composition of functions.

2.3 If-then-else

If P_1 and P_2 are two pieces of code, and γ is a test on their state space, then

$$\underline{\text{if}}(\alpha)\ \underline{\text{then}}\ \{P_1\}\ \text{else}\ P_2$$

this is a piece of code.

2.4 While

if α is a test and P a piece of code (both on X), then while(α) $\{P\}$ is a piece of code. We call P the “body of the loop”. Given input state $\underline{x}$, the output is obtained by applying P until α is no longer satisfied, the answer is the value of $\underline{x}$ at the point where this occurs (which may be immediately!)

Note: while(α) $\{P\}$ may not terminate for some input states: The “exit condition” $\neg\alpha$ may never be achieved

2.4.1 Example

Let P = “while($x \geq 1$) $\{x := x + 1\}$”. If the input is $x \geq 1$ then x is increased each time the body of the loop is applied so the exit condition $\neg(x \geq 1) \Leftrightarrow (x < 1)$ is never satisfied. For example, if $x = 1$ initially, then the vector (1) evolves as follows:

$$(1), (2), (3), (4), ...$$

and does not stop. So P is not defined when $x \geq 1$. For $x < 1$, the entry condition is never satisfied so the output is x: $(x), (x)$

2.5 Code

Generally, we must think of pieces of code only as partial functions $X \to X$, whose domains generally can't be determined (by the Halting Problem).

2.5.1 Example One

let $X_1 = X_2 = X_3 = \{0,1\}*$ (all binary strings (*)). Let P = “$h := a; a := b; b := h$”. The state vector generally is $(h, a, b) \in X = x_1$ x x_2 x x_3 Supose input vector is $x = (110, 01, 1001)$.

before:	code:	after:
$(110, 01, 1001)$	$h := a$	$(01, 01, 1001)$
$(01, 01, 1001)$	$a := b$	$(01, 1001, 1001)$
$(01, 1001, 1001)$	$b := h$	$(01, 1001, 01)$

Generally, if the input vector is (p, q, r), then the output is determined via: $(p, q, r) \mapsto (q, q, r) \mapsto (q, r, r) \mapsto (q, r, q)$ so the second and third positions are swapped, and the value originally in the first position is lost.

2.5.2 Example Two

Suppose code is:

```
p:=1;i:=1
while(i =< n)
    {p:=p*i;
    i:=i+1}
```

Suppose the state space is $\mathbb{N}$x$\mathbb{N}$x$\mathbb{N}$, with (p, i, n) a typical state. If input is $(p_0, i_0, 4)$, where p_0, i_0 are any “default values”, we get:

$$(p_0, i_0, 4) \mapsto (1, i_0, 4) \mapsto (1, 1, 4) \tag{2.1}$$
$$\mapsto (1, 1, 4) \mapsto (2, 2, 4) \tag{2.2}$$
$$\mapsto (1, 2, 4) \mapsto (2, 3, 4) \tag{2.3}$$
$$\mapsto (6, 3, 4) \mapsto (6, 4, 4) \tag{2.4}$$
$$\mapsto (24, 4, 4) \mapsto (24, 5, 4) \tag{2.5}$$

where 2.1 is after two assignments, 2.2 is after one loop, 2.3 is after two loops, 2.4 is after three loops and 2.5 is the output state after the exit condition is satisfied.

This piece of code seems to compute $p = n!$. How to proive this? How to assert it even?

Chapter 3

Floyd-Hoare Logic

3.1 Hoare Tripple

A Hoare tripple is a statement $\langle\alpha\rangle\ P\ \langle\beta\rangle$ where α, β are tests on the state space X and P a program (= piece of code) acting on X. We say the tripple is partially correct if, whenever $\underline{x} \in X \text{satisfies} \alpha$, and P is applied to $\underline{x}$, the result (if it exists) satisfies β. It is totally correct if any input $\underline{x}$ satisfying α does give a result when P is applied (P terminates on $\underline{x}$), and this result satisfies β.

Recalling the factorial program P, it appears that $\langle n \geq 1\rangle\ P\ \langle P = n!\rangle$ is both partially and totally correct. Note taht neither $(n \geq 1)$ nor $(p = n!)$ involves the varible "i", so its value is not relevant to the correctness.

We call the α, β in $\langle\alpha\rangle\ P\ \langle\beta\rangle$ the precondition and postcondition respectively.

