Lecture 8:
Floyd-Hoare Logic for Partial Correctness

Aims:

- To look at the following inference rules
  - sequence;
  - assignment and
  - consequence.

8.1 The Deduction System for Partial Correctness

- The inference rules we are about to look at are credited to R. Floyd and C.A.R. Hoare. Hence, we sometimes call this *Floyd-Hoare Logic.*

  There are inference rules for each of the constructs of the programming language. This allows us to compose proofs, command by command.

- A global piece of advice is to read the rules 'backwards': from bottom to top.
- In a lot of the examples and exercises, I won't use specification variables, even though the program specifications might be better if I did. This is OK because we're proving programs rather than constructing them and the ones I will generally give you to prove will not exploit the laxity of specification — in particular, the programs I give you to prove will not directly assign to the parameters.

8.1.1 Sequencing

- Sequencing

$$\frac{(\!|\, P \,|\!)\, C_1 \,(\!|\, R \,|\!), \quad (\!|\, R \,|\!)\, C_2 \,(\!|\, Q \,|\!)}{(\!|\, P \,|\!)\, C_1; C_2 \,(\!|\, Q \,|\!)}$$

  So, if you know that C_1 takes you from states satisfying P to states satisfying R, and that C_2 takes you from states satisfying R to states satisfying Q, then you know that executing C_1 and C_2 in sequence will take you from states satisfying P to states satisfying R.

  But, as per my piece of global advice, when we use this inference rule in program verification, we read it from bottom to top. In order to prove that the sequence $C_1; C_2$ will take us from states satisfying P to states satisfying Q, we need to find an R and prove that $(\!|\, R \,|\!)\, C_2 \,(\!|\, Q \,|\!)$ and then that $(\!|\, P \,|\!)\, C_1 \,(\!|\, R \,|\!)$.

8.1.2 Assignment

- *Notation:* Let $Q[V \mapsto E]$ be the result of replacing all occurrences of variable V in Q by expression E.
- Assignment

  $$\frac{}{(\!|\, Q[V \mapsto E] \,|\!)\; V := E \;(\!|\, Q \,|\!)}$$

  Since this inference rule has no conditions, it is more accurate to call it an axiom, but we won't worry about that.

  The rule states that if we want to show that Q holds after the assignment of E into V, then we must show that $Q[V \mapsto E]$ holds before the assignment.
- In program verification, we will apply this 'rule' backwards. We know Q and we wish to find a precondition that makes Q true after the assignment $V := E$.
- The nice thing about this 'rule' is that it is completely mechanical. You just do a substitution and, hey presto, that gives you the precondition. Unfortunately, not all the inference rules are so simple; some require ingenuity.
- Applying the rule mechanically and 'backwards' in this way will generate the *weakest precondition* that is needed for execution of the assignment to result in a state that satisfies the postcondition.
- However, students don't like this rule! It appears to be written the wrong way round.

  First, let's look at some examples that show that it does work.
- $\vdash_{\text{par}} (\!|\, y = 2 \,|\!)\; x := 2 \;(\!|\, y = x \,|\!)$

  The postcondition is $y = x$. We do a substitution to get the precondition: $y = x[x \mapsto 2]$. This gives $y = 2$.

  And we can read the statement as follows: if you want to prove that $y = x$ after the assignment $x := 2$, then you must be able to prove that $y = 2$ before it.
- $\vdash_{\text{par}} (\!|\, 2 > 0 \,|\!)\; x := 2 \;(\!|\, x > 0 \,|\!)$

  If you want to prove that $x > 0$ after $x := 2$, then you must be able to prove that $2 > 0$ before it.
- $\vdash_{\text{par}} (\!|\, x + 1 = 2 \,|\!)\; x := x + 1 \;(\!|\, x = 2 \,|\!)$

  If you want to prove that $x = 2$ after $x := x + 1$, then you must be able to prove that $x + 1 = 2$ (i.e. $x = 1$) before it.
- I'll give some more examples, but I won't include any more paraphrases into English. You can do the paraphrases yourself.

  $\vdash_{\text{par}} (\!|\, x + 1 = y \,|\!)\; x := x + 1 \;(\!|\, x = y \,|\!)$

  $\vdash_{\text{par}} (\!|\, x + 1 + 5 = y \,|\!)\; x := x + 1 \;(\!|\, x + 5 = y \,|\!)$

  $\vdash_{\text{par}} (\!|\, x + 1 > 0 \land y > 0 \,|\!)\; x := x + 1 \;(\!|\, x > 0 \land y > 0 \,|\!)$
- So we've seen that, although you may feel it has been written the wrong way round, it does seem to work. The key to making sense of it is to think about what you need to prove about the precondition. That's the way our paraphrases have been written.
- Another way to try to convince you that the rule is correct is to show you that various alternative rules get things wrong. We'll look at two bad (incorrect) attempts.

- Bad-Assignment-1

  $$\frac{}{\llparenthesis\, P \,\rrparenthesis\, V := E \,\llparenthesis\, P[V \mapsto E] \,\rrparenthesis}$$

  Example of this getting it wrong; it allows us to prove the following:

  $\vdash_{\text{par}} \llparenthesis\, x = 0 \,\rrparenthesis\, x := 1 \,\llparenthesis\, 1 = 0 \,\rrparenthesis$

  Need I say more? (Yes? Then you haven't read it!)

