Arrays and `FOR`-commands
Soundness and Completeness

Overview

- **All the axioms and rules given so far were quite straightforward**
  - **may have given a false sense of simplicity**
- **Hard to give rules for anything other than *very* simple constructs**
  - **an incentive for using simple languages**
- **We already saw with the assignment axiom that intuition over how to formulate a rule might be wrong**
  - **the assignment axiom can seem 'backwards'**
- **We now look at the remaing commands in our little language**
  - **array assignments**
  - **`FOR`-commands**

Array assignments

- **Syntax:** $V(E_1)$`:=`E_2
- **Semantics: the state is changed by assigning the value of the term E_2 to the E_1-th component of the array variable V**
- **Example:** `A(X+1) := A(X)+2`
  - **if the the value of `X` is x**
  - **and the value of the x-th component of `A` is n**
  - **then the value stored in the $(x+1)$-th component of `A` becomes $n+2$**

Naive Array Assignment Axiom Fails

- **The axiom**

  $\vdash \ \{P\texttt{[}E_2/A(E_1)\texttt{]}\}\ A(E_1) := E_2\ \{P\}$

  doesn't work

- **Take** $P \equiv$ '`X=Y` $\wedge$ `A(Y)=0`', $E_1 \equiv$ '`X`', $E_2 \equiv$ '`1`'
  - **since** `A(X)` **does not occur in** P
  - **it follows that** P`[1/A(X)] =` P
  - **hence the axiom yields:** $\vdash$ {`X=Y` $\wedge$ `A(Y)=0`} `A(X):=1` {`X=Y` $\wedge$ `A(Y)=0`}
  - **which is false since it implies:** $\vdash$ $\{\cdots\}$ `A(X):=1` {`A(X)=0`}

- **Must take into account possibility that changes to** `A(X)` **may change** `A(Y)`, `A(Z)` **etc**
  - **since** `X` **might equal** `Y`, `Z` **etc (i.e. aliasing)**

- **Related to the** *Frame Problem* **in AI**

Reasoning About Arrays

- **The naive array assignment axiom**

$$\vdash\ \{P[E_2/A(E_1)]\}\ A(E_1) := E_2\ \{P\}$$

does not work: changes to $A(X)$ may also change $A(Y)$, $A(Z)$, ...

- **The solution, due to Hoare, is to treat an array assignment**

$$A(E_1){:=}E_2$$

as an ordinary assignment

$$A := A\{E_1 \leftarrow E_2\}$$

where the term $A\{E_1 \leftarrow E_2\}$ denotes an array identical to A, except that the E_1-th component is changed to have the value E_2

Array Assignment axiom

- **Array assignment is a special case of ordinary assignment**

$$A\texttt{:=}A\{E_1\leftarrow E_2\}$$

- **Array assignment axiom just ordinary assignment axiom**

$$\vdash\ \{P\texttt{[}A\{E_1\leftarrow E_2\}/A\texttt{]}\}\ A\texttt{:=}A\{E_1\leftarrow E_2\}\ \{P\}$$

- **Thus:**

The array assignment axiom

$$\vdash\ \{P\texttt{[}A\{E_1\leftarrow E_2\}/A\texttt{]}\}\ A(E_1)\texttt{:=}E_2\ \{P\}$$

Where A is an array variable, E_1 is an integer valued expression, P is any statement and the notation $A\{E_1\leftarrow E_2\}$ denotes the array identical to A, except that $A(E_1) = E_2$.

