COMS22202: 2015/16

# Language Engineering

**Dr Oliver Ray**
**(csxor@Bristol.ac.uk)**

**Department of Computer Science**
**University of Bristol**

**Thursday 10th March, 2016**

# Total Correctness Schemata: cf. p192

$$[\,P\,]\ \mathbf{skip}\ [\,P\,]$$ SKIP

$$[\,P(a)\,]\ x := a\ [\,P(x)\,]$$ ASS

$$\frac{[\,P\,]\ S_1\ [\,Q\,] \qquad [\,Q\,]\ S_2\ [\,R\,]}{[\,P\,]\ S_1\ ;\ S_2\ [\,R\,]}$$ SEQ

$$\frac{[\,P \wedge b\,]\ S_1\ [\,Q\,] \qquad [P \wedge \neg b\,]\ S_2\ [\,Q\,]}{[\,P\,]\ \mathbf{if}\ \ b\ \mathbf{then}\ \ S_1\ \ \mathbf{else}\ \ S_2\ [\,Q\,]}$$ COND

$$\frac{[\,P(z+1)\,]\ S\ [\,P(z)\,]}{[\,\exists_{z \in N}.P(z)\,]\ \mathbf{while}\ b\ \mathbf{do}\ S\ [\,P(0)\,]} \quad : \quad \begin{array}{l} \models \forall_{z \in N}.P(z+1) \rightarrow b \\ \models \ \ p(0) \rightarrow \neg b \end{array}$$ LOOP

$$\frac{[\,P'\,]\ \ S\ \ [\,Q'\,]}{[\,P\,]\ \ S\ \ [\,Q\,]} \quad : \quad \begin{array}{l} P\ \ \models\ \ P' \\ Q'\ \models\ \ Q \end{array}$$ CONS

# Axiom Schema for While: cf. p192

$$\frac{[\,P(z+1)\,]\ S\ [\,P(z)\,]}{[\,\exists_{z\in N}.P(z)\,]\ \text{while } b \text{ do } S\ [\,P(0)\,]} \quad : \quad \begin{array}{l} \models \forall_{z\in N}.P(z+1) \rightarrow b \\ \models\ \ p(0) \rightarrow \neg b \end{array}$$

- The loop rule is the only significant change required when moving from the partial to the total semantics as we must show non-termination is avoided
- We prefer the more common notation [P] S [Q] to the notation used in the book to represent total correctness assertions {P} S {⇓Q}
- The loop invariant P is now parameterised on a natural number z (called a *variant*) representing the number of times the loop needs to be unfolded
- If the loop is run in a state that requires some finite number z of iterations, then it will terminate in state that requires 0 iterations
- If the loop still needs to execute then P(z+1) must entail b; but when it is about to terminate then P(0) must entail ¬b (and we need to prove these)
- The premise effectively states that number of iterations remaining must decrease by one every time the body executes
- In the factorial example it is easy to see x=z+1 (as the loop must run x-1 times) so this must be added to the invariant

# Example: Total Correctness Proof Tree

$$\dfrac{\dfrac{\dfrac{}{\begin{bmatrix} n!=(y*x)(x-1)! \\ x-1>0 \\ x-1=z+1 \end{bmatrix} \; y:=y*x \; \begin{bmatrix} n!=y(x-1)! \\ x-1>0 \\ x-1=z+1 \end{bmatrix}}\,\text{ASS}}{\begin{bmatrix} n!=yx! \\ x-1>0 \\ x=(z+1)+1 \end{bmatrix} \; y:=y*x \; \begin{bmatrix} n!=y(x-1)! \\ x-1>0 \\ x-1=z+1 \end{bmatrix}}\,\text{CONS-2} \qquad \dfrac{}{\begin{bmatrix} n!=y(x-1)! \\ x-1>0 \\ x-1=z+1 \end{bmatrix} \; x:=x-1 \; \begin{bmatrix} n!=yx! \\ x>0 \\ x=z+1 \end{bmatrix}}\,\text{ASS}}{[n!=yx! \;\&\; x>0 \;\&\; x=(z+1)+1] \; y:=y*x;\; x:=x-1 \; [n!=yx! \;\&\; x>0 \;\&\; x=z+1] \longleftarrow \boxed{P(z)}}\,\text{SEQ}$$

$$\dfrac{\dfrac{[n!=yx! \;\&\; x>0 \;\&\; x=(z+1)+1] \; y:=y*x;\; x:=x-1 \; [n!=yx! \;\&\; x>0 \;\&\; x=z+1]}{\begin{bmatrix} \exists_{z\in N}.\; n!=yx! \\ x>0 \\ x=z+1 \end{bmatrix} \; \text{while } \neg(x=1) \text{ do } (y:=y*x;\; x:=x-1) \; \begin{bmatrix} n!=yx! \\ x>0 \\ x=0+1 \end{bmatrix}}\,\text{LOOP-1}}{[y=1 \;\&\; x=n \;\&\; n>0] \;\; \text{while } \neg(x=1) \text{ do } (y:=y*x;\; x:=x-1) \;\; [y=n!]}\,\text{CONS-1}$$