  Well, if Bad-Assignment-1 isn't right, then what about this one:

- Bad-Assignment-2

  $$\frac{}{\llparenthesis\, P \,\rrparenthesis\, V := E \,\llparenthesis\, P[E \mapsto V] \,\rrparenthesis}$$

  Example of this getting it wrong; it allows us to prove the following:

  $\vdash_{\text{par}} \llparenthesis\, x = 0 \,\rrparenthesis\, x := 1 \,\llparenthesis\, x = 0 \,\rrparenthesis$

  The rule tells us to replace E by V. In this case, E is 1. But 1 doesn't occur in P ($x = 0$), so the result of the substitution is still $x = 0$.

Class Exercise

- Determine the weakest precondition P that satisfies
  1. $\llparenthesis\, P \,\rrparenthesis\, x := x + 1 \,\llparenthesis\, x > 0 \,\rrparenthesis$
  2. $\llparenthesis\, P \,\rrparenthesis\, x := x^2 \,\llparenthesis\, x > 0 \,\rrparenthesis$
  3. $\llparenthesis\, P \,\rrparenthesis\, x := x + 1 \,\llparenthesis\, x^3 - 5x^2 + 2x > 0 \,\rrparenthesis$
  4. $\llparenthesis\, P \,\rrparenthesis\, x := x + 1 \,\llparenthesis\, x = x_0 + 1 \,\rrparenthesis$
  5. $\llparenthesis\, P \,\rrparenthesis\, x := y \bmod 2 \,\llparenthesis\, x = y \bmod 2 \,\rrparenthesis$
- Determine the weakest precondition P that satisfies
  1. $\llparenthesis\, P \,\rrparenthesis\, x := x + 1; x := x + 1 \,\llparenthesis\, x > 0 \,\rrparenthesis$
  2. $\llparenthesis\, P \,\rrparenthesis\, \mathit{temp} := x; x := y; y := \mathit{temp} \,\llparenthesis\, x = y_0 \wedge y = x_0 \,\rrparenthesis$
  3. $\llparenthesis\, P \,\rrparenthesis\, x := x + y; y := x - y; x := x - y \,\llparenthesis\, x = m \wedge y = n \,\rrparenthesis$

8.1.3 Rule of Consequence

- Consequence

  $$\frac{P \Rightarrow P', \quad \llparenthesis\, P' \,\rrparenthesis\, C \,\llparenthesis\, Q' \,\rrparenthesis, \quad Q' \Rightarrow Q}{\llparenthesis\, P \,\rrparenthesis\, C \,\llparenthesis\, Q \,\rrparenthesis}$$

  This rule tells us that if we have proved $\llparenthesis\, P' \,\rrparenthesis\, C \,\llparenthesis\, Q' \,\rrparenthesis$ and we have a wff P which 'implies' P' and another one Q which is 'implied' by Q', then we can conclude that $\llparenthesis\, P \,\rrparenthesis\, C \,\llparenthesis\, Q \,\rrparenthesis$. In some books, this one rule is presented as two separate rules, referred to as *precondition strengthening* and *postcondition weakening*.

- One thing to note about this rule is that two of its conditions have nothing to do with programs and their partial correctness. Instead these two conditions require you to prove statements of 'normal' logic (about P, P', Q and Q'). You would prove these using the kinds of techniques we were exploring in the lecture on proof. (Use the methods for informal proof.)

  Hence, this rule acts as a link between Floyd-Hoare logic (for programs) and 'normal' logic.

  Its greatest use for us is in glueing together different components of proofs.

- E.g. We can prove that

  $\vdash_{\text{par}} (\!|\, x = n \,|\!)\; x := x + 1 \;(\!|\, x = n + 1 \,|\!)$

- This time we are not looking for a weakest precondition P. Instead, we've been given a precondition $(x = n)$.

  So we have to lay this one out as a proof.

  We write out the precondition, the program and the postcondition with lots of blank lines.

```
(| x = n |)

x := x + 1;
(| x = n + 1 |)
```

- We start with the postcondition and push it upwards through the program until a wff P' emerges at the top. Ideally, P' is the weakest precondition which guarantees that Q will hold if the program is executed and terminates. We then check to see whether P' follows from P, the given precondition. This is a use of the rule of consequence.

  As we proceed, we fill in weakest preconditions and we add comments that justify the different steps.

  So, in our example, let's push the postcondition upwards through the assignment command. This involves using the Assignment rule of inference, we write the precondition above the assignment command. We write the comment alongside the postcondition. This is logical because, when we read the final proof we will read it in a forwards direction.

```
(| x = n |)
(| x + 1 = n + 1 |)
x := x + 1;
(| x = n + 1 |)Assignment
```

  To finish this proof, we need to show that $x = n \Rightarrow x + 1 = n + 1$.

  So we would write an extra justification alongside the program and show where to find the proof.

```
(| x = n |)
(| x + 1 = n + 1 |)Consequence (proof ①)
x := x + 1;
(| x = n + 1 |)Assignment
```

  Proof ①: To show $x = n \Rightarrow x + 1 = n + 1$. Assume $x = n$. By arithmetic, we get $x + 1 = n + 1$. Discharging the assumption, we have $x = n \Rightarrow x + 1 = n + 1$ as desired.

- In the real world, when the proofs are really simple (like this), it suffices to write 'Consequence' and omit the proof. *But if you do this in exercises and exam questions, do not expect credit. If you want credit, include your proofs!*

- Here's another example. In the lecture, we'll prove that

  $\vdash_{\text{par}} (\!|\, \textbf{True} \,|\!)\; y := x; y := y + y \;(\!|\, y = 2x \,|\!)$

Acknowledgements

I continue to base material on that in Chapter 4 of [HR00].

References

[HR00] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 2000.