Array Axioms

- **In order to reason about arrays, the following axioms, which define the meaning of the notation $A\{E_1 \leftarrow E_2\}$, are needed**

The array axioms

$$\vdash \; A\{E_1 \leftarrow E_2\}(E_1) \; = \; E_2$$

$$\vdash \; E_1 \neq E_3 \;\; \Rightarrow \;\; A\{E_1 \leftarrow E_2\}(E_3) \; = \; A(E_3)$$

- **Second of these is a** *Frame Axiom*

Example

- **We show**

$$\vdash \{\texttt{A(X)=x} \wedge \texttt{A(Y)=y}\}$$

```
R    := A(X);
A(X) := A(Y);
A(Y) := R
```

$$\{\texttt{A(X)=y} \wedge \texttt{A(Y)=x}\}$$

- **Working backwards using the array assignment axiom**

$$\vdash \{\texttt{(A}\{\texttt{Y}\leftarrow\texttt{R}\}\texttt{)(X)=y} \wedge \texttt{(A}\{\texttt{Y}\leftarrow\texttt{R}\}\texttt{)(Y)=x}\}$$

```
A(Y) := R
```

$$\{\texttt{A(X)=y} \wedge \texttt{A(Y)=x}\}$$

- **Array assignments are variable assignments of array values, so:**

$$\vdash \{\texttt{(A}\{\texttt{Y}\leftarrow\texttt{R}\}\texttt{)(X)=y} \wedge \texttt{(A}\{\texttt{Y}\leftarrow\texttt{R}\}\texttt{)(Y)=x}\}$$

$$\texttt{A := A}\{\texttt{Y}\leftarrow\texttt{R}\}$$

$$\{\texttt{A(X)=y} \wedge \texttt{A(Y)=x}\}$$

Example Continued (1)

- **Using**

  $\vdash$ A{Y←R}(Y) = R

- **It follows that**

  $\vdash$ {(A{Y←R})(X)=y ∧ R=x}
  A(Y) := R
  {A(X)=y ∧ A(Y)=x}

- **Continuing backwards**

  $\vdash$ {((A{X←A(Y)}){Y←R})(X)=y ∧ R=x}
  A(X) := A(Y)
  {(A{Y←R})(X)=y ∧ R=x}

- **Maybe more intuitive if the assignment is rewritten**

  $\vdash$ {((A{X←A(Y)}){Y←R})(X)=y ∧ R=x}
  A := A{X←A(Y)}
  {(A{Y←R})(X)=y ∧ R=x}

Example Continued (2)

- **Continuing backwards**

$$\vdash \{((\texttt{A}\{\texttt{X}\leftarrow\texttt{A(Y)}\})\{\texttt{Y}\leftarrow\texttt{A(X)}\})\texttt{(X)=y} \wedge \texttt{A(X)=x}\}$$

```
R := A(X)
```

$$\{((\texttt{A}\{\texttt{X}\leftarrow\texttt{A(Y)}\})\{\texttt{Y}\leftarrow\texttt{R}\})\texttt{(X)=y} \wedge \texttt{R=x}\}$$

- **Hence by the derived sequencing rule**

$$\vdash \{((\texttt{A}\{\texttt{X}\leftarrow\texttt{A(Y)}\})\{\texttt{Y}\leftarrow\texttt{A(X)}\})\texttt{(X)=y} \wedge \texttt{A(X)=x}\}$$

```
R := A(X); A(X) := A(Y); A(Y) := R
```

$$\{\texttt{A(X)=y} \wedge \texttt{A(Y)=x}\}$$

- **By the array axioms (considering the cases X=Y and X≠Y separately):**

$$\vdash ((\texttt{A}\{\texttt{X}\leftarrow\texttt{A(Y)}\})\{\texttt{Y}\leftarrow\texttt{A(X)}\})\texttt{(X)} \;=\; \texttt{A(Y)}$$

- **Hence, (as desired)**

$$\vdash \{\texttt{A(Y)=y} \wedge \texttt{A(X)=x}\}$$

```
R := A(X); A(X) := A(Y); A(Y) := R
```

$$\{\texttt{A(X)=y} \wedge \texttt{A(Y)=x}\}$$

FOR-commands

- **Syntax:** `FOR` V`:=`E_1 `UNTIL` E_2 `DO` C
  - **restriction: V must not occur in E_1 or E_2,**
    or be the left hand side of an assignment in C
    (explained later)
- **Semantics:**
  - **if the values of terms E_1 and E_2 are positive numbers e_1 and e_2**
  - **and if $e_1 \leq e_2$**
  - **then C is executed $(e_2 - e_1) + 1$ times with the variable V taking on the sequence of values e_1, $e_1 + 1$, $\ldots$, e_2 in succession**
  - **for any other values, the `FOR`-command has no effect**
- **Example:** `FOR N:=1 UNTIL M DO X:=X+N`
  - **if the value of the variable `M` is m and $m \geq 1$, then the command `X:=X+N` is repeatedly executed with `N` taking the sequence of values `1`, $\ldots$, m**
  - **if $m < 1$ then the `FOR`-command does nothing**