# Example: Proof Obligations

## (A) CONS-1-Postcondition (OK)

$n! = yx! \;\&\; x>0 \;\&\; x=0+1 \models y=n!$

| | | |
|---|---|---|
| 1. | $n!=yx!$ | given |
| 2. | $x>0$ | given |
| 3. | $x=0+1$ | given |
| 4. | $x=1$ | from 3 by defn of add |
| 5. | $x!=1!=1$ | by defn of fac using 4 |
| 6. | $n!=y1$ | sub 5 in rhs of 1 |
| 7. | $n!=y$ | by defn of mult |
| 8. | $y=n!$ | by symmetry of = |

## (B) CONS-1-Precondition (OK)

$y=1 \;\&\; x=n \;\&\; n>0 \models \exists z.\, n!=yx \;\&\; x>0 \;\&\; x=z+1$

| | | |
|---|---|---|
| 1. | $y=1$ | given |
| 2. | $x=n$ | given |
| 3. | $n>0$ | given |
| 4. | $x!=n!$ | fac lhs & rhs of 2 |
| 5. | $x!=1x!=yx!$ | by 1 using defn of mult |
| 6. | $n!=yx!$ | sub 5 in lhs of 5 by sym = |
| 7. | $x>0$ | sub 2 in lhs of 3 |
| 8. | $x=(x-1)+1$ | by defn of + and - |
| 9. | $\exists z.\, n!=yx \;\&\; x>0 \;\&\; x=z+1$ | by 6,7,8 using $z=x-1$ |

## (C) CONS-2-Postcondition (OK)

$n!=y(x-1)! \;\&\; x-1>0 \quad \models$

$n!=y(x-1)! \;\&\; x-1>0$

Trivial (by reflexivity of |=)

## (D) CONS-2-Precondition (OK)

$n!=yx! \qquad \&\; x-1>0 \quad \&\; x=(z+1)+1 \quad \models$

$n!=(y*x)(x-1)! \;\&\; x-1>0 \quad \&\; x-1=z+1$

Trivial (by defn of fac using $x\neq 0$)

## (E) LOOP-1-Induction-Step (OK)

$\models \forall_{z\in N}.\, n!=yx! \;\&\; x>0 \;\&\; x=z+1+1 \rightarrow \neg(x=1)$

Trivial (as $x=z+2$ and $z\geq 0$ mean $x>2$)

## (F) LOOP-1-Base-Case (OK)

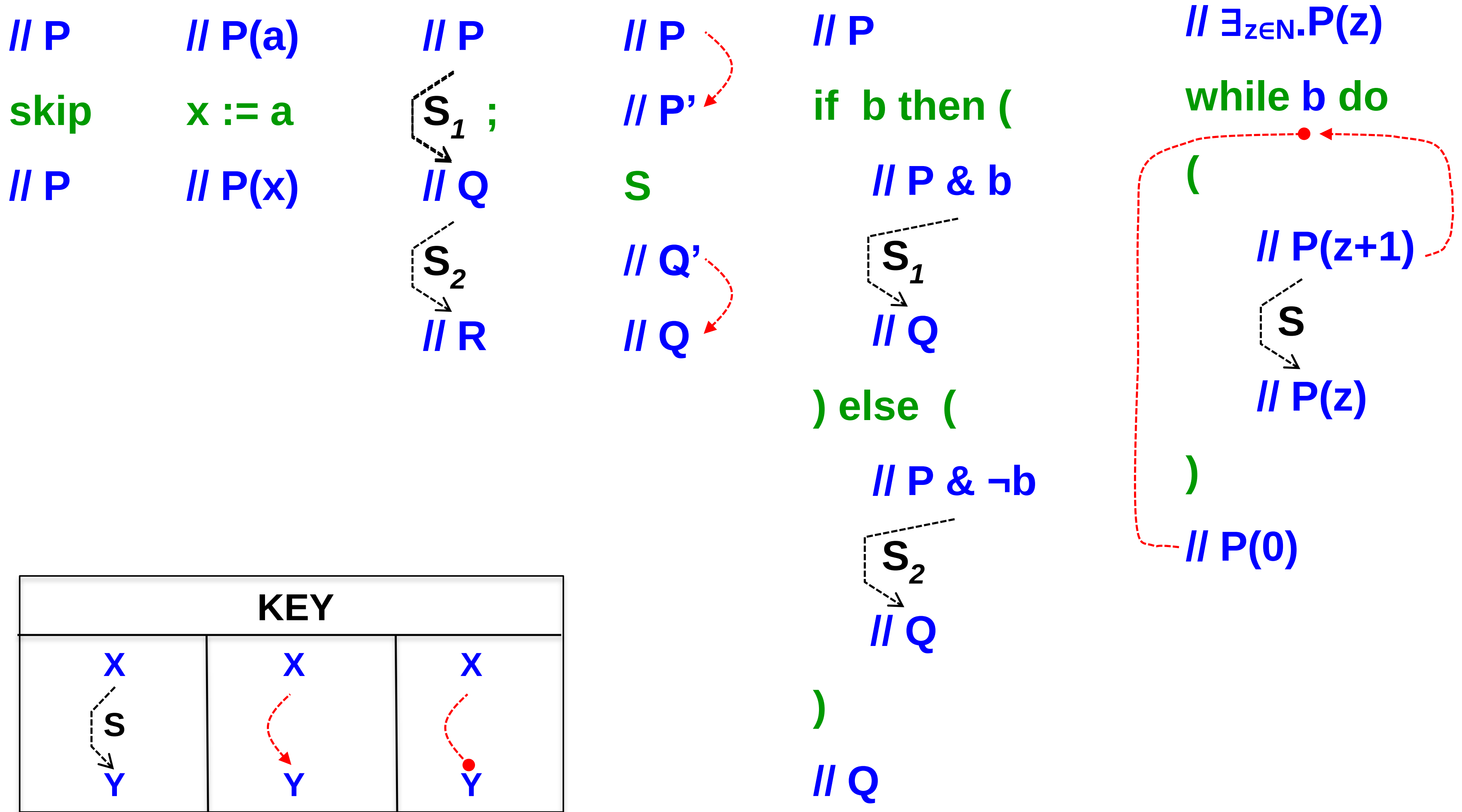
$\models n!=yx! \;\&\; x>0 \;\&\; x=0+1 \rightarrow \neg\neg(x=1)$

