Proving Programs Terminate

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

1.	$\vdash\ \{\mathtt{T}\}\ \mathtt{SKIP}\ \{\mathtt{T}\}$	(SKIP axiom)
2.	$\vdash\ \{\mathtt{T} \wedge \mathtt{T}\}\ \mathtt{SKIP}\ \{\mathtt{T}\}$	(precondition strengthening)
3.	$\vdash\ \{\mathtt{T}\}\ \mathtt{WHILE\ T\ DO\ SKIP}\ \{\mathtt{T} \wedge \neg\mathtt{T}\}$	(2 and the WHILE-rule)

- If the WHILE-rule were true for total correctness, then the proof above would show that

$$\vdash\ [\mathtt{T}]\ \mathtt{WHILE\ T\ DO\ SKIP}\ [\mathtt{T} \wedge \neg\mathtt{T}]$$

- Thus the WHILE-rule is unsound for total correctness

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**

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

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**

$$fact(n) \ = \ \textbf{if } n = 0 \textbf{ then } 1 \textbf{ else } n \times fact(n-1)$$

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 encountered**

- **In our logic it follows that**

$$\vdash \ [\mathtt{T}]\ \mathtt{X}\ \mathtt{:=}\ 1/0\ [\mathtt{X} = 1/0]$$

- **The assignment $\mathtt{X}\ \mathtt{:=}\ 1/0$ halts in a state in which $\mathtt{X} = 1/0$ holds**

- **This assumes that $1/0$ denotes some value that $\mathtt{X}$ can have**

Two Possibilities

- **There are two possibilities**

  (i) $1/0$ **denotes some number;**

  (ii) $1/0$ **denotes some kind of 'error value'.**

- **It seems at first sight that adopting (ii) 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

  - **only property assumed about** m/n **is:** $\neg(n = 0) \quad \Rightarrow \quad (m/n) \times n = m$

  - **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** $n \times 0 = 0$

`WHILE`-rule for Total Correctness (i)

- `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 `WHILE`-rule for total correctness is
  - to prove `WHILE` S `DO` C terminates
  - show that some non-negative quantity decreases on each iteration of C
  - this decreasing quantity is called a **variant**

$\texttt{WHILE}$-Rule for Total Correctness (ii)

- **In the rule below, the variant is E, and the fact that it decreases is specified with an auxiliary variable n**

- **The hypothesis $\vdash\ P \wedge S \Rightarrow E \geq 0$ ensures the variant is non-negative**

$\texttt{WHILE}$-rule for total correctness

$$\frac{\vdash\ [P \wedge S \wedge (E = n)]\ C\ [P \wedge (E < n)], \qquad \vdash\ P \wedge S \Rightarrow E \geq 0}{\vdash\ [P]\ \texttt{WHILE}\ S\ \texttt{DO}\ C\ [P \wedge \neg S]}$$

where E is an integer-valued expression
and n is an identifier not occurring in P, C, S or E.

Example

- **We show**

$\vdash$ $[\mathtt{Y} > 0]$ `WHILE Y≤R DO BEGIN R:=R-Y; Q:=Q+1 END` $[\mathtt{T}]$

- **Take**

$$
\begin{array}{lcl}
P & = & \mathtt{Y} > \mathtt{0} \\
S & = & \mathtt{Y} \leq \mathtt{R} \\
E & = & \mathtt{R} \\
C & = & \mathtt{BEGIN\ R:=R-Y\ Q:=Q+1\ END}
\end{array}
$$

- **We want to show** $\vdash$ $[P]$ `WHILE` S `DO` C $[\mathtt{T}]$

- **By the `WHILE`-rule for total correctness it is sufficient to show**

(i) $\vdash$ $[P \wedge S \wedge (E = \mathtt{n})]\ C\ [P \wedge (E < \mathtt{n})]$

(ii) $\vdash$ $P \wedge S \Rightarrow E \geq 0$

Example Continued

- **We want to show**

  (i) $\vdash\ [P \land S \land (E = \mathtt{n})]\ C\ [P \land (E < \mathtt{n})]$

  (ii) $\vdash\ P \land S \Rightarrow E \geq 0$

- **The first of these, (i), can be proved by establishing**

  $\vdash\ \{P \land S \land (E = \mathtt{n})\}\ C\ \{P \land (E < \mathtt{n})\}$

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

Example Continued

- **The verification condition for**

  $$\vdash \ \{P \wedge S \wedge (E = \mathtt{n})\} \ C \ \{P \wedge (E < \mathtt{n})\}$$

  is

  $$\begin{array}{l} \mathtt{Y} > 0 \ \wedge \ \mathtt{Y} \leq \mathtt{R} \ \wedge \ \mathtt{R} = \mathtt{n} \quad \Rightarrow \\ \quad (\mathtt{Y} > 0 \ \wedge \ \mathtt{R} < \mathtt{n})[\mathtt{Q}{+}1/\mathtt{Q}][\mathtt{R}{-}\mathtt{Y}/\mathtt{R}] \end{array}$$

  i.e.