Subtleties of `FOR`-commands

- **There are many subtly different versions of `FOR`-commands**
- **For example**
  - **the expressions E_1 and E_2 could be evaluated at each iteration**
  - **and the controlled variable V could be treated as global rather than local**
- **Early languages like Algol 60 failed to notice such subtleties**
- **Note that with the semantics presented here**
  `FOR`-commands cannot *generate* non termination

More on the semantics of FOR-commands

- **The semantics of**

$$\texttt{FOR}\ V\texttt{:=}E_1\ \texttt{UNTIL}\ E_2\ \texttt{DO}\ C$$

is as follows

(i) E_1 **and** E_2 **are evaluated once to get values** e_1 **and** e_2**, respectively.**

(ii) If either e_1 **or** e_2 **is not a number, or if** $e_1 > e_2$**, then nothing is done.**

(iii) If $e_1 \leq e_2$ **the** FOR**-command is equivalent to:**

$$\texttt{BEGIN VAR}\ V\texttt{;}\ V\texttt{:=}e_1\texttt{;}\ C\texttt{;}\ V\texttt{:=}e_1\texttt{+}1\texttt{;}\ C\ \texttt{;}\ \ldots\ ;\ V\texttt{:=}e_2\texttt{;}\ C\ \texttt{END}$$

i.e. C **is executed** $(e_2{-}e_1){+}1$ **times with** V **taking on the sequence of values** e_1, $e_1{+}1$, $\ldots$, e_2

- **If** C **doesn't modify** V **then** FOR**-command equivalent to:**

$$\texttt{BEGIN VAR}\ V\texttt{;}\ V\texttt{:=}e_1\texttt{;}\ \ldots\ \underbrace{C\ \texttt{;}\ V\texttt{:=}V\texttt{+}1}_{\textbf{repeated}}\texttt{;}\ \ldots\ V\texttt{:=}e_2\texttt{;}\ C\ \texttt{END}$$

The Derived Frame Rule (used in motivating FOR Rule)

- **The following can be derived**

The frame rule

$$\frac{\vdash\ \{P\}\ C\ \{Q\}}{\vdash\ \{P \wedge R\}\ C\ \{Q \wedge R\}}$$

where no variable assigned to in C occurs in R

- **Outline of derivation**
  - **prove $\{R\}\ C\ \{R\}$ by induction on C**
  - **then use Specification Conjunction**

Towards the FOR-Rule

- **If $e_1 \leq e_2$ the** `FOR`**-command is equivalent to:**

$$\texttt{BEGIN VAR } V\texttt{;}\ V\texttt{:=}e_1\texttt{;}\ \ldots\ C\ \texttt{;}\ V\texttt{:=}V{+}1\texttt{;}\ \ldots\ V\texttt{:=}e_2\texttt{;}\ C\ \texttt{END}$$

