Proving Programs Terminate

Total Correctness Specification

- So far our discussion has been concerned with parti
  - what about termination
- A total correctness specification $[P]\ C\ [Q]$ is true if
  - whenever C is executed in a state satisfying P, then the execution of C terminates
  - after C terminates Q holds
- Except for the `WHILE`-rule, all the axioms and rules are sound for total correctness as well as partial co

Termination of WHILE-Commands

- WHILE-commands are the only commands that migh
- Consider now the following proof

1. $\vdash \{\mathtt{T}\}\ \mathtt{SKIP}\ \{\mathtt{T}\}$ (SKIP ax
2. $\vdash \{\mathtt{T} \wedge \mathtt{T}\}\ \mathtt{SKIP}\ \{\mathtt{T}\}$ (precondition strengthe
3. $\vdash \{\mathtt{T}\}\ \mathtt{WHILE\ T\ DO\ SKIP}\ \{\mathtt{T} \wedge \neg\mathtt{T}\}$ (2 and the WHILE-

- If the WHILE-rule worked for total correctness, then t

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

- Thus the WHILE-rule is unsound for total correctnes

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 correct**
  assignment commands *always* **terminate**

- **So all function applications in expressions must ter**

- **This might not be the case if functions could be def**

- **Consider** $X := fact(-1)$**, where** $fact(n)$ **is defined rec**

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

Error Termination

- **We assume erroneous expressions like $1/0$ don't cau**
- **Most programming languages will cause an error sto 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 `X := 1/0` halts in a state in which `X`**
- **This assumes that $1/0$ denotes some value that `X` ca**

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**
  - **this makes it tricky to see what arithmetical laws should**
  - **is** $(1/0) \times 0$ **equal to** 0 **or to some 'error value'?**
  - **if the latter, then it is no longer the case that** $\forall n.\ n \times 0 =$

- **It is possible to make everything work with undefin values, but the resultant theory is a bit messy**

Example

- **We assume that arithmetic expressions** *always* **deno**
- **In some cases exactly what the number is will be n**
  - **for example, we will assume that** m/n **denotes a number f**
  - **only assume:** $\neg(n = 0) \quad \Rightarrow \quad (m/n) \times n = m$
  - **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** $\forall n.\ n \times 0 = 0$
- **People still argue about this – e.g. advocate "three**

WHILE-rule for Total Correctness (i)

- WHILE-commands are the only commands in our litt
  can cause non-termination
  - they are thus the only kind of command with a non-trivia
- The idea behind the WHILE-rule for total correctness
  - to prove WHILE S DO C terminates
  - show that some non-negative quantity decreases on each
  - this decreasing quantity is called a variant

WHILE-Rule for Total Correctness (ii)

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

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

WHILE-rule for total correctness

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

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

Example

- **We show**

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

- **Take**

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

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

- **By the WHILE-rule for total correctness it is sufficien**

(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 (1)

- **From previous slide:**

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

- **We want to show**

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

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

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

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

- **Then using the total correctness rule for non-loopi**

Example Continued (2)

- **From previous slide:**

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

- **The verification condition for** $\{P \wedge S \wedge (E = \texttt{n})\}\ C\ \{P$

$$\texttt{Y} > 0 \ \wedge\ \texttt{Y} \leq \texttt{R} \ \wedge\ \texttt{R} = \texttt{n} \ \Rightarrow\ (\texttt{Y} > 0 \ \wedge\ \texttt{R} < \texttt{n})\,[\texttt{Q}+1/\texttt{Q}]\,[\texttt{R}-\texttt{Y}/\texttt{R}]$$

  i.e. $\texttt{Y} > 0 \ \wedge\ \texttt{Y} \leq \texttt{R} \ \wedge\ \texttt{R} = \texttt{n} \ \Rightarrow\ \texttt{Y} > 0 \ \wedge\ \texttt{R}-\texttt{Y} < \texttt{n}$

  which follows from the laws of arithmetic

- **The second subgoal, (ii), is just** $\vdash\ \texttt{Y} > 0 \ \wedge\ \texttt{Y} \leq \texttt{R} \Rightarrow$

Termination Specifications

- **The relation between partial and total correctne given by the equation**

$$\textit{Total correctness} = \textit{Termination} + \textit{Partial correctness}$$

- **This informal equation can be represented by the fol 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 deri
  way as for partial correctness
  - the rules are the same up to the brackets used
  - same derivations with total correctness rules replacing part

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**

$$\dfrac{\dfrac{\vdash\ P \Rightarrow Q \quad \overline{\vdash\ [Q]\ \texttt{SKIP}\ [Q]}\ SKP}{\vdash\ [P]\ \texttt{SKIP}\ [Q]}\ PRE}$$

The Derived While Rule

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

Derived `WHILE`-rule for total correctness

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

VCs for Termination

- Verification conditions are easily extended to total
- To generate total correctness verification condit commands, it is necessary to add a variant as an a dition to an invariant
- Variant added directly after the invariant, in squar
- No other extra annotations are needed for total co
- VCs generation algorithm same as for partial corre

WHILE Annotation

- A correctly annotated total correctness specificat
  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-nega
  that decreases each time around the WHILE loop

- The other annotations, which are enclosed in cur
  meant to be conditions that are true whenever contr
  (as before)

WHILE VCs

- **A correctly annotated specification of a WHILE-comma**

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

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**

$$[R{=}X \wedge Q{=}0]$$
$$\text{WHILE } Y \leq R \text{ DO } \{X{=}R{+}Y{\times}Q\}[R]$$
$$\text{BEGIN } R{:=}R{-}Y;\ Q{=}Q{+}1 \text{ END}$$
$$[X = R{+}(Y{\times}Q) \wedge R{<}Y]$$

are:

(i) $R{=}X \wedge Q{=}0 \Rightarrow (X = R{+}(Y{\times}Q))$

(ii) $X = R{+}Y{\times}Q \wedge \neg(Y \leq R) \Rightarrow (X = R{+}(Y{\times}Q) \wedge R{<}Y)$

(iii) $X = R{+}Y{\times}Q \wedge Y \leq R \Rightarrow R \geq 0$

together with the verification condition for

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

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 a 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 t
- Must prove a non-negative expression is decreased b
- Derived rules and VC generation rules for partial co extended to total correctness
- Interesting stuff on the web
  - http://www.crunchgear.com/2008/12/31/zune-bug-expla
  - http://research.microsoft.com/TERMINATOR