Program Refinement

Refinement

- **So far we have focused on proving programs meet specifications**
- **An alternative is to ensure a program is correct by construction**
- **The proof is performed in conjunction with the development**
  - **errors are spotted earlier in the design process**
  - **the reasons for design decisions are available**
- **Programming becomes less of a black art and more like an engineering discipline**
- **Rigorous development methods such as the B-Method, SPARK and the Vienna Development Method (VDM) are based on this idea**
- **The approach here is based on “Programming From Specifications”**
  - **a book by Carroll Morgan**
  - **simplified and with a more concrete semantics**

Refinement Laws

- **Laws of Programming** refine a specification to a program
- As each law is applied, proof obligations are generated
- The laws are derived from the Hoare logic rules
- Several laws will be applicable at a given time
  - corresponding to different design decisions
  - and thus different implementations
- The "Art" of Refinement is in choosing appropriate laws to give an efficient implementation
- For example, given a specification that an array should be sorted:
  - one sequence of laws will lead to Bubble Sort
  - a different sequence will lead to Insertion Sort
  - see Morgan's book for an example of this

Refinement Specifications

- **A** *refinement specification* **has the form** $[P,\ Q]$
  - P **is the precondition**
  - Q **is the postcondition**
- **Unlike a partial or total correctness specification, a refinement specification does not include a command**
- **Goal: derive a command that satisfies the specification**
- P **and** Q **correspond to the pre and post condition of a total correctness specification**
- **A command is required which if started in a state satisfying** P**, will terminate in a state satisfying** Q

Example

- [T, X=1]
  - this specifies that the code provided should terminate in a state where X has value 1 whatever state it is started in
- [X>0, Y=X^2]
  - from a state where X is greater than zero, the program should terminate with Y the square of X

A Little Wide Spectrum Programming Language

- **Let P, Q range over statements (predicate calculus formulae)**
- **Add specifications to commands**

$$E \;::=\; N \mid V \mid E_1 + E_2 \mid E_1 - E_2 \mid E_1 \times E_2 \mid \ldots$$

$$B \;::=\; \mathtt{T} \mid \mathtt{F} \mid E_1{=}E_2 \mid E_1 \leq E_2 \mid \ldots$$

$$\begin{array}{lll} C & ::= & \mathtt{SKIP} \\ & \mid & V\ \mathtt{:=}\ E \\ & \mid & C_1\ \mathtt{;}\ C_2 \\ & \mid & \mathtt{IF}\ B\ \mathtt{THEN}\ C_1\ \mathtt{ELSE}\ C_2 \\ & \mid & \mathtt{BEGIN\ VAR}\ V_1\ \mathtt{;}\ \mathtt{..}\ \mathtt{VAR}\ V_1\ \mathtt{;}\ C\ \mathtt{END} \\ & \mid & \mathtt{WHILE}\ B\ \mathtt{DO}\ C \\ & \mid & [P,\ Q] \end{array}$$

Specifications as Sets of Commands

- **Refinement specifications can be mixed with other commands but are not in general executable**
- **Example**

```
R:=X;
Q:=0;
[R=X ∧ Y> 0 ∧ Q=0, X=R+Y×Q]
```

- **Think of a specification as defining the set of implementations**

$$[P,\ Q]\ =\ \{\ C\ \mid\ \ \vdash\ [P]\ C\ [Q]\ \}$$

- **For example**

$$[\mathtt{T},\ \mathtt{X=1}]\ =\ \{\texttt{"X:=1"},\ \texttt{"IF}\ \neg\texttt{(X=1) THEN X:=1"},\ \texttt{"X:=2;X:=X-1"},\ \cdots\ \}$$

