5 Total Correctness

5.1 Termination

Total Correctness Specification

- So far our discussion has been concerned with partial correctness
  - what about termination?
- A total correctness specification \([P] C [Q]\) is true if and only if:
  - whenever \(C\) is executed in a state satisfying \(P\), then the execution of \(C\) terminates
  - after \(C\) terminates \(Q\) holds
- With the exception of the WHILE rule, all the axioms and rules described so far are sound for total correctness as well as partial correctness

Termination of WHILE Commands

- WHILE commands are the only commands that might not terminate

Consider now the following proof:

\[ \vdash \{ T \} \text{WHILE } T \text{ DO SKIP } \{ T \land \neg T \} \]

\[ = \{ \text{WHILE rule} \} \]

\[ \vdash \{ T \land T \} \text{SKIP } \{ T \} \]

\[ = \{ \text{precondition strengthening} \} \]

\[ \vdash \{ T \} \text{SKIP } \{ T \} \]

\[ = \{ \text{SKIP axiom} \} \]

True

- If the WHILE rule were true for total correctness, then the proof above would show that:
  \[ \vdash \{ T \} \text{WHILE } T \text{ DO SKIP } \{ T \land \neg T \} \]

- Thus the WHILE rule is unsound for total correctness

5.2 Total Correctness Rules

Rules for Non-Looping Commands

- Replace \{ and \} by [ and ], respectively, in:
  - Assignment axiom (see next slide for discussion)
  - Consequence rules
  - Conditional rules
  - Sequencing rule
  - Block rule

- The following is a valid derived rule:

\[
\frac{\{P\} C \{Q\}}{\{P\} C [Q]}
\]

if \(C\) contains no WHILE commands
Total Correctness Assignment Axiom

- Assignment axiom for total correctness:
  $\vdash [P[E/V]] V := E [P]$
- Note that the assignment axiom for total correctness states that assignment commands always terminate
- So all function applications in expressions must terminate
- This might not be the case if functions could be defined recursively
- Consider the assignment: $X := fact(-1)$, where $fact(n)$ is defined recursively by:
  
  $$fact(n) = \text{if } n = 0 \text{ then } 1 \text{ else } n \times fact(n - 1)$$

5.3 Error Termination

Error Termination

- It is also assumed that erroneous expressions like $1/0$ do not cause problems
- Most programming languages will cause an error stop when division by zero is encountered
- In our logic it follows that:
  $\vdash \top X := 1/0 \ [X = 1/0]$
- The assignment $X := 1/0$ halts in a state in which $X = 1/0$ holds
- This assumes that $1/0$ denotes some value that $X$ can have

Error Termination

- There are two possibilities:
  1. $1/0$ denotes some number
  2. $1/0$ denotes some kind of ‘error value’
- It seems at first sight that adopting 2 above is the most natural choice
  - this makes it tricky to see what arithmetical laws should hold
  - is $(1/0) \times 0$ equal to 0 or to some ‘error value’?
  - if the latter, then it is no longer the case that $n \times 0 = 0$ is a valid general law of arithmetic
- It is possible to make everything work with undefined and/or error values, but the resultant theory is a bit messy

Example

- We assume that arithmetic expressions always denote numbers
- In some cases exactly what the number is will be not fully specified
  - for example, we will assume that $m/n$ denotes a number for any $m$ and $n$
  - the only property assumed about $m/n$ is: $\neg (n = 0) \Rightarrow (m/n) \times n = m$
  - it is not possible to deduce anything about $m/0$ from this
  - in particular it is not possible to deduce that $(m/0) \times 0 = 0$
  - but $(m/0) \times 0 = 0$ does follow from $n \times 0 = 0$
5.4 \textbf{WHILE Rule for Total Correctness}

\textbf{WHILE Rule for Total Correctness}

- WHILE commands are the only commands in our little language that can cause non-termination
  - they are thus the only kind of command with a non-trivial termination rule
- The idea behind the \texttt{WHILE} rule for total correctness is:
  - to prove \texttt{WHILE S DO C} terminates
  - show that some non-negative quantity decreases on each iteration of \texttt{C}
  - this decreasing quantity is called a \textit{variant}

\textbf{WHILE Rule for Total Correctness}

- In the rule below, the variant is \textit{E}, and the fact that it decreases is specified with an auxiliary variable \textit{n}
- The hypothesis $\vdash P \land S \Rightarrow E \geq 0$ ensures the variant is non-negative

\begin{center}
\textbf{WHILE Rule for Total Correctness}

$\vdash [P \land S \land (E = n)] \ C \ [P \land (E < n)], \vdash P \land S \Rightarrow E \geq 0$

$\vdash [P] \ \text{WHILE} \ S \ \text{DO} \ C \ [P \land \neg S]$

where \textit{E} is an integer-valued expression
and \textit{n} is an identifier not occurring in \textit{P}, \textit{C}, \textit{S} or \textit{E}.
\end{center}

\textbf{Example}

- We show:
  $\vdash [Y > 0]$
  \begin{align*}
  \text{WHILE} \ Y \leq R \ \text{DO} \ & R := R - Y; \ Q := Q + 1 \ \text{END} \\
  [T]
  \end{align*}
- Take:
  $P = Y > 0$
  $S = Y \leq R$
  $E = R$
  $C = \text{BEGIN} \ R := R - Y; \ Q := Q + 1 \ \text{END}$
- We want to show $\vdash [P] \ \text{WHILE} \ S \ \text{DO} \ C \ [T]$
- By the \texttt{WHILE} rule for total correctness it is sufficient to show:
  1. $\vdash [P \land S \land (E = n)] \ C \ [P \land (E < n)]$
  2. $\vdash P \land S \Rightarrow E \geq 0$
Example