- **Assume C doesn't modify V and $\vdash \{P\}\ C\ \{P[V{+}1/V]\}$**

- **Hence:**

$\vdash\ \{P[e_1/V]\}\ V\texttt{:=}e_1\ \{P \wedge V{=}e_1\}$ (assign. ax + pre. streng.)
$\vdash\ \{P \wedge V{=}e_1\}\ C\ \{P[V{+}1/V] \wedge V{=}e_1\}$ (assumption + frame rule)
$\vdash\ \{P[V{+}1/V] \wedge V{=}e_1\}\ V\texttt{:=}V{+}1\ \{P \wedge V{=}e_1{+}1\}$ (assign. ax)
$\vdots$
$\vdash\ \{P \wedge V{=}v\}\ C\ \{P[V{+}1/V] \wedge V{=}v\}$ (assumption + frame rule)
$\vdash\ \{P[V{+}1/V] \wedge V{=}v\}\ V\texttt{:=}V{+}1\ \{P \wedge V{=}v{+}1\}$ (assign. ax + pre. streng.)
$\vdots$
$\vdash\ \{P \wedge V{=}e_2\}\ C\ \{P[V{+}1/V] \wedge V{=}e_2\}$ (assumption + frame rule)
$\vdash\ \{P \wedge V{=}e_2\}\ C\ \{P[e_2{+}1/V]\}$ (post. weak.)

- **Hence by the derived block rule**

$$\frac{\vdash \{P\}C\{P[V{+}1/V]\}}{\vdash \{P[e_1/V]\}\texttt{BEGIN VAR } V\texttt{;}V\texttt{:=}e_1\texttt{;}\ \ldots\ C\texttt{;}V\texttt{:=}V{+}1\texttt{;}\ \ldots\ V\texttt{:=}e_2\texttt{;}C\ \texttt{END}\{P[e_2{+}1/V]\}}$$

Problems with the FOR-rule (i)

- **Previous derivation suggests a rule**

$$\frac{\vdash \{P\}\ C\ \{P[V+1/V]\}}{\vdash \{P[E_1/V]\}\ \texttt{FOR}\ V\texttt{:=}E_1\ \texttt{UNTIL}\ E_2\ \texttt{DO}\ C\ \{P[E_2+1/V]\}}$$

- **This is a good start, but needs debugging**
- **Consider:**

$$\vdash\ \{\texttt{X=Y}\}\ \texttt{X:=Y+1}\ \{\texttt{X=Y+1}\}$$

- **Taking P as 'X=Y' we have**

$$\vdash\ \{P\}\ \texttt{X:=Y+1}\ \{P[\texttt{Y+1/Y}]\}$$

- **By the FOR-rule above, with $V = \texttt{Y}$, $E_1 = \texttt{3}$ and $E_2 = \texttt{1}$**

$$\vdash\ \{\ \underbrace{\texttt{X=3}}_{P[\texttt{3/Y}]}\ \}\ \texttt{FOR Y:=}3\ \texttt{UNTIL 1 DO X:=Y+1}\ \{\ \underbrace{\texttt{X=2}}_{P[\texttt{1+1/Y}]}\ \}$$

Problems with the FOR-rule (ii)

- **The conclusion below is clearly undesirable**

$$\vdash \{\underbrace{\texttt{X=3}}_{P\texttt{[3/Y]}}\}\ \texttt{FOR Y:=}3\ \texttt{UNTIL 1 DO X:=Y+1}\ \{\underbrace{\texttt{X=2}}_{P\texttt{[1+1/Y]}}\}$$

- **It was specified that**
  - **if the value of E_1 were greater than the value of E_2**
  - **then the FOR-command should have no effect**
  - **in this example it changes the value of X from 3 to 2**
- **To avoid this, the FOR-rule can be modified to**

$$\frac{\vdash \{P\}\ C\ \{P\texttt{[}V\texttt{+1/}V\texttt{]}\}}{\vdash \{P\texttt{[}E_1/V\texttt{]}\ \wedge\ E_1 \leq E_2\}\ \texttt{FOR}\ V\texttt{:=}E_1\ \texttt{UNTIL}\ E_2\ \texttt{DO}\ C\ \{P\texttt{[}E_2\texttt{+1/}V\texttt{]}\}}$$

Problems with the FOR-rule (iii)

- **FOR-rule so far**

$$\frac{\vdash \{P\}\ C\ \{P\texttt{[}V\texttt{+1}/V\texttt{]}\}}{\vdash \{P\texttt{[}E_1/V\texttt{]}\ \wedge\ E_1 \leq E_2\}\ \texttt{FOR}\ V\texttt{:=}E_1\ \texttt{UNTIL}\ E_2\ \texttt{DO}\ C\ \{P\texttt{[}E_2\texttt{+1}/V\texttt{]}\}}$$