Notation for combining sets of commands

- **Wide spectrum language commands are sets of ordinary commands**
- **Let c, c_1, c_2 etc. denote sets of commands, then define:**

$$
\begin{array}{ll}
c_1;\ \cdots\ ;c_n & = \{\ C \ \mid\ \exists C_1 \cdots C_n.\ C\ =\ C_1;\ \cdots\ ;C_n\ \wedge \\
 & \qquad\qquad C_1 \in c_1\ \wedge\ \cdots\ \wedge\ C_n \in c_n\ \} \\
 & \\
 & = \{\ C_1;\ \cdots\ ;C_n\ \mid\ C_1 \in c_1\ \wedge\ \cdots\ \wedge\ C_n \in c_n\ \} \\
 & \\
\texttt{BEGIN VAR}\ V_1;\ \cdots\ \texttt{VAR}\ V_n;\ c\ \texttt{END} & = \{\ \texttt{BEGIN VAR}\ V_1;\ \cdots\ \texttt{VAR}\ V_n;\ C\ \texttt{END}\ \mid\ C \in c\ \} \\
 & \\
\texttt{IF}\ S\ \texttt{THEN}\ c_1\ \texttt{ELSE}\ c_2 & = \{\ \texttt{IF}\ S\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2\ \mid\ C_1 \in c_1\ \wedge\ C_2 \in c_2\ \} \\
 & \\
\texttt{WHILE}\ S\ \texttt{DO}\ c & = \{\ \texttt{WHILE}\ S\ \texttt{DO}\ C\ \mid\ C \in c\ \}
\end{array}
$$

Refinement based program development

- **The client provides a non-executable program (the specification)**
- **The programmer's job is to transform it into an executable program**
- **It will pass through a series of stages in which some parts are executable, but others are not**
- **Specifications give lots of freedom about how a result is obtained**
  - **executable code has no freedom**
  - **mixed programs have some freedom**
- **We use the notation** $p_1 \supseteq p_2$ **to mean program** p_2 **is more refined (i.e. has less freedom) than program** p_1
- **N.B. The standard notation is** $p_1 \sqsubseteq p_2$ **(see the notes for a discussion)**
- **A program development takes us from the specification, through a series of mixed programs to (we hope) executable code**

$$spec \supseteq mixed_1 \supseteq ... \supseteq mixed_n \supseteq code$$

Skip Law

The Skip Law

$$[P,\quad P] \supseteq \{\texttt{SKIP}\}$$

- **Derivation:**

$$\begin{array}{ll}
C \in \{\texttt{SKIP}\} & \\
\quad \Leftrightarrow C \;=\; \texttt{SKIP} & \\
\quad \Rightarrow \;\vdash\; [P]\ C\ [P] & \text{(Skip Axiom)} \\
\quad \Leftrightarrow C \in [P,\ P] & \text{(Definition of } [P,\ P]\text{)}
\end{array}$$

- **Examples**

  [X=1, X=1] $\supseteq$ {SKIP}

  [T, T] $\supseteq$ {SKIP}

  [X=R+Y$\times$Q, X=R+Y$\times$Q] $\supseteq$ {SKIP}

Notational Convention

- **Omit { and } around individual commands**
- **Skip law becomes:**

$$[P, \quad P] \supseteq \texttt{SKIP}$$

- **Examples become:**

$\texttt{[X=1,} \quad \texttt{X=1]} \supseteq \texttt{SKIP}$

$\texttt{[T,} \quad \texttt{T]} \supseteq \texttt{SKIP}$

$\texttt{[X=R+Y}\times\texttt{Q,} \quad \texttt{X=R+Y}\times\texttt{Q]} \supseteq \texttt{SKIP}$

Assignment Law

The Assignment Law

$$[P[E/V],\quad P] \supseteq \{V \texttt{ := } E\}$$

- **Derivation**

$$\begin{array}{ll}
C \in \{V \texttt{ := } E\} & \\
\quad \Leftrightarrow C \;=\; V \texttt{ := } E & \\
\quad \Rightarrow \;\vdash\; [P\texttt{[}E/V\texttt{]}]\; C\; [P] & \text{(Assignment Axiom)} \\
\quad \Leftrightarrow C \in [P\texttt{[}E/V\texttt{]},\; P] & \text{(Definition of } [P\texttt{[}E/V\texttt{]},\; P])
\end{array}$$

- **Examples (using bracket-omitting convention)**

`[Y=1,  X=1]` $\supseteq$ `X:=Y`

`[X+1=n+1,  X=n+1]` $\supseteq$ `X:=X+1`

Laws of Consequence

Precondition Weakening

$$[P, \quad Q] \sqsupseteq [R, \quad Q]$$
$$\textbf{provided } \vdash \quad P \;\Rightarrow\; R$$

Postcondition Strengthening

$$[P, \quad Q] \sqsupseteq [P, \quad R]$$
$$\textbf{provided } \vdash \quad R \;\Rightarrow\; Q$$