Trivial (by ¬¬elim using $x=0+1=1$ )

## Tableau Representation

- The tree-based representation of axiomatic proofs closely follows the structure of the rules, but it is also very verbose as program fragments are repeated every time the tree branches
- It is always possible to translate a proof tree into a linear representation in which program statements are written exactly once (as they would be in a source file) and marked up with axiomatic assertions (like comments)
- The following slide presents the (informal) rules for building such tableau (which directly mirror the axiomatic correctness rules but avoid branching by placing the proofs of premises within the conclusion assertion)
- Although you are always free to use proof trees, it is worth bearing in mind that tableau proofs are much easier to write and alter (especially in exam situations) and are used in the solutions of past exam papers
- Don’t forget that the proof obligations are the same whichever format is used and at least half of the marks for an axiomatic proof will be given for the entailment proofs
- The slide after next shows how the earlier proof tree for the factorial program would be written in tableau format, and indicates which lines the earlier entailment proofs correspond to

# Tableau Construction



```
// P
skip
// P
```

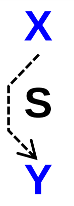
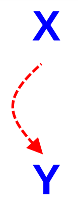
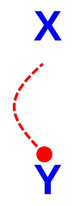
```
// P(a)
x := a
// P(x)
```

```
// P
S_1 ;
// Q
S_2
// R
```

```
// P
// P'
S
// Q'
// Q
```

```
// P
if  b then (
      // P & b
      S_1
      // Q
) else  (
      // P & ¬b
      S_2
      // Q
)
// Q
```

```
// ∃z∈N.P(z)
while b do
(
      // P(z+1)
      S
      // P(z)
)
// P(0)
```

| KEY | | |
|---|---|---|
|  X S Y |  X Y |  X Y |
| $\vdash$ [X] S [Y] | $\models X \rightarrow Y$ | $\models X \rightarrow \neg Y$ |

# Example: Tableau Format

```
// $y=1$ & $x=n$ & $n>0$
                                                  B
// $\exists_{z \in N}$ . $n!=yx!$ & $x>0$ & $x=z+1$
while $\neg(x=1)$ do
(
         E
        // $n!=yx!$ & $x>0$ & $x=(z+1)+1$
                                                        D
        // $n!=(y*x)(x-1)!$ & $(x-1)>0$ & $(x-1)=z+1$
        $y:=y*x$ ;
F       // $n!=y(x-1)!$ & $(x-1)>0$ & $(x-1)=z+1$
        $x:=x-1$
        // $n!=yx!$ & $x>0$ & $x=z+1$      C
)
// $n!=yx!$ & $x>0$ & $x=0+1$
                                       A
// $y=n!$
```

## (Semantic) Equivalence: p182

Two programs $S_1$ and $S_2$ are (axiomatically) equivalent whenever

**{P}** $S_1$ **{Q}** if and only if **{P}** $S_2$ **{Q}** for all properties **P and Q**

Note: as we saw in class, a useful proof technique for showing such equivalences is reasoning about the shape of proof trees