- **On the example just considered this rule results in**

$$\vdash\ \{\texttt{X=3}\ \wedge\ \underbrace{\texttt{3} \leq \texttt{1}}_{\text{never true!}}\}\ \texttt{FOR Y:=}3\ \texttt{UNTIL 1 DO X:=Y+1}\ \{\texttt{X=2}\}$$

- **This conclusion is harmless**
  - **only asserts X changed if FOR-command executed in impossible starting state**

Problems with the FOR-rule (iv)

- **Unfortunately, there is a bug in**

$$\frac{\vdash \{P\}\ C\ \{P[V\texttt{+1}/V]\}}{\vdash \{P[E_1/V]\ \wedge\ E_1 \leq E_2\}\ \texttt{FOR}\ V\texttt{:=}E_1\ \texttt{UNTIL}\ E_2\ \texttt{DO}\ C\ \{P[E_2\texttt{+1}/V]\}}$$

- **Take P to be '$\texttt{Y=1}$' and note that**

$$\vdash\ \{\underbrace{\texttt{Y=1}}_{P}\}\ \texttt{Y:=Y-1}\ \{\underbrace{\texttt{Y+1=1}}_{P[\texttt{Y+1}/\texttt{Y}]}\}$$

- **So by this FOR-rule**

$$\vdash\ \{\underbrace{\texttt{1=1}}_{P[\texttt{1}/\texttt{Y}]}\ \wedge\ 1 \leq 1\}\ \texttt{FOR}\ \ \texttt{Y:=1}\ \texttt{UNTIL}\ \texttt{1}\ \texttt{DO}\ \texttt{Y:=Y-1}\ \{\underbrace{\texttt{2=1}}_{P[\texttt{1+1}/\texttt{Y}]}\}$$

Problems with the `FOR`-rule (v)

- **Whatever the command does, it doesn't lead to a state in which `2=1`**
- **The problem is that the body of the `FOR`-command modifies the controlled variable**
- **This is why it was explicitly assumed that the body didn't modify the controlled variable**

Problems with the FOR-rule (vi)

- **This problem may also arise if any variables in the expressions E_1 and E_2 are modified**
- **For example, taking P to be `Z=Y`, then**

$$\vdash\ \{\underbrace{\texttt{Z=Y}}_{P}\}\ \texttt{Z:=Z+1}\ \{\underbrace{\texttt{Z=Y+1}}_{P[\texttt{Y+1}/\texttt{Y}]}\}$$

- **Thus the following can be derived**

$$\vdash\ \{\underbrace{\texttt{Z=1}}_{P[\texttt{1}/\texttt{Y}]}\ \wedge\ \texttt{1} \leq \texttt{Z}\}\ \texttt{FOR Y:=1 UNTIL Z DO Z:=Z+1}\ \{\underbrace{\texttt{Z=Z+1}}_{P[\texttt{Z+1}/\texttt{Y}]}\}$$

- **This is clearly wrong**
  - **one can never have `Z=Z+1`**
- **Not a problem because the FOR-command doesn't terminate?**
  - **in some languages this might be the case**
  - **semantics of our language defined so that FOR-commands always terminate**

The `FOR`-Rule

- **To rule out the problems that arise when the controlled variable or variables in the bounds expressions, are changed by the body, we simply impose a side condition on the rule that stipulates that it cannot be used in these situations**

The `FOR`-rule

$$\frac{\vdash \{P \wedge (E_1 \leq V) \wedge (V \leq E_2)\}\ C\ \{P[V\texttt{+1}/V]\}}{\vdash \{P[E_1/V] \wedge (E_1 \leq E_2)\}\ \texttt{FOR}\ V := E_1\ \texttt{UNTIL}\ E_2\ \texttt{DO}\ C\ \{P[E_2\texttt{+1}/V]\}}$$

where neither V, nor any variable occurring in E_1 or E_2, is assigned to in the command C.