- **We are now "weakening the precondition" and "strengthening the post condition"**
  - **this is the opposite terminology to the Hoare rules**
  - **refinement consequence rules are 'backwards'**

Derivation of Consequence Laws

- **Derivation of Precondition Weakening**

$$
\begin{array}{l}
C \in [R,\ Q] \\
\quad \Leftrightarrow\ \vdash\ [R]\ C\ [Q]\ \ \text{(Definition of } [R,\ Q]\text{)} \\
\quad \Rightarrow\ \vdash\ [P]\ C\ [Q]\ \ \text{(Precondition Strengthening} \vdash P \Rightarrow R\text{)} \\
\quad \Leftrightarrow\ C \in [P,\ Q]\ \ \ \ \text{(Definition of } [P,\ Q]\text{)}
\end{array}
$$

- **Derivation of Postcondition Strengthening**

$$
\begin{array}{l}
C \in [P,\ R] \\
\quad \Leftrightarrow\ \vdash\ [P]\ C\ [R]\ \ \text{(Definition of } [R,\ Q]\text{)} \\
\quad \Rightarrow\ \vdash\ [P]\ C\ [Q]\ \ \text{(Postcondition Weakening} \vdash R \Rightarrow Q\text{)} \\
\quad \Leftrightarrow\ C \in [P,\ Q]\ \ \ \ \text{(Definition of } [P,\ Q]\text{)}
\end{array}
$$

Examples (illustrates refinement notation)

- **A previous example:**

  `[X=1,  X=1]`

  $\sqsupseteq$ (`Skip`)

  `SKIP`

- **An alternative refinement:**

  `[Y=1,  X=1]`

  $\sqsupseteq$ (`Precondition Weakening` $\vdash$ `Y=1` $\Rightarrow$ `1=1`)

  `[1=1,  X=1]`

  $\sqsupseteq$ (`Assignment`)

  `X := 1`

- **Another example**

  `[T,  R=X]`

  $\sqsupseteq$ (`Precondition Weakening` $\vdash$ `T` $\Rightarrow$ `X=X`)

  `[X=X,  R=X]`

  $\sqsupseteq$ (`Assignment`)

  `R := X`

Derived Assignment Law

Derived Assignment Law

$$[P, \quad Q] \supseteq \{V := E\}$$

provided $\vdash \; P \; \Rightarrow \; Q[E/V]$

- **Derivation**

  $[P, \quad Q]$

  $\supseteq$ (Precondition Weakening $\vdash \; P \Rightarrow Q[E/V]$)

  $[Q[E/V], \quad Q]$

  $\supseteq$ (Assignment)

  $V := E$

- **Example**

  [T, R=X]

  $\supseteq$ (Derived Assignment $\vdash$ T $\Rightarrow$ X=X)

  R := X

Technical interlude: Monotonicity

- **Sets of commands are *monotonic* w.r.t. $\supseteq$**
  - **if** $c \supseteq c'$, $c_1 \supseteq c_1'$, … , $c_n \supseteq c_n'$
  - **then:**

$$
\begin{array}{lll}
c_1;\ \cdots\ ;c_n & \supseteq & c_1';\ \cdots\ ;c_n' \\[1em]
\texttt{BEGIN VAR}\ V_1;\ \cdots\ \texttt{VAR}\ V_n;\ c\ \texttt{END} & \supseteq & \texttt{BEGIN VAR}\ V_1;\ \cdots\ \texttt{VAR}\ V_n;\ c'\ \texttt{END} \\[1em]
\texttt{IF}\ S\ \texttt{THEN}\ c_1\ \texttt{ELSE}\ c_2 & \supseteq & \texttt{IF}\ S\ \texttt{THEN}\ c_1'\ \texttt{ELSE}\ c_2' \\[1em]
\texttt{WHILE}\ S\ \texttt{DO}\ c & \supseteq & \texttt{WHILE}\ S\ \texttt{DO}\ c'
\end{array}
$$

- **Monotonicity shows that a command can be refined by separately refining its constituents**
- **Laws of refinement now follow**

