Lecture 12:
Total Correctness

Aims:

- To look at the inference rules for total correctness.

12.1 Total Correctness

- Suppose we are asked to prove

  $\vdash_{\text{tot}} (\!| P |\!)\, C\, (\!| Q |\!)$

  So now we are concerned with termination as well as partial correctness.

- We need a different deduction system with a different set of inference rules. In a language as simple as MOCCA, there are only two kinds of problem that can prevent termination.

  - The value of an expression may be undefined, e.g. division by zero.
  - A **while** loop may never be exited.

  Of course, if we extend our language, then there could be other sources of non-termination. For example, if we add procedures and we allow recursion then any command that contains a call to a recursive procedure (including cases of mutual recursion) may result in an infinite computation.

12.2 A Deduction System for Total Correctness

- Most of the inference rules are unchanged. But wherever we evaluate an expression, we must make sure that its value is defined. For example, if we have the following assignments,

  $$x := 10 \operatorname{div} u$$

  $$x := 32 \bmod v$$

  $$x := z \operatorname{div} (w + 1)$$

  $$x := 1 + z \bmod (x - y)$$

  we must make sure that u, v, $(w + 1)$ and $(x - y)$ are not zero, respectively.

  If we have an **if** command or a **while** loop, the Boolean expression may involve some arithmetic, and again this gives the potential for a division by zero. Here are examples:

  $$\textbf{if } x = 10 \operatorname{div} y \ldots$$

  $$\textbf{while } x = 32 \bmod y \ldots$$

- In our new inference rules, we write 'E is defined'. But you don't write this in a proof. You replace it by a suitable assertion. So if, within E, there's a division (using div or mod) by some expression E', you replace 'E is defined' by $E' \neq 0$. If there's no division, then simply ignore 'E is defined'.

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

  Assignment
  $$\frac{E \text{ is defined}}{(\!|\, Q[V \mapsto E] \,|\!)\, V := E\, (\!|\, Q \,|\!)}$$

  Consequence
  $$\frac{P \Rightarrow P', \quad (\!|\, P' \,|\!)\, C\, (\!|\, Q' \,|\!), \quad Q' \Rightarrow Q}{(\!|\, P \,|\!)\, C\, (\!|\, Q \,|\!)}$$

  One-armed-conditional
  $$\frac{B \text{ is defined}, \quad (\!|\, B \wedge P \,|\!)\, C\, (\!|\, Q \,|\!), \quad (\neg B \wedge P) \Rightarrow Q}{(\!|\, P \,|\!)\, \textbf{if } B\ C\, (\!|\, Q \,|\!)}$$

  Two-armed-conditional
  $$\frac{B \text{ is defined}, \quad (\!|\, B \wedge P \,|\!)\, C_1\, (\!|\, Q \,|\!), \quad (\!|\, \neg B \wedge P \,|\!)\, C_2\, (\!|\, Q \,|\!)}{(\!|\, P \,|\!)\, \textbf{if } B\ C_1 \textbf{ else } C_2\, (\!|\, Q \,|\!)}$$

- The rule for **while** loops, however, is much more complicated.

  While
  $$\frac{\begin{array}{c} B \text{ is defined}, \\ (\!|\, \mathit{Inv} \wedge B \wedge 0 \leq \mathit{VE} \wedge \mathit{VE} = \mathit{VE}_0 \,|\!)\, C\, (\!|\, \mathit{Inv} \wedge 0 \leq \mathit{VE} \wedge \mathit{VE} < \mathit{VE}_0 \,|\!) \end{array}}{(\!|\, \mathit{Inv} \wedge 0 \leq \mathit{VE} \,|\!)\, \textbf{while } B\ C\, (\!|\, \mathit{Inv} \wedge \neg B \,|\!)}$$

- We have to try to discover a *variant*. A *variant* is an integer expression whose value can be shown to decrease every time we go round the loop, but which is always non-negative. If we can find an expression that has these properties, the loop must terminate. The expression can only decrease in value a finite number of times before it becomes zero.

  In the rule above VE is the variant. We require that its value decreases by saying in the condition of the rule that, if its value is VE_0 before the body, then its value is less than VE_0 after the body.

- How do we prove the total correctness of a **while** loop?

  1. Guess a wff Inv that you hope is an invariant and an integer expression VE that you hope is a variant.
  2. Prove that $(\mathit{Inv} \wedge \neg B) \Rightarrow Q$.
  3. Push $\mathit{Inv} \wedge 0 \leq \mathit{VE} \wedge \mathit{VE} < \mathit{VE}_0$ upwards through C. Let's call the wff you get from this W.
  4. Prove that $(\mathit{Inv} \wedge B \wedge 0 \leq \mathit{VE} \wedge \mathit{VE} = \mathit{VE}_0) \Rightarrow W$.
  5. Now write $\mathit{Inv} \wedge 0 \leq \mathit{VE}$ above the **while** loop. (Continue to push this up through the rest of the program, if any.)

- Prove that $\vdash_{\text{tot}} (\!| x \geq 0 |\!)\ \mathit{ProgA}\ (\!| y = x! |\!)$ where ProgA is:

$y := 1;$

$z := 0;$

while $x \neq z$
{

$z := z + 1;$

$y := y \times z;$

}

  - The invariant is $y = z!$ as before.
  - The variant is $x - z$.

  We can see that loop-test B, i.e. $x \neq z$, and all the expressions in the assignment commands are defined (no uses of div or mod).

Acknowledgements

My approach here was partly based on the approach taken in [Kal90] as well as [HR00].

References

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

[Kal90] A. Kaldewaij. *Programming: The Derivation of Algorithms*. Prentice Hall, 1990.