- **Note $(E_1 \leq V) \wedge (V \leq E_2)$ in precondition of rule hypothesis**
  - **added to strengthen rule to allow proofs to use facts about V's range of values**
- **Can be tricky to think up P**

Comment on the FOR-Rule

- **The FOR-rule does not enable anything to be deduced about FOR-commands whose body assigns to variables in the bounds expressions**
- **This precludes such assignments being used if commands are to be reasoned about**
- **Only defining rules of inference for non-tricky uses of constructs motivates writing programs in a perspicuous manner**
- **It is possible to devise a rule that does cope with assignments to variables in bounds expressions**
- **Consider:**

$$\frac{\vdash \{P \wedge (E_1 \leq V) \wedge (V \leq E_2)\}\ C\ \{P[V+1/V]\}}{\vdash \{P[E_1/V] \wedge (E_1 \leq E_2) \wedge (E_2 = e_2)\}\ \texttt{FOR}\ V := E_1\ \texttt{UNTIL}\ E_2\ \texttt{DO}\ C\ \{P[e_2+1/V]\}}$$

The FOR-axiom

- **To cover the case when $E_2 < E_1$, we need the FOR-axiom below**

The FOR-axiom

$$\vdash \ \{P \wedge (E_2 < E_1)\}\ \texttt{FOR}\ V := E_1\ \texttt{UNTIL}\ E_2\ \texttt{DO}\ C\ \{P\}$$

- **This says that when E_2 is less than E_1 the FOR-command has no effect**

Exercise: understand the example on this slide

- **By the assignment axiom and precondition strengthening**

  $\vdash$ {X = ((N-1)×N) DIV 2} X:=X+N {X=(N×(N+1)) DIV 2}

- **Strengthening the precondition of this again yields**

  $\vdash$ {(X = ((N-1)×N) DIV 2)∧(1≤N)∧(N≤M)}
  X:=X+N
  {X = (N×(N+1)) DIV 2}

- **Hence by the `FOR`-rule**

  $\vdash$ {(X = ((1-1)×1) DIV 2)∧(1≤M)}
  FOR N:=1 UNTIL M DO X:=X+N
  {X = (M×(M+1)) DIV 2}

- **Hence**

  $\vdash$ {(X=0)∧(1≤M)}
  FOR N:=1 UNTIL M DO X:=X+N
  {X = (M×(M+1)) DIV 2}

Note on using the FOR-Rule

- **Note that if any of the following hold**

  (i) $\vdash\ \{P\}\ C\ \{P[V+1/V]\}$

  (ii) $\vdash\ \{P\ \wedge\ (E_1 \leq V)\}\ C\ \{P[V+1/V]\}$

  (iii) $\vdash\ \{P\ \wedge\ (V \leq E_2)\}\ C\ \{P[V+1/V]\}$

- **Then by precondition strengthening:**

  $\vdash\ \{P\ \wedge\ (E_1 \leq V)\ \wedge\ (V \leq E_2)\}\ C\ \{P[V+1/V]\}$

- **So any of (i), (ii) or (iii) above is a sufficient hypothesis for FOR Rule**

Deriving the For Rule

- **The following is a command equivalent to `FOR I:=E`$_1$ `UNTIL E`$_2$ `DO C`**

```
BEGIN
 VAR I;
 VAR UpperBound;
 I := E₁;
 UpperBound := E₂;
 WHILE I≤UpperBound DO (C; I := I+1)
END
```

  - `UpperBound` **is assumed to be a 'new' variable**
  - **and `I` is not assigned to inside `C`**

- **Thus we could derive a rule from the implementation**
  - **we must be sure the implementation is correct**

- **Exercise: try deriving the `FOR`-rule from the `WHILE`-rule**

Exercise: think about Wickerson's FOR-Rule (see below)

The FOR rule as presented in the notes had always seemed quite unsatisfactory to me, because it couldn't deal with the case when the lower and upper bounds on the looping variable were the wrong way around (hence the need for the FOR-axiom).