Sequencing

The Sequencing Law

$$[P,\ \ Q] \supseteq [P,\ \ R];\ [R,\ \ Q]$$

- **Derivation of Sequencing Law**

$C \in [P,\ R]\ ;\ [R,\ Q]$

$\Leftrightarrow\ C \in \{\ C_1\ ;\ C_2\ \mid\ C_1 \in [P,\ R]\ \wedge\ C_2 \in [R,\ Q]\}$ (Definition of $c_1\ ;\ c_2$)

$\Leftrightarrow\ C \in \{\ C_1\ ;\ C_2\ \mid\ \ \vdash\ [P]\ C_1\ [R]\ \wedge\ \ \vdash\ [R]\ C_2\ [Q]\}$ (Definition of $[P,\ R]$ and $[R,\ Q]$)

$\Rightarrow\ C \in \{\ C_1\ ;\ C_2\ \mid\ \ \vdash\ [P]\ C_1\ ;\ C_2\ [Q]\}$ (Sequencing Rule)

$\Rightarrow\ \ \vdash\ [P]\ C\ [Q]$

$\Leftrightarrow\ C \in [P,\ Q]$ (Definition of $[P,\ Q]$)

- **Example**

```
[T,  R=X∧Q=0]
⊒ (Sequencing)
[T,  R=X] ; [R=X,  R=X∧Q=0]
⊒ (Derived Assignment ⊢  T ⇒ X=X)
R:=X; [R=X,  R=X∧Q=0]
⊒ (Derived Assignment ⊢  R=X ⇒ R=X ∧ 0=0)
R:=X; Q:=0
```

Creating different Programs

- **By applying the laws in a different way, we obtain different programs**
- **Consider previous example: using a different assertion with the sequencing law creates a program with the assignments reversed**

  `[T,  R=X∧Q=0]`

  $\supseteq$ (`Sequencing`)

  `[T,  Q=0] ; [Q=0,  R=X∧Q=0]`

  $\supseteq$ (`Derived Assignment` $\vdash$ `T` $\Rightarrow$ `0=0`)

  `Q:=0; [Q=0,  R=X∧Q=0]`

  $\supseteq$ (`Derived Assignment` $\vdash$ `Q=0` $\Rightarrow$ `X=X` $\wedge$ `Q=0`)

  `Q:=0; R:=X`

Inefficient Programs

- **Refinement does not prevent you making silly coding decisions**
- **It does prevent you from producing incorrect executable code**
- **Example**

```
[T,  R=X∧Q=0]
⊇ (Sequencing)
[T,  R=X∧Q=0] ; [R=X∧Q=0,  R=X∧Q=0]
⊇ (as previous example)
Q:=0; R:=X; [R=X∧Q=0,  R=X∧Q=0]
⊇ (Skip)
Q:=0; R:=X; SKIP
```

Blind Alleys

- **The refinement rules give the freedom to wander down blind alleys**
- **We may end up with an unrefinable step**
  - **since it will not be executable, this is safe**
  - **we will not get an incorrect executable program**
- **Example**

```
[X=x∧Y=y,  X=y∧Y=x]
⊇ (Sequencing)
[X=x∧Y=y,  X=x∧Y=x] ; [X=x∧Y=x,  X=y∧Y=x]
⊇ (Derived Assignment  ⊢ X=x∧Y=y⇒X=x∧X=x)
Y:=X; [X=x∧Y=x,  X=y∧Y=x]
⊇ (Sequencing)
Y:=X;
[X=x∧Y=x,  Y=y∧Y=x];
[Y=y∧Y=x,  X=y∧Y=x]
⊇ (Assignment)
Y:=X;
[X=x∧Y=x,  Y=y∧Y=x];
X:=Y
```

Blocks

The Block Law

$$[P,\quad Q] \supseteq \texttt{BEGIN VAR } V_1\texttt{;}\ \ldots\ \texttt{; VAR } V_n\texttt{;}\ [P,\quad Q]\ \texttt{END}$$

where $V_1, \ldots, V_n$ do not occur in P or Q

- **Derivation: exercise (see notes)**
- **Example**

```
[X=x∧Y=y,  X=y∧Y=x]
⊇ (Block)
BEGIN VAR R; [X=x∧Y=y,  X=y∧Y=x] END
⊇ (Sequencing and Derived Assignment)
BEGIN VAR R; R:=X; X:=Y; Y:=R END
```