- The first of these can be proved by establishing:
  \[ \vdash \{ P \land S \land (E = n) \} \ C \ \{ P \land (E < n) \} \]

- Then using the total correctness rule for non-looping commands

- The verification condition for:
  \[ \vdash \{ P \land S \land (E = n) \} \ C \ \{ P \land (E < n) \} \]
  is:
  \[ Y > 0 \land Y \leq R \land R = n \Rightarrow (Y > 0 \land R < n) [Q + 1/Q] [R - Y/R] \]
  i.e.
  \[ Y > 0 \land Y \leq R \land R = n \Rightarrow Y > 0 \land R - Y < n \]
  which follows from the laws of arithmetic

- The second subgoal (2) is just:
  \[ \vdash Y > 0 \land Y \leq R \Rightarrow R \geq 0 \]
  which follows from the laws of arithmetic

Termination Specifications

- The relation between partial and total correctness is informally given by the equation:

  \[ Total \ correctness = \text{Termination} + \text{Partial correctness} \]

- This informal equation can be represented by the following two formal rules of inference:

\[
\begin{align*}
\vdash & P \ C \ {\{Q\}}, \quad \vdash [P] \ C \ [T] \\
\quad \vdash & [P] \ C \ [Q] \\
\vdash & [P] \ C \ [Q], \quad \vdash [P] \ C \ [T]
\end{align*}
\]

5.5 Derived Rules

Derived Rules

- Multiple step rules for total correctness can be derived in the same way as for partial correctness
  - the rules are the same up to the brackets used
  - same derivations with total correctness rules replacing partial correctness ones

- Example: the derived SKIP rule

\[
\begin{align*}
\vdash & P \Rightarrow Q \\
\vdash & [P] \text{SKIP} \ [Q]
\end{align*}
\]
The Derived WHILE Rule

- The derived WHILE rule needs to handle the variant

\[
\text{The Derived WHILE Rule}
\]

\[
\begin{align*}
\vdash & P \Rightarrow R \\
\vdash & R \land S \Rightarrow E \geq 0 \\
\vdash & R \land \neg S \Rightarrow Q \\
\vdash & [R \land S \land (E = n)] C [R \land (E < n)] \\
\vdash & [P] \text{WHILE } S \text{ DO } C [Q]
\end{align*}
\]

5.6 Verification Conditions for Termination

VCs for Termination

- Verification conditions are easily extended to deal with total correctness
- To generate total correctness verification conditions for WHILE commands, it is necessary to add a variant as an annotation in addition to an invariant
- No other extra annotations are needed for total correctness
- We assume this is added directly after the invariant, surrounded by square brackets
- VC generation algorithm same as for partial correctness

WHILE Annotation

- A correctly annotated total correctness specification of a WHILE command thus has the form:

\[
[P] \text{WHILE } S \text{ DO } \{R\} [E] C [Q]
\]

where \(R\) is the invariant and \(E\) the variant
- Note that the variant is intended to be a non-negative expression that decreases each time around the WHILE loop
- The other annotations, which are enclosed in curly brackets, are meant to be conditions that are true whenever control reaches them

VCs for WHILE Commands

- A correctly annotated specification of a WHILE command has the form:

\[
[P] \text{WHILE } S \text{ DO } \{R\} [E] C [Q]
\]
WHILE Commands

The verification conditions generated by:

\[
[P] \text{WHILE } S \text{ DO \{} [R] \} [E] C [Q]
\]

are:

1. \( P \Rightarrow R \)
2. \( R \land \neg S \Rightarrow Q \)
3. \( R \land S \Rightarrow E \geq 0 \)
4. The verification conditions generated by:

\[
[R \land S \land (E = n)] C [R \land (E < n)]
\]

where \( n \) does not occur in \( P, R, E, C, S \) or \( Q \).

Example

The verification conditions for:

\[
[R = X \land Q = 0] \text{WHILE } Y \leq R \text{ DO \{} [X = R + Y \times Q] [R] \}
\]

BEGIN \( R := R - Y; \ Q := Q + 1 \) END

\[
[X = R + (Y \times Q) \land R < Y]
\]

are:

1. \( R = X \land Q = 0 \Rightarrow (X = R + (Y \times Q)) \)
2. \( X = R + Y \times Q \land \neg(Y \leq R) \Rightarrow (X = R + (Y \times Q) \land R < Y) \)
3. \( X = R + Y \times Q \land Y \leq R \Rightarrow R \geq 0 \)

together with the verification conditions for:

\[
[X = R + (Y \times Q) \land (Y \leq R) \land (R = n)] \text{BEGIN } R := R - Y; \ Q := Q + 1 \text{ END}
\]

\[
[X = R + (Y \times Q) \land (R < n)]
\]

Example

- The single verification condition for:

\[
[X = R + (Y \times Q) \land (Y \leq R) \land (R = n)] \text{BEGIN } R := R - Y; \ Q := Q + 1 \text{ END}
\]

\[
[X = R + (Y \times Q) \land (R < n)]
\]

is:

\[
X = R + (Y \times Q) \land (Y \leq R) \land (R = n) \Rightarrow
X = (R - Y) + (Y \times (Q + 1)) \land ((R - Y) < n)
\]

- But this isn’t true (take \( Y = 0 \))

- To prove \( R - Y < n \) we need to know \( Y > 0 \)

- Exercise: Explain why one would not expect to be able to prove the verification conditions of this last example

- Hint: Consider the original specification