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

- Except for 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. $\vdash$ {T} X := X {T} (assignment axiom)
2. $\vdash$ {T $\wedge$ T} X := X {T} (precondition strengthening)
3. $\vdash$ {T} WHILE T DO X := X {T $\wedge$ $\neg$T} (2 and the WHILE-rule)

- If the WHILE-rule worked for total correctness, then this would show:

$$\vdash\ [\mathtt{T}]\ \mathtt{WHILE\ T\ DO\ X\ :=\ X}\ [\mathtt{T} \wedge \neg\mathtt{T}]$$

- 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 rule**
  - **Sequencing 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** X := $fact(-1)$**, where** $fact(n)$ **is defined recursively:**

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

Error Termination ✓

- **We assume erroneous expressions like** $1/0$ **don't cause problems**
- **Most programming languages will raise an error on division by zero**
- **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** $\forall n.\ n \times 0 = 0$ **is valid**

- **It is possible to make everything work with undefined and/or error values, but the resultant theory is a bit messy**

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**

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**

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.

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 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**

- **Variant added directly after the invariant, in square brackets**

- **No other extra annotations are needed for total correctness**

- **VCs for `WHILE`-free code 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 (as before)**

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.

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 a non-negative expression is decreased by the loop body**
- **Derived rules and VC generation rules for partial correctness easily extended to total correctness**
- **Interesting stuff on the web**
  - **http://www.crunchgear.com/2008/12/31/zune-bug-explained-in-detail**
  - **http://research.microsoft.com/en-us/projects/t2/**

Soundness and completeness of Hoare logic ✓

- **Review of first-order logic**
  - syntax: languages, function symbols, predicate symbols, terms, formulae
  - semantics: interpretations, valuations
  - soundness and completeness

- **Formal semantics of Hoare triples**
  - preconditions and postconditions as terms
  - semantics of commands
  - soundness of Hoare axioms and rules
  - completeness and relative completeness

Semantics: terms and formulae ✓

- **Assume: language $\mathcal{L}$, interpretation $\mathcal{I} = (D, I)$, valuation $s \in \mathit{Var} \to D$**

- **Define $\texttt{Esem}\ E\ s \in D$ by:**
  - **if** $E \in \mathit{Var}$ **then** $\texttt{Esem}\ E\ s = s(E)$
  - **if** $E = f$**, where** f **a function symbol of arity** 0**, then** $\texttt{Esem}\ E\ s = I[f]$
  - **if** $E = f(E_1, \ldots, E_n)$**, then** $\texttt{Esem}\ E\ s = I[f](\texttt{Esem}\ E_1\ s, \ldots, \texttt{Esem}\ E_n\ s)$

- **Define $\texttt{Ssem}\ S\ s \in \mathit{Bool}$ by:**
  - **if** $S = p$**, where** p **a predicate symbol of arity** 0**, then** $\texttt{Ssem}\ S\ s = I[p]$
  - **if** $S = p(E_1, \ldots, E_n)$**, then** $\texttt{Ssem}\ S\ s = I[p](\texttt{Esem}\ E_1\ s, \ldots, \texttt{Esem}\ E_n\ s)$
  - $$\begin{array}{lcl} \texttt{Ssem}\ (\neg S)\ s & = & \neg(\texttt{Ssem}\ S\ s) \\ \texttt{Ssem}\ (S_1 \wedge S_2)\ s & = & (\texttt{Ssem}\ S_1\ s) \wedge (\texttt{Ssem}\ S_2\ s) \\ \texttt{Ssem}\ (S_1 \vee S_2)\ s & = & (\texttt{Ssem}\ S_1\ s) \vee (\texttt{Ssem}\ S_2\ s) \\ \texttt{Ssem}\ (S_1 \Rightarrow S_2)\ s & = & (\texttt{Ssem}\ S_1\ s) \Rightarrow (\texttt{Ssem}\ S_2\ s) \end{array}$$
  - $$\begin{array}{lcl} \texttt{Ssem}\ (\forall v.\ S)\ s & = & \textbf{\textit{if}}\ (\textbf{\textit{for all}}\ d \in D : \quad \texttt{Ssem}\ S\ (s\texttt{[}d\texttt{/}v\texttt{]}) = \mathit{true})\ \textbf{\textit{then}}\ \mathit{true}\ \textbf{\textit{else}}\ \mathit{false} \\ \texttt{Ssem}\ (\exists v.\ S)\ s & = & \textbf{\textit{if}}\ (\textbf{\textit{for some}}\ d \in D : \texttt{Ssem}\ S\ (s\texttt{[}d\texttt{/}v\texttt{]}) = \mathit{true})\ \textbf{\textit{then}}\ \mathit{true}\ \textbf{\textit{else}}\ \mathit{false} \end{array}$$