Conditional

The Conditional Law

$$[P, \quad Q] \supseteq \texttt{IF}\ S\ \texttt{THEN}\ [P \wedge S, \quad Q]\ \texttt{ELSE}\ [P \wedge \neg S, \quad Q]$$

- **The Conditional Law can be used to refine** *any* **specification and** *any* **test can be introduced**
- **You may not make any progress by applying the law however**
  - **you may need the same program on each branch!**

Derivation of the Conditional Law

$C \in$ IF S THEN $[P \wedge S,\ Q]$ ELSE $[P \wedge \neg S,\ Q]$

$\Leftrightarrow\ C \in \{$IF S THEN C_1 ELSE C_2 $\mid$
$C_1 \in [P \wedge S,\ Q]\ \&\ C_2 \in [P \wedge \neg S,\ Q]\}$ (Definition of IF S THEN $\{\cdots\}$ ELSE $\{\cdots\}$)

$\Leftrightarrow\ C \in \{$IF S THEN C_1 THEN C_2 $\mid$
$\vdash\ [P \wedge S]\ C_1\ [Q]\ \&\ \vdash\ [P \wedge \neg S]\ C_2\ [Q]\}$ (Definition of $[P \wedge S,\ Q]\ \&\ [P \wedge \neg S,\ Q]$)

$\Rightarrow\ C \in \{$IF S THEN C_1 ELSE C_2 $\mid$
$\vdash\ [P]$ IF S THEN C_1 ELSE C_2 $[Q]\}$ (Two-armed Conditional Rule)

$\Rightarrow\ \vdash\ [P]\ C\ [Q]$

$\Leftrightarrow\ C \in [P,\ Q]$ (Definition of $[P,\ Q]$)

Example

$$[\texttt{T},\ \ \texttt{M=max(X,Y)}]$$

$\supseteq$ (Conditional)

```
IF X≥Y
THEN [T∧ X≥Y,  M=max(X,Y)]
ELSE [T∧¬(X≥Y),  M=max(X,Y)]
```

$\supseteq$ (Derived Assignment $\vdash$ T∧X≥Y $\Rightarrow$ X=max(X,Y))

```
IF X≥Y
THEN M:=X
ELSE [T∧¬X≥Y,  M=max(X,Y)]
```

$\supseteq$ (Derived Assignment $\vdash$ T∧¬X≥Y $\Rightarrow$ Y=max(X,Y))

```
IF X≥Y THEN M:=X ELSE M:=Y
```

While

The While Law

$$[R,\quad R \wedge \neg S] \supseteq \texttt{WHILE}\ S\ \texttt{DO}\ [R \wedge S \wedge (E\texttt{=}n),\quad R \wedge (E\texttt{<}n)]$$

$$\textbf{provided}\ \vdash\quad R \wedge S \Rightarrow\ E \geq 0$$

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

- **Example**

$[\texttt{X=R+Y}\times\texttt{Q}\ \wedge\ \texttt{Y>0},\quad \texttt{X=R+Y}\times\texttt{Q}\ \wedge\ \texttt{Y>0}\ \wedge\ \neg\ \texttt{Y}\leq\texttt{R]}$

$\supseteq\ (\texttt{While}\quad\vdash\quad \texttt{X=R+Y}\times\texttt{Q}\ \wedge\ \texttt{Y>0}\ \wedge\ \texttt{Y}\leq\texttt{R}\ \Rightarrow\ \texttt{R}\geq\texttt{0})$

$\texttt{WHILE Y}\leq\texttt{R DO}$

$\quad[\texttt{X=R+Y}\times\texttt{Q}\ \wedge\ \texttt{Y>0}\ \wedge\ \texttt{Y}\leq\texttt{R}\ \wedge\ \texttt{R=n},$

$\quad\ \texttt{X=R+Y}\times\texttt{Q}\ \wedge\ \texttt{Y>0}\ \wedge\ \texttt{R<n]}$

Derivation of the While Law

$C \in \texttt{WHILE}\ S\ \texttt{DO}\ [P \wedge S \wedge (E = n),\ P \wedge (E < n)]$

$\Leftrightarrow\ C \in \{\texttt{WHILE}\ S\ \texttt{DO}\ C' \mid C' \in [P \wedge S \wedge (E = n),\ P \wedge (E < n)]\}$ (Definition of $\texttt{WHILE}\ S\ \texttt{DO}\ \{\cdots\}$)