3.1.1 Example One

if X = $\mathbb{R}$x$\mathbb{R}$, the tripple $\langle y \neq 0\rangle\ x := 1/y\ \langle x = 1/y\rangle$ is totally correct where $(x, y) \in \mathbb{R}\text{x}\mathbb{R}$.

3.1.2 Example Two

$\langle\ \rangle\ a := b\ \langle a = b\rangle$ is totally correct for any choice of state space $X_1 \text{x} X_1$. Here $\langle\ \rangle$ is called the empty precondition, equivalent to $\langle\underline{\text{true}}\rangle$ - it is always satisfied.

3.1.3 Example Three

Pick $X = \mathbb{Z}$, the integers.

$$\langle\ \rangle\ \underline{\text{while}}(P \geq 1)\{P := P + 1\ \langle P < 1\rangle$$

is partially correct (as if $P < 1$, the entry condition is not satisfied and if $P \geq 1$ it does not terminate). But

$$\langle P < 1\rangle\ \underline{\text{while}}(P \geq 1)\{P := P + 1\ \langle P < 1\rangle$$

is totally correct.

3.2 Assignment Rule

The basis of all correctness proofs is provided by the Assignment Rule, which states that the following Hoare tripple is (totally) correct:

$$\langle \alpha[E/x] \rangle \; x := E \; \langle \alpha \rangle$$

Here E is some value, easily computed from the current state variables, either using library functions, or maybe E is a variable or a constant. x is a state variable name, and $\alpha[E/x]$ is the predicate α with x replaced by E where-ever it occurs.

This rule is sound (i.e. does give correct tripples) because if the output to $x := E$ is to satisfy α it is enough that E satisfies the same condition x had to satisfy according to α.

It is also complete (i.e. the precondition is as weak (= general) as possible), so $\alpha[E/x]$ is the weakest possible precondition given the piece of code $x := E$ and postcondition $\langle \alpha \rangle$.

3.2.1 Example One

$$\langle \; \rangle \; x := 2 * y \; \langle x = 2y \rangle$$

(where $x = \mathbb{R}\mathrm{x}\mathbb{R}$) is totally correct, since if we replace x by $E = 2y$ in $\alpha =$ “$(x = 2y)''$ gives “$(2y = 2y)$” which is just True

3.2.2 Example Two

$$\langle x \leq 0 \rangle \; x := -x \; \langle x \geq 0 \rangle$$

is correct, since replacing x by $E = -x$ in $(x \geq 0)$ gives $(-x \geq 0)$ which is logically equivalent to $(x \leq 0)$

3.2.3 Proofs

See Appendix C for two proofs of the Assignment Rule.

3.3 Weakening Rule

(Precondition strengthening, postcondition weakening)

Suppose $\langle \alpha \rangle \; P \; \langle \beta \rangle$ is partially correct. Suppose $\alpha_1 \to \alpha$ is True (if α_1 evaluates to True at $\underline{x}$, so does α), and suppose $\beta \to \beta$ is True . So α_1 is stronger (or as strong as) α, and β is stronger than (or as strong as) β_1. Then also $\langle \alpha_1 \rangle \; P \; \langle \beta_1 \rangle$ is partially correct.

Proof. Suppose $\langle \alpha \rangle \; P \; \langle \beta \rangle$ is partially correct and $\alpha_1 \to \alpha$ and $\beta \to \beta_1$ are both tautologies. Suppose input $\underline{x}$ satisfies α_1. Then $\underline{x}$ satisfies α. So when the code P is applied to x, any output satisfies β (since $\langle \alpha \rangle \; P \; \langle \beta \rangle$ is partially correct). So any such output satisfies β_1 (since $\beta \to \beta_1$ is a tautology). So by definition $\langle \alpha \rangle \; P \; \langle \beta_1 \rangle$ is partially corect. ☐

3.3.1 Rule

The weakening rule can be stated as follows:

$$\frac{\alpha_1 \to \alpha, \beta \to \beta_1, \langle \alpha \rangle \ P \ \langle \beta \rangle}{\langle \alpha_1 \rangle \ P \ \langle \beta_1 \rangle} \tag{3.1}$$

3.3.2 Example

The trippe $\langle \ \rangle \ y := x^2 \ \langle y \geq 0 \rangle$ is correct by the Assignment Rule (if state space is $\mathbb{R}$x$\mathbb{R}$), since we get weakest precondition $\langle x^2 \geq 0 \rangle$ (=) $\langle \ \rangle$. Hence so is $\langle x > -10 \rangle \ y := x^2 \ \langle y \geq -5 \rangle$.

