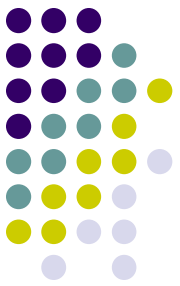


Formal methods

Total Correctness

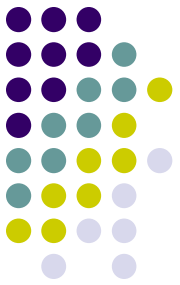




Total Correctness

- We introduced a stronger kind of specification:
a total correctness specification
- 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

Termination of WHILE command

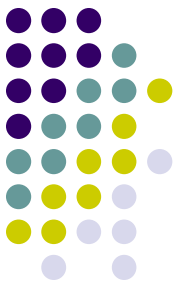


- With the exception of the WHILE-rule, all the axioms and rules described so far are sound for total correctness as well as partial correctness
- If the WHILE-rule were true for total correctness, then the proof above would show that

$\vdash [T] \text{ WHILE } T \text{ DO } X := 0 [T \wedge \neg T]$ because

The WHILE-rule

$$\frac{\vdash \{P \wedge S\} C \{P\}}{\vdash \{P\} \text{ WHILE } S \text{ DO } C \{P \wedge \neg S\}}$$



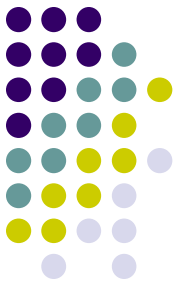
Rules for Non-looping Commands

- Replace { and } by [and], respectively, in:
 - Assignment axiom (see below)
 - 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

Termination



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

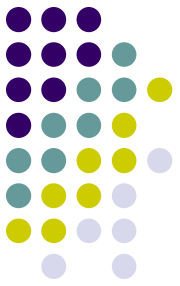
$$\begin{aligned} \textit{Total correctness} &= \\ &\textit{Termination} + \textit{Partial correctness} \end{aligned}$$

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

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

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

Total Correctness of Assignment

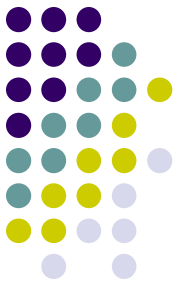


- Assignment axiom for total correctness

$$\vdash [P[E/V]] V := E [P]$$

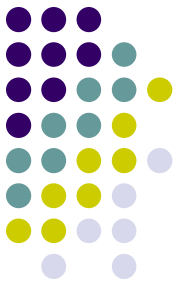
- Note that the assignment axiom for total correctness states that assignment commands *always* terminate

Total Correctness of Assignment



- This implicitly assumes that all function applications in expressions 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

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



Error Termination

- It is also assumed that erroneous expressions like $1/0$ do not cause problems

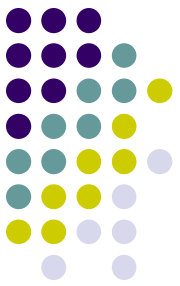
- Most programming languages will cause an error stop when division by zero is encountered

1

- In our logic it follows that

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

- i.e. the assignment $X := 1/0$ always halts in a state in which the condition $X = 1/0$ holds
- This assumes that $1/0$ denotes some value that X can have

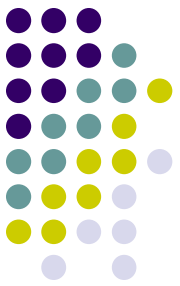


Two possibilities

- 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?



Definition of Arithmetics

- 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

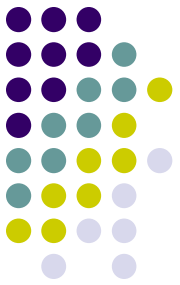
- The only property of “/” that will be assumed is:

$$\neg(n = 0) \Rightarrow (m/n) \times n = m$$

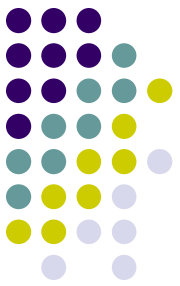
- It is not possible to deduce anything about $m/0$ from this, in particular it is not possible to deduce that $(m/0) \times 0 = 0$

- but $(m/0) \times 0 = 0$ does follow from $n \times 0 = 0$

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



WHILE-rule for total correctness

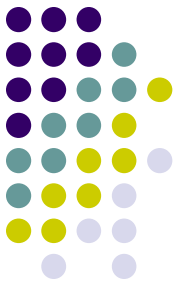
- In the rule below, the variant is E , and the fact that it decreases is specified with an auxiliary variable n
- An extra hypothesis, $\vdash P \wedge S \Rightarrow E \geq 0$, ensures the variant is non-negative

WHILE-rule for total correctness

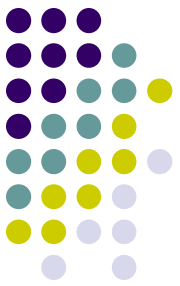
$$\frac{\vdash [P \wedge S \wedge (E = n)] C [P \wedge (E < n)], \quad \vdash P \wedge S \Rightarrow E \geq 0}{\vdash [P] \text{ WHILE } S \text{ DO } C [P \wedge \neg S]}$$

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

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



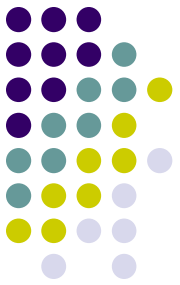
Derived WHILE-rule

- The derived While rule is slightly different to the partial correctness version
 - The extra information about the variant is needed

WHILE-rule for total correctness

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

Example



- We show

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

- Take

$P = Y > 0$

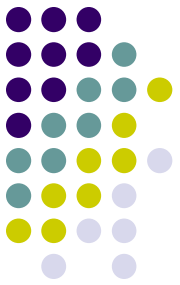
$S = Y \leq R$

$E = R$

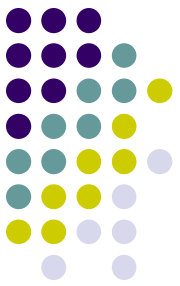
$C = \text{BEGIN } R := R - Y \text{ } Q := Q + 1 \text{ END}$

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

Verification Conditions



- The idea of verification conditions is easily extended to deal with total correctness
- To generate 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



WHILE annotation

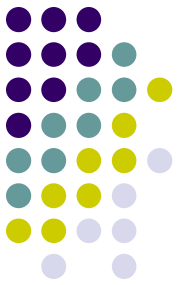
- A correctly annotated total correctness specification of a WHILE-command thus has the form

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

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

Verification Conditions



The verification conditions generated from

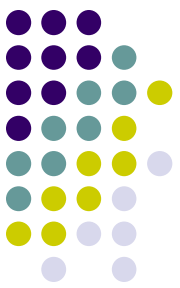
$$[P] \text{ WHILE } S \text{ DO } \{R\}[E] C [Q]$$

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]
```

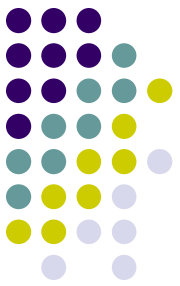
(i) $R=X \wedge Q=0 \Rightarrow (X = R+(Y \times Q))$

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



Example

- The single verification condition for

$$[X = R + (Y \times Q) \wedge (Y \leq R) \wedge (R = n)]$$

BEGIN R := R - Y; Q := Q + 1 END

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

$$(iv) \quad \begin{array}{l} X = R + (Y \times Q) \wedge (Y \leq R) \wedge (R = n) \Rightarrow \\ X = (R - Y) + (Y \times (Q + 1)) \wedge ((R - Y) < n) \end{array}$$

- 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