$\Leftrightarrow\ C \in \{\texttt{WHILE}\ S\ \texttt{DO}\ C' \mid\ \vdash\ [P \wedge S \wedge (E = n)]\ C'\ [P \wedge (E < n)]\}$ (Definition of $[P \wedge S \wedge (E = n),\ P \wedge (E < n)]$)

$\Rightarrow\ C \in \{\texttt{WHILE}\ S\ \texttt{DO}\ C' \mid\ \vdash\ [P]\ \texttt{WHILE}\ S\ \texttt{DO}\ C'\ [P \wedge \neg S]\}$ (While Rule & $\vdash\ P \wedge S \Rightarrow E \geq 0$)

$\Rightarrow\ \vdash\ [P]\ C\ [P \wedge \neg S]$

$\Leftrightarrow\ C \in [P,\ P \wedge \neg S]$ (Definition of $[P,\ P \wedge \neg S]$)

Example (i)

$$
\begin{array}{l}
\texttt{[Y>0,}\ \ \texttt{X=R+Y}\times\texttt{Q}\ \wedge\ \texttt{R}\ \leq\ \texttt{Y]} \\
\supseteq\ (\texttt{Block}) \\
\texttt{BEGIN [Y>0,}\ \ \texttt{X=R+Y}\times\texttt{Q}\ \wedge\ \texttt{R}\ \leq\ \texttt{Y] END} \\
\supseteq\ (\texttt{Sequencing}) \\
\texttt{BEGIN} \\
\texttt{[Y>0,}\ \ \texttt{R=X}\ \wedge\ \texttt{Y>0] ;} \\
\texttt{[R=X}\ \wedge\ \texttt{Y>0,}\ \ \texttt{X=R+Y}\times\texttt{Q}\ \wedge\ \texttt{R}\ \leq\ \texttt{Y]} \\
\texttt{END} \\
\supseteq\ (\texttt{Derived Assignment}\ \ \vdash\ \ \texttt{Y>0}\ \Rightarrow\ \texttt{X=X}\ \wedge\ \texttt{Y>0}) \\
\texttt{BEGIN} \\
\texttt{R:=X ;} \\
\texttt{[R=X}\ \wedge\ \texttt{Y>0,}\ \ \texttt{X=R+Y}\times\texttt{Q}\ \wedge\ \texttt{R}\ \leq\ \texttt{Y]} \\
\texttt{END}
\end{array}
$$

Example (ii)

BEGIN
R:=X ;
[R=X $\wedge$ Y>0, X=R+Y$\times$Q $\wedge$ R $\leq$ Y]
END $\sqsupseteq$ (Sequencing)

BEGIN
R:=X ;
[R=X $\wedge$ Y>0, R=X $\wedge$ Y>0 $\wedge$ Q=0] ;
[R=X $\wedge$ Y>0 $\wedge$ Q=0, X=R+Y$\times$Q $\wedge$ R $\leq$ Y]
END

$\sqsupseteq$ (Derived Assignment $\vdash$ R=X $\wedge$ Y>0 $\Rightarrow$ R=X $\wedge$ Y>0 $\wedge$ 0=0)
BEGIN
R:=X ;
Q:=0 ;
[R=X $\wedge$ Y>0 $\wedge$ Q=0, X=R+Y$\times$Q $\wedge$ R $\leq$ Y]
END

- **Exercise: complete the refinement (see next few slides)**

Example (iii)

$$\sqsupseteq \left(\begin{array}{l}\texttt{Precondition Weakening}\\ \qquad \vdash \texttt{R=X} \wedge \texttt{Y>0} \wedge \texttt{Q=0} \Rightarrow \texttt{X=R+Y}\times\texttt{Q} \wedge \texttt{Y>0}\end{array}\right)$$

```
BEGIN
R:=X; Q:=0;
[X=R+Y×Q ∧ Y>0,  X=R+Y×Q ∧ R ≤ Y]
END
```

$$\sqsupseteq \left(\begin{array}{l}\texttt{Postcondition Strengthening}\\ \qquad \vdash \texttt{X=R+Y}\times\texttt{Q} \wedge \texttt{Y>0} \wedge \neg(\texttt{Y}\leq\texttt{R})\\ \qquad\qquad \Rightarrow \texttt{X=R+Y}\times\texttt{Q} \wedge \texttt{R}\leq\texttt{Y}\end{array}\right)$$

```
BEGIN
R:=X; Q:=0;
[X=R+Y×Q ∧ Y>0,  X=R+Y×Q ∧ Y>0 ∧ ¬(Y≤R)]
END
```