Proof

$$(x > -10) \to \underline{\text{True}} \ , (y \geq 0) \to (y \geq -5)$$

is True , so we use weakening:

$$\frac{(x > -10) \to \text{true}, (y \geq 0) \to (y \geq -5), \langle \ \rangle \ y := x^2 \ \langle y \geq 0 \rangle}{\langle x > -10 \rangle \ y := x^2 \ \langle y \geq -5 \rangle}$$

3.4 Concatenation (Composition) Rule

The rule looks like this:

$$\frac{\langle \alpha \rangle \ P \ \langle \beta \rangle, \langle \beta \rangle \ Q \ \langle \gamma \rangle}{\langle \alpha \rangle \ P; Q \ \langle \gamma \rangle} \tag{3.2}$$

3.4.1 Proof of Soundness

Assume the hypothesis (are partially correct tripples). Suppose input $\underline{x}$ satisfies α. Apply P to it, then Q. Then after applying P to $\underline{x}$, any result satisfies β. So wehn Q is applied to that, the result (if any) satisfies γ (by the correctness of $\langle \beta \rangle \ Q \ \langle \gamma \rangle$). So by definition, $\langle \alpha \rangle \ P; Q \ \langle \gamma \rangle$ is partially correct.

3.4.2 Example

Prove the following is correct:

$$\langle \ \rangle \ c := a + b; c := c/2 \ \langle c = (1 + b)/2 \rangle$$

(assume a, b, c are Real-valued)

Proof. Work backwards: "feed" the postcondition back through the code. Thus, by assignment rule,

$$\langle c = a + b \rangle (=) \langle c/2 = (a + b)/2 \rangle \ c := c/2 \ \langle c = (a + b)/2 \rangle$$

But so is:

$$\langle \ \rangle \Leftrightarrow \langle a + b = a + b \rangle \ c := a + b \ \langle c = a + b \rangle$$

by Assignment. We can now use the Concatenation Rule:

$$\frac{\langle \ \rangle \ c := a + b \ \langle c = a + b \rangle, \langle c = a + b \rangle \ c := c/2 \ \langle c = \frac{a+b}{2} \rangle}{\langle \ \rangle \ c := a + b; c := c/2 \ \langle c = \frac{a+b}{2} \rangle}$$

Typically we would set out such a proof "Vertically":

$$\langle a+b=a+b\rangle \Leftrightarrow \langle\ \rangle$$
$$c := a+b;$$
$$\langle c/2=\frac{a+b}{2}\rangle \Leftrightarrow \langle c=a+b\rangle$$
$$c := c/2$$
$$\langle c=\frac{a+b}{2}\rangle$$

□

The method easily extends to arbitrary long concatenations.

3.5 If-rules

3.6 While Rule

Appendix A

Lecture Date Map

This document is a collection notes sorted by topic. This appendix covers which bits of the notes came from which lectures

Date	Content
9 September 2008	chapter 1, The intro to chapter 2 and its sections 2.1, 2.2, 2.3
10 September 2008	Sections 2.4 and 2.5 in chapter 2
11 September 2008	Sections 3.1 and 3.2 in Chapter 3.
16 September 2008	Sections 3.3 and 3.4 in Chapter 3, Appendix C
17 September 2008	Section 3.5 of Chapter 3.
18 September 2008	Section 3.6 of Chapter 3.

Appendix B

Lecture Start Notes

Notes, announcements and revision content from the beginning of the lectures is not included inline with the notes. It is instead recorded here when present. In the original paper notes, the appropriate section here would come first followed by the content specified in appendix A for the specific date.

B.1 9 September 2008

- Assignments are to be handed in on the third floor of G block under the Mathematics reception counter
- Chapters 4.1, 4.2 and 4.3 of the text book will be covered
- The rest of the paper will cover logical & algebraic methods for reasoning about (the correctness of) programs

B.2 10 September 2008

B.2.1 Revision

$$\text{if}(\alpha) \text{ then } \{P_1\} \text{ else } \{P_2\}$$

is a piece of code. It acts as expected on the state $\underline{x}$: if α is $\underline{\text{true}}$ at $\underline{x}$ then the output agrees with that of P_1, otherwise it agrees with P_2's.