- **Note: will just say "$\texttt{Ssem}\ S\ s$" to mean that "$\texttt{Ssem}\ S\ s = \mathit{true}$"**

Satisfiability, validity and completeness ✓

- **Recall that a language $\mathcal{L}$ specifies predicate and function symbols**
- S **is** *satisfiable* **iff for some interpretation of $\mathcal{L}$ and s:** `Ssem` $S\ s = \mathit{true}$
- S **is** *valid* **iff for all interpretations of $\mathcal{L}$ and all s:** `Ssem` $S\ s = \mathit{true}$
- **Notation:** $\models S$ **means** S **is valid**
- **Deductive system for first-order logic specifies** $\vdash S$ **– i.e.** S **is provable**
- **Soundness:** **if** $\vdash S$ **then** $\models S$ **(easy induction on length of proof)**
- **Completeness:** **if** $\models S$ **then** $\vdash S$ **(Gödel 1929)**

Sentences, Theories

- **A** *sentence* **is a statement with** *no free variables*
  - **truth or falsity of sentences solely determined by interpretation**
  - **if** S **is a sentence then** `Ssem` $S\ s_1 =$ `Ssem` $S\ s_2$ **for all** s_1, s_2
- **A** *theory* **is a set of sentences**
  - Γ **will range over sets of sentences**
- $\Gamma \vdash S$ **means** S **can be deduced from** Γ **using first-order logic**
- Γ **is** *consistent* **iff there is no** S **such that** $\Gamma \vdash S$ **and** $\Gamma \vdash \neg S$
- $\Gamma \models_{\mathcal{I}} S$ **means** S **true if** $\mathcal{I}$ **makes all of** Γ **true**
- $\Gamma \models S$ **means** $\Gamma \models_{\mathcal{I}} S$ **true for all** $\mathcal{I}$
- **Soundness and Completeness:** $\Gamma \models S$ **iff** $\Gamma \vdash S$

Gödel's incompleteness theorem ✓

- $\mathcal{L}_{\mathbf{PA}}$ is the language of Peano Arithmetic
- $\mathcal{I}_{\mathbf{PA}}$ is the *standard interpretation* of arithmetic
- $\models_{\mathcal{I}_{\mathbf{PA}}} S$ means S is true in $\mathcal{I}_{\mathbf{PA}}$
- PA is the first-order theory of Peano Arithmetic
- There exists a sentence G of $\mathcal{L}_{\mathbf{PA}}$ and neither PA $\vdash G$ nor PA $\vdash \neg G$
  - Gödel's first incompleteness theorem (1930)
  - as G is a sentence either $\models_{\mathcal{I}_{\mathbf{PA}}} G$ or $\models_{\mathcal{I}_{\mathbf{PA}}} \neg G$
  - so there is a sentences, G_T say, true in $\mathcal{I}_{\mathbf{PA}}$ but can't be proved from PA
  - i.e. $\models_{\mathcal{I}_{\mathbf{PA}}} G_T$ but not PA $\vdash G_T$

Semantics of Hoare triples ✓

- **Recall that $\{P\}\ C\ \{Q\}$ is true if**
  - **whenever C is executed in a state satisfying P**
  - **and *if* the execution of C terminates**
  - **then C terminates in a state satisfying Q**

- **P and Q are first-order statements**

- **Will formalise semantics of $\{P\}\ C\ \{Q\}$ to express:**
  - **whenever C is executed in a state s_1 such that $\texttt{Ssem}\ P\ s_1$**
  - **and *if* the execution of C starting in s_1 terminates**
  - **then C terminates in a state s_2 such that $\texttt{Ssem}\ Q\ s_2 = \mathit{true}$**

- **Need to define "C starts in s_1 and terminates in s_2"**
  - **this is the semantics of commands**
  - **will define $\texttt{Csem}\ C\ s_1\ s_2$ to mean if C starts in s_1 then it can terminate in s_2**

- **Semantics of $\{P\}\ C\ \{Q\}$ is $\texttt{Hsem}\ P\ C\ Q$ where:**

$$\texttt{Hsem}\ P\ C\ Q = \forall s_1\ s_2.\ \texttt{Ssem}\ P\ s_1 \wedge \texttt{Csem}\ C\ s_1\ s_2 \Rightarrow \texttt{Ssem}\ Q\ s_2$$

- **Sometimes write $\models \{P\}\ C\ \{Q\}$ to mean $\texttt{Hsem}\ P\ C\ Q$**