$\sqsupseteq$ (While $\vdash$ X=R+Y×Q ∧ Y>0 ∧ Y≤R ⇒ R≥0)

```
BEGIN
R:=X; Q:=0;
WHILE Y ≤ R DO
  [X=R+Y×Q ∧ Y>0 ∧ Y≤R ∧ R=n,
   X=R+Y×Q ∧ Y>0 ∧ R<n]
END
```

Example (iv)

```
⊇ (Block)
BEGIN
R:=X; Q:=0;
WHILE Y ≤ R DO
 BEGIN
   [X=R+Y×Q ∧ Y>0 ∧ Y≤R ∧ R=n,
    X=R+Y×Q ∧ Y>0 ∧ R<n]
 END
END
⊇ (Sequence)
BEGIN
R:=X; Q:=0;
WHILE Y ≤ R DO
 BEGIN
   [X=R+Y×Q ∧ Y>0 ∧ Y≤R ∧ R=n,
    X=(R-Y)+Y×Q ∧ Y>0 ∧ (R-Y)<n] ;
   [X=(R-Y)+Y×Q ∧ Y>0 ∧ (R-Y)<n,
    X=R+Y×Q ∧ Y>0 ∧ R<n]
 END
END
```

Example (v)

```
⊇ (Assignment)
BEGIN
R:=X; Q:=0;
WHILE Y ≤ R DO
 BEGIN
   [X=R+Y×Q ∧ Y>0 ∧ Y≤R ∧ R=n,
    X=(R-Y)+Y×Q ∧ Y>0 ∧ (R-Y)<n] ;
  R := R-Y
 END
END
  ⎛ Derived Assignment                      ⎞
⊇ ⎜     ⊢ X=R+Y×Q ∧ Y>0 ∧ Y≤R ∧ R=n ⇒      ⎟
  ⎝       X=(R-Y)+Y×(Q+1) ∧ Y>0 ∧ (R-Y)<n   ⎠
BEGIN
R:=X; Q:=0;
WHILE Y ≤ R DO
 BEGIN
   Q:= Q+1;
  R:= R-Y
 END
END
```

More Notation

- **The notation**

  $$[P_1,\ P_2,\ P_3,\ \cdots\ , P_{n-1},\ P_n]$$

  is used to abbreviate:

  $$[P_1,\ P_2]\ ;\ [P_2,\ P_3]\ ;\ \cdots\ ;\ [P_{n-1},\ P_n]$$

- **Brackets around specifications** $\{C\}$ **omitted**
- **If** $\mathcal{C}$ **is a set of commands, then**