Also used is $\underline{\text{if-then}}$ without "else": $\underline{\text{if}}(\alpha)$ $\underline{\text{then}}$ $\{P_1\}$ is a piece of code which, if the input $\underline{x}$ satisfies α, gives the same output as P_1, otherwise the output is $\underline{x}$ itself.

Note: the underlining in $\underline{x}$ means its a vector

B.3 11 September 2008

The tutorial is now in K.G.06.

B.4 16 September 2008

A copy of two hand-written rigorous proofs of the Assignment Rule were handed out. See Appendix C sections C.1 and C.2.

B.5 17 September 2008

No pre-content notes were recorded for this lecture.

B.6 18 September 2008

No pre-content notes were recorded for this lecture.

Appendix C

Rigorous Proof of Assignment Rule

Note: This was copied from a photocopy of handwritten text. Difficulties in reading such text can result in copy errors.

C.1 Proof One

Proof.

$$\langle \alpha[E/x] \rangle \; x := E \; \langle \alpha \rangle \tag{C.1}$$

Say state variables are $x_1, x_2, \ldots, x_n, \;\; x_i \in X_i$ (without loss of generality). Now suppose assignment is $x_1 := E(x_1, x_2, \ldots, x_n)$ (wlog). Let:

$$\begin{aligned} \beta(x_1, \ldots, x_n) &= \alpha[E(x_1, \ldots, x_n), x_2, \ldots, x_n] \\ &= \alpha[E/x_1] \end{aligned}$$

another predicate.

Suppose an input state vector $(a_1, a_2, \ldots, a_n)$ satisfies β, so $\alpha(E(a_1, a_2, \ldots, a_n), a_2, \ldots, a_n)$ satisfies β, so $\alpha(E(a_1, a_2, \ldots, a_n), a_2, \ldots, a_n)$ is $\underline{\text{True}}$. Now set $b_1 = E(a_1, a_2, \ldots, a_n)$, so output vector is $(E(a_1, a_2, \ldots, a_n), a_2, \ldots, a_n)$. Then $\alpha(b_1, a_2, \ldots, a_n) = \beta(a_1, a_2, \ldots, a_n)$ is $\underline{\text{True}}$. So this shows equation C.1 is $\underline{\text{correct}}$.

Is $\beta = \alpha[E/x_1]$ as general as possible? Need to show any input to the assignment that satisfies the postcondition must also satisfiy the precondition. Put another way, we need to show that if the precondition β is $\underline{\text{not}}$ satisfied, nor can the postcondition be.

So say $\beta(a_1, a - 2, \ldots, a_n) = \alpha(E(a_1, a_2, \ldots, a_n), a_2, \ldots, a_n)$ is $\underline{\text{False}}$. Then letting $b_1 = E(a_1, a_2, \ldots, a_n)$, the output of the assignment given input $(a_1, a_2, \ldots, a_n)$ is $(b_1, a_2, \ldots, a_n)$, and of course $\alpha(b_1, a_2, \ldots, a_n) = \beta(a_1, a_2, \ldots, a_n)$, which is $\underline{\text{False}}$. □

C.2 Proof Two

Proof.

$$\langle \alpha[E/x] \rangle \; x := E \; \langle \alpha \rangle$$

Suppose the state variables are $x_1, x_2, \ldots, x_n$. Suppose that the assignment has the form $x_1 := E(x_1, x_2, \ldots, x_n)$, with no loss of generality.

Define $\beta(x_1, x_2, \ldots, x_n) = \alpha(E(x_1, \ldots, x_n), x_2, \ldots, x_n)$, a test. Then let $(a_1, a_2, \ldots, a_n)$ be a state vector for input.

Following the application of the assignment, the output is

$$E(a_1, a_2, \ldots, a_n), a_2, \ldots, a_n) = (b_1, a_2, \ldots, a_n)$$

Clearly then, $\beta(a_1, a_2, \ldots, a_n) = \alpha(b_1, a_2, \ldots, a_n)$. So $(a_1, a_2, \ldots, a_n)$ satisfies $\beta = \alpha[E/x]$ if and only if $(b_1, a_2, \ldots, a_n)$ satisfies α

So $\alpha[E/x_1]$ counts as <u>True</u> every possible input to the assignment that gives an output satisfying α. So $\beta = \alpha[E/x_1]$ is the most general possible precondition making the tripple correct. $\square$