I have derived a new rule, which removes the need for the FOR-axiom completely. This rule doesn't suffer from the problems that the early incarnations of the FOR-rule suffered from in the lecture notes, and I believe the rule to be equally powerful.

It is derived, very easily, by noting that: FOR V:=E1 UNTIL E2 DO C is equivalent to:

```
 BEGIN VAR V; V:=E1; WHILE V<=E2 DO (C; V:=V+1) END
```

(where the standard syntactic constraints still apply, i.e. neither V nor any variable in E1 or E2 may be assigned to in C). Then we simply apply the Floyd-Hoare rules of blocks, sequencing and while-commands to derive the following rule:

```
   |- P ==> R[E1/V]   |- R & V>E2 ==> Q   |- {R & V<=E2} C {R[V+1/V]}
   -----------------------------------------------------------------
              |- {P} FOR V:=E1 UNTIL E2 DO C {Q}
```

This rule is similar to, but subtly different from, the FOR-rule derived in the notes. I've tried my rule on various examples in the notes, and I reckon it works fine. I'll just give one quick example here; suppose we want to prove:

```
 {X = 2} FOR V := 10 UNTIL 0 DO X:=1 {X=2}
```

Then we set R = (V=10 & X=2). The three antecedents of the (new) rule are instantiated to

```
 (1) X=2 ==> 10=10 & X=2
 (2) V=10 & X=2 & V>0 ==> X=2
 (3) |- {V=10 & X=2 & V<=0} X:=1 {V+1=10 & X=2}
```

Note that (1) and (2) are trivially true, and (3) holds because the precondition is unsatisfiable (V cannot be both equal to 10 and no greater than 0).

Soundness and Completeness

- **It is clear from the discussion of the `FOR`-rule that it is not always straightforward to devise correct rules of inference**
- **It is important that the axioms and rules be sound. There are two approaches to ensure this**

  (i) define the language by the axioms and rules of the logic

  (ii) prove that the logic fits the language

- **Approach (i) is called** *axiomatic semantics*
  - **the idea is to** *define* **the semantics of the language by requiring that it make the axioms and rules of inference true**
  - **it is then up to implementers to ensure that the logic matches the language**

Axiomatic Semantics

- **One snag with axiomatic semantics is that most existing languages have already been defined in some other way**
  - **usually by informal and ambiguous natural language statements**
- **The other snag with axiomatic semantics is that it is known to be impossible to devise adequate Floyd-Hoare logics for certain constructs (Clarke's Theorem)**
  - **it could be argued that this is not a snag at all but an advantage, because it forces programming languages to be made logically tractable**
- **An example of a language defined axiomatically is Euclid**

From *Proof rules for the programming language Euclid*

7.1. (module rule)

(1) $Q \supset Q0(A/t)$,

(2) $P1\{\textbf{const } K; \textbf{ var } V; S_4\}\ Q4(A/t) \wedge Q$,

(3) $P2(A/t) \wedge Q\ \{S_2\}\ Q2(A/t) \wedge Q$,

(4) $\exists g1(P3(A/t) \wedge Q\ \{S_3\}\ Q3(A/t) \wedge g = g1(A, c, d))$,

(5) $\exists g(P3(A/t) \wedge Q \supset Q3(A/t))$,

(6) $P6(A/t) \wedge Q\ \{S_6\}\ Q1$,

(7) $P \supset P1(a/c)$,

(8.1) $[Q0(a/c, x/t, x'/t') \supset (P2(x/t, x'/t', a2/x2, e2/c2, a/c) \wedge$
$(Q2(x2\sharp/t, x'/t', a2\sharp/x2, e2/c2, a/c, y2\sharp/y2, a2/x2', y2/y2') \supset$
$R1(x2\sharp/x, a2\sharp/a2, y2\sharp/y2)))\ \{x\,.\,p(a2, e2)\}\ R1 \wedge Q0(a/c, x/t, x'/t')$,