  $$\mathtt{Y} > 0 \ \wedge \ \mathtt{Y} \leq \mathtt{R} \ \wedge \ \mathtt{R} = \mathtt{n} \quad \Rightarrow \quad \mathtt{Y} > 0 \ \wedge \ \mathtt{R}{-}\mathtt{Y} < \mathtt{n}$$

  which follows from the laws of arithmetic

- **The second subgoal, (ii), is just**

  $$\vdash \ \mathtt{Y} > 0 \ \wedge \ \mathtt{Y} \leq \mathtt{R} \Rightarrow \mathtt{R} \geq 0$$

  which follows from the laws of arithmetic

Termination Specifications

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

  Total correctness = Termination + Partial correctness

- **This informal equation can be represented by the following two formal rule of inferences**

$$\frac{\vdash\ \{P\}\ C\ \{Q\}, \qquad \vdash\ [P]\ C\ [\mathtt{T}]}{\vdash\ [P]\ C\ [Q]}$$

$$\frac{\vdash\ [P]\ C\ [Q]}{\vdash\ \{P\}\ C\ \{Q\}, \qquad \vdash\ [P]\ C\ [\mathtt{T}]}$$

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

Derived Skip Rule

$$\frac{\vdash\ P \Rightarrow Q}{\vdash\ [P]\ \texttt{SKIP}\ [Q]}$$

- **Proof:**

1. $\vdash\ P \Rightarrow Q$ By assumption.
2. $\vdash\ [Q]\ \texttt{SKIP}\ [Q]$ By the skip axiom for total correctness.
3. $\vdash\ [P]\ \texttt{SKIP}\ [Q]$ By precondition strengthening with 1 and 2.

- **Derivation Tree**

$$\dfrac{\vdash\ P \Rightarrow Q \qquad \dfrac{}{\vdash\ [Q]\ \texttt{SKIP}\ [Q]}\,SKP}{\vdash\ [P]\ \texttt{SKIP}\ [Q]}\,PRE$$

The Derived While Rule

- **Derived `WHILE`-rule needs to handle the variant**

`WHILE`-rule for total correctness

$$\frac{\begin{array}{c}\vdash\ P \Rightarrow R \\ \vdash\ R \wedge S \Rightarrow E \geq 0 \\ \vdash\ R \wedge \neg S\ \Rightarrow Q \\ \vdash\ [R \wedge S \wedge (E = n)]\ C\ [R \wedge (E < n)]\end{array}}{\vdash\ [P]\ \texttt{WHILE}\ S\ \texttt{DO}\ C\ [Q]}$$

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**
- **VCs generation algorithm same as for partial correctness**

`WHILE` Annotation

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

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

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**

WHILE VCs

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

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

WHILE-commands

The verification conditions generated from

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

are

(i) $P \ \Rightarrow\ R$

(ii) $R\ \wedge\ \neg S\ \Rightarrow\ Q$

(iii) $R\ \wedge\ S\ \Rightarrow\ E \geq 0$

(iv) the verification conditions generated by

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

where n is a variable not occurring in P, R, E, C, S or Q.

Example

- **The verification conditions for**

```
[R=X ∧ Q=0]
 WHILE Y≤R DO {X=R+Y×Q}[R]
   BEGIN R:=R-Y; Q=Q+1 END
[X = R+(Y×Q) ∧ R<Y]
```

are:

(i) `R=X ∧ Q=0 ⇒ (X = R+(Y×Q))`

(ii) `X = R+Y×Q ∧ ¬(Y≤R) ⇒ (X = R+(Y×Q) ∧ R<Y)`

(iii) `X = R+Y×Q ∧ Y≤R ⇒ R≥0`

together with the verification condition for

```
[X = R+(Y×Q) ∧ (Y≤R) ∧ (R=n)]
 BEGIN R:=R-Y; Q:=Q+1 END
[X=R+(Y×Q) ∧ (R<n)]
```

Example Continued

- **The single verification condition for**

  ```
  [X = R+(Y×Q) ∧ (Y≤R) ∧ (R=n)]
   BEGIN R:=R-Y; Q:=Q+1 END
  [X=R+(Y×Q) ∧ (R<n)]
  ```

  is

  (iv) `X = R+(Y×Q) ∧ (Y≤R) ∧ (R=n) ⇒`
  `X = (R-Y)+(Y×(Q+1)) ∧ ((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**

Summary

- We have given rules for total correctness
- They are similar to those for partial correctness
- The main difference is in the `WHILE`-rule
  - because `WHILE` commands are the only ones that can fail to terminate
- Must prove some positive expression is decreased by the loop body
- Derived rules and VC generation rules for total correctness easily extended to total correctness