  $$R\ \texttt{:=}\ X\ ;\ \mathcal{C}$$

  abbreviates

  $$\{R\ \texttt{:=}\ X\}\ ;\ \mathcal{C}$$

Exercise: check the refinement on this slide

- **Let $\mathcal{I}$ stand for $X = R + (Y \times Q)$, then:**

$$
\begin{array}{l}
[Y > 0,\ \ \mathcal{I}\ \wedge\ R \leq Y] \\
\supseteq\ \text{(Sequencing)} \\
[Y > 0,\ R = X\ \wedge\ Y > 0,\ \mathcal{I}\ \wedge\ R \leq Y] \\
\supseteq\ \text{(Assignment)} \\
R\ \texttt{:=}\ X\ \texttt{;}\ [R = X\ \wedge\ Y > 0,\ \mathcal{I}\ \wedge\ R \leq Y] \\
\supseteq\ \text{(Sequencing)} \\
R\ \texttt{:=}\ X\ \texttt{;}\ [R = X\ \wedge\ Y > 0,\ R = X\ \wedge\ Y > 0\ \wedge\ Q = 0,\ \mathcal{I}\ \wedge\ R \leq Y] \\
\supseteq\ \text{(Assignment)} \\
R\ \texttt{:=}\ X\ \texttt{;}\ Q\ \texttt{:=}\ 0\ \texttt{;}\ [R = X\ \wedge\ Y > 0\ \wedge\ Q = 0,\ \mathcal{I}\ \wedge\ R \leq Y] \\
\supseteq\ \text{(Precondition Weakening)} \\
R\ \texttt{:=}\ X\ \texttt{;}\ Q\ \texttt{:=}\ 0\ \texttt{;}\ [\mathcal{I}\ \wedge\ Y > 0,\ \mathcal{I}\ \wedge\ R \leq Y] \\
\supseteq\ \text{(Postcondition Strengthening)} \\
R\ \texttt{:=}\ X\ \texttt{;}\ Q\ \texttt{:=}\ 0\ \texttt{;}\ [\mathcal{I}\ \wedge\ Y > 0,\ \mathcal{I}\ \wedge\ Y > 0\ \wedge\ \neg(Y \leq R)] \\
\supseteq\ \text{(While)} \\
R\ \texttt{:=}\ X\ \texttt{;}\ Q\ \texttt{:=}\ 0\ \texttt{;} \\
\texttt{WHILE}\ Y \leq R\ \texttt{DO}\ [\mathcal{I}\ \wedge\ Y > 0\ \wedge\ Y \leq R\ \wedge\ R = n, \\
\qquad\qquad\qquad \mathcal{I}\ \wedge\ Y > 0\ \wedge\ R < n] \\
\supseteq\ \text{(Sequencing)} \\
R\ \texttt{:=}\ X\ \texttt{;}\ Q\ \texttt{:=}\ 0\ \texttt{;} \\
\texttt{WHILE}\ Y \leq R\ \texttt{DO}\ [\mathcal{I}\ \wedge\ Y > 0\ \wedge\ Y \leq R\ \wedge\ R = n, \\
\qquad\qquad\qquad X = (R - Y) + (Y \times Q)\ \wedge\ Y > 0\ \wedge\ (R - Y) < n, \\
\qquad\qquad\qquad \mathcal{I}\ \wedge\ Y > 0\ \wedge\ R < n] \\
\supseteq\ \text{(Derived Assignment)} \\
R\ \texttt{:=}\ X\ \texttt{;}\ Q\ \texttt{:=}\ 0\ \texttt{;} \\
\texttt{WHILE}\ Y \leq R\ \texttt{DO}\ [\mathcal{I}\ \wedge\ Y > 0\ \wedge\ Y \leq R\ \wedge\ R = n, \\
\qquad\qquad\qquad X = (R - Y) + (Y \times Q)\ \wedge\ Y > 0\ \wedge\ (R - Y) < n]\texttt{;} \\
\qquad\qquad\qquad R\ \texttt{:=}\ R - Y \\
\supseteq\ \text{(Derived Assignment)} \\
R\ \texttt{:=}\ X\ \texttt{;}\ Q\ \texttt{:=}\ 0\ \texttt{;} \\
\texttt{WHILE}\ Y \leq R\ \texttt{DO}\ Q\ \texttt{:=}\ Q + 1\ \texttt{;}\ R\ \texttt{:=}\ R - Y
\end{array}
$$

Derived Laws

- **Above development could be shortened by deriving appropriate laws**
- **For example, a derived `WHILE` law could be derived**
- **Exercise: Develop a factorial program from the specification:**

$$[X = n, \quad Y = n!]$$

- **Exercise: devise refinement laws for arrays, one-armed conditionals, and `FOR`-commands**

Data Refinement

- So far we have given laws to refine commands
- This is termed *Operation Refinement*
- It is also useful to be able to refine the representation of data
  - replacing an abstract data representation by a more concrete one
  - e.g. replacing numbers by binary representations
- This is termed *Data Refinement*
- Data Refinement Laws allow us to make refinements of this form
- The details are beyond the scope of this course
  - they can be found in Morgan's book

Summary

- Refinement 'laws' based on the Hoare logic can be used to develop programs formally
- A program is gradually converted from an unexecutable specification to executable code
- By applying different laws, different programs are obtained
  - may reach unrefinable specifications (blind alleys)
  - but will never get incorrect code
- A program developed in this way will meet its formal specification
  - one approach to 'Correct by Construction' (CbC) software engineering