(8.2) $(Q0(a/c, x/t) \supset P3(x/t, a3/c3, a/c)) \supset$
$Q3(x/t, a3/c3, a/c, f(a3, d3)/g) \wedge Q0(a/c, x/t)$,

(8.3) $P1(a/c) \wedge (Q4(x4\sharp/t, x'/t', a/c, y4\sharp/y4, y4/y4') \supset R4(x4\sharp/x, y4\sharp/y4))$
$\{x\,.\,Initially\}\ R4 \wedge Q0(a/c, x/t, x'/t')$,

(8.4) $(Q0(a/c, x/t, x'/t') \supset P6(x/t, x'/t', a/c)) \wedge (Q1(a/c, y6\sharp/y6, y6/y6') \supset$
$R(y6\sharp/y6))\ \{x\,.\,Finally\}\ R]$

$\vdash$

(8.5) $P(x\sharp/x)\ \{x\,.\,Initially; S; x\,.\,Finally\}\ R(x\sharp/x)$

$P\{\textbf{var } x\text{: } T(a); S\}\ R \wedge Q1$

Proving the Logic Sound

- **The other approach to ensuring the logic is sound is to prove that the axioms and rules are valid**
- **To do this, a mathematical model of states is constructed**
  - **a function,** $Meaning$ **say, is defined which takes an arbitrary command** C **to a function** $Meaning(C)$ **from states to states**
  - **thus** $Meaning(C)(s)$ **denotes the state resulting from executing command** C **in state** s
- **The specification** $\{P\}C\{Q\}$ **is then defined to be true if**
  - **whenever** P **is true in a state** s
  - **and** $Meaning(C)(s) = s'$
  - **then** Q **is true in state** s'
- **It is then possible to attempt to prove rigorously that all the axioms are true and that the rules of inference lead from true premisses to true conclusions**

Completeness

- **Even if we are sure that our logic is sound, how can we be sure that every true specification can be proved?**
- **Maybe for some particular `P`, `Q` and `C` the specification `{P}C{Q}` is true, but the rules of our logic were too weak to prove it. Consider:**
  - `{X=1} BEGIN VAR X; X:=0 END {X=1}`
  - `{T} FOR X:=1 UNTIL 1 DO Y:=X {Y=1}`
- **A logic is said to be *complete* if every true statement in it is provable**

Completeness of Hoare Logic

- **Consider $\{\texttt{T}\}$ `IF` S `THEN X := 1 ELSE X := 0` $\{\texttt{X = 1}\}$**
  - **S could be an arbitrary statement of arithmetic**
  - **arithmetic is known to be undecidable**
- **There are various subtle technical problems in formulating precisely what it means for a Floyd-Hoare logic to be complete**
  - **it is necessary to distinguish incompleteness arising due to incompleteness in the assertion language (e.g. arithmetic)**
  - **from incompleteness due to inadequate axioms and rules for programming language constructs**
  - **the completeness of a Floyd-Hoare logic must thus be defined independently of that of its assertion language**
- **The correct notion of completeness is called 'relative completeness'**
  - *roughly*: **complete assuming an oracle for logic and mathematical theories**

Clarke's Results

- **Ed Clarke (of Carnegie Mellon University) has proved the impossibility of giving complete inference systems for certain combinations of programming language constructs**
- **He shows it is *impossible* to give a sound and relatively complete system for any language combining:**
  - **procedures as parameters of procedure calls**
  - **recursion**
  - **static scopes**
  - **global variables**
  - **internal procedures as parameters of procedure calls**
- **These features are found in Algol 60, which thus cannot have an adequate Floyd-Hoare logic**

Summary

- **Devising rules for programming constructs can be tricky**
  - for safety critical applications logically tractable languages should be used
- **Two desirable properties of the logic are:**
  - **soundness:** only true things can be proved
  - **completeness:** everything that is true can be proved
- **Fully automatic program verification is impossible**
  - arithmetic is undecidable
  - in practice a lot of success verifying weak properties automatically (e.g. `http://research.microsoft.com/en-us/um/redmond/groups/rise/staticanalysis.aspx`)
- **User-assisted program verification is active research area**