

6 Refinement

6.1 Introduction

Refinement

- So far we have focused on proving that complete programs meet their 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 and the Vienna Development Method (VDM) are based on this idea
- The approach here is based on “Programming From Specifications” by Carroll Morgan, but 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 further sequence will lead to Insertion Sort
 - see Morgan’s book for an example of this

6.2 Refinement Specifications

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 (the aim is to 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
- The specification says what the client wants: the programmer must supply it

Examples

- $[T, X = 1]$
 - 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 | V | E1 + E2 | E1 - E2 | E1 × E2 | ...
B ::= T | F | E1 = E2 | E1 ≤ E2 | ...
C ::= SKIP
    | V := E
    | C1 ; C2
    | IF B THEN C1 ELSE C2
    | BEGIN VAR V1 ; ... VAR Vn ; C END
    | WHILE B DO C
    | [P, Q]

```

6.3 Specifications as Sets of Commands

Specifications as Sets of Commands

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

- Example:

```

R := X;
Q := 0;
WHILE Y ≤ R DO
    [X = R + Y × Q ∧ Y ≤ R, X = R + Y × Q]

```

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

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

- For example:

$$[T, X = 1] = \{ \text{“}X := 1\text{”}, \text{“IF } X \neq 1 \text{ THEN } X := 1\text{”}, \\ \text{“}X := 2; X := X - 1\text{”}, \dots \}$$

Notation for Combining Sets of Commands

- Let c, c_1, c_2 etc. denote *sets* of commands
- Define:

$$c_1; \dots; c_n = \{C_1; \dots; C_n \mid C_1 \in c_1 \wedge \dots \wedge C_n \in c_n\}$$

$$\text{BEGIN VAR } V_1; \dots \text{ VAR } V_n; c \text{ END}$$

$$= \{\text{BEGIN VAR } V_1; \dots \text{ VAR } V_n; C \text{ END} \mid C \in c\}$$

$$\text{IF } S \text{ THEN } c_1 \text{ ELSE } c_2$$

$$= \{\text{IF } S \text{ THEN } C_1 \text{ ELSE } C_2 \mid C_1 \in c_1 \wedge C_2 \in c_2\}$$

$$\text{WHILE } S \text{ DO } c$$

$$= \{\text{WHILE } S \text{ DO } C \mid C \in c\}$$

- Wide spectrum language commands are sets of ordinary commands

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
- Note: the standard notation is $p_1 \sqsubseteq p_2$
- A program development takes us from the specification, through a series of mixed programs to (we hope) executable code:

$$\text{spec} \supseteq \text{mixed}_1 \supseteq \dots \supseteq \text{mixed}_n \supseteq \text{code}$$

Monotonicity

- Sets of commands are *monotonic* w.r.t. \supseteq

– if $c \supseteq c', c_1 \supseteq c'_1, \dots, c_n \supseteq c'_n$

– then:

$$\begin{array}{l} c_1; \dots; c_n \\ \text{BEGIN VAR } V_1; \dots \text{ VAR } V_n; c \text{ END} \\ \supseteq \\ \text{BEGIN VAR } V_1; \dots \text{ VAR } V_n; c' \text{ END} \\ \text{IF } S \text{ THEN } c_1 \text{ ELSE } c_2 \\ \text{WHILE } S \text{ DO } c \end{array} \supseteq \begin{array}{l} c'_1; \dots; c'_n \\ \text{IF } S \text{ THEN } c'_1 \text{ ELSE } c'_2 \\ \text{WHILE } S \text{ DO } c' \end{array}$$

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

6.4 Refinement Laws

SKIP Law

$$\begin{array}{c} \text{The SKIP Law} \\ [P, P] \supseteq \{\text{SKIP}\} \end{array}$$

- Derivation:

$$\begin{array}{l} C \in \{\text{SKIP}\} \Leftrightarrow C = \text{SKIP} \\ \Rightarrow \quad \{ \text{SKIP Axiom} \} \\ \quad \vdash [P] C [P] \\ \Leftrightarrow \quad \{ \text{Definition of } [P, P] \} \\ \quad C \in [P, P] \end{array}$$
- Examples: $[X = 1, X = 1] \supseteq \{\text{SKIP}\}$ $[T, T] \supseteq \{\text{SKIP}\}$ $[X = R + Y \times Q, X = R + Y \times Q] \supseteq \{\text{SKIP}\}$

Notational Convention

- Omit { and } around individual commands
- Skip law becomes: $[P, P] \supseteq \text{SKIP}$
- Examples become: $[X = 1, X = 1] \supseteq \text{SKIP}$ $[T, T] \supseteq \text{SKIP}$ $[X = R + Y \times Q, X = R + Y \times Q] \supseteq \text{SKIP}$

Assignment Law

$$\begin{array}{c} \text{The Assignment Law} \\ [P[E/V], P] \supseteq \{V := E\} \end{array}$$

- Derivation:

$$\begin{array}{l} C \in \{V := E\} \Leftrightarrow C = V := E \\ \Rightarrow \quad \{ \text{Assignment Axiom} \} \\ \quad \vdash [P[E/V]] C [P] \\ \Leftrightarrow \quad \{ \text{Definition of } [P[E/V], P] \} \\ \quad C \in [P[E/V], P] \end{array}$$
- 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

$$\begin{array}{c} \text{Precondition Weakening} \\ [P, Q] \supseteq [R, Q] \\ \text{provided } \vdash P \Rightarrow R \end{array}$$

$$\begin{array}{c} \text{Postcondition Strengthening} \\ [P, Q] \supseteq [P, R] \\ \text{provided } \vdash R \Rightarrow Q \end{array}$$

- We are now “weakening the precondition” and “strengthening the postcondition”
 - this is the opposite terminology to the Hoare rules
 - refinement rules are ‘backwards’

Derivation of Consequence Laws

- Derivation of Precondition Weakening:

$$\begin{aligned}
 & C \in [R, Q] \\
 \Leftrightarrow & \quad \{ \text{Definition of } [R, Q] \} \\
 & \vdash [R] C [Q] \\
 \Rightarrow & \quad \{ \text{Precondition Strengthening } \vdash P \Rightarrow R \} \\
 & \vdash [P] C [Q] \\
 \Leftrightarrow & \quad \{ \text{Definition of } [P, Q] \} \\
 & C \in [P, Q]
 \end{aligned}$$

- Derivation of Postcondition Strengthening:

$$\begin{aligned}
 & C \in [P, R] \\
 \Leftrightarrow & \quad \{ \text{Definition of } [P, R] \} \\
 & \vdash [P] C [R] \\
 \Rightarrow & \quad \{ \text{Postcondition Weakening } \vdash R \Rightarrow Q \} \\
 & \vdash [P] C [Q] \\
 \Leftrightarrow & \quad \{ \text{Definition of } [P, Q] \} \\
 & C \in [P, Q]
 \end{aligned}$$

Examples

- A previous example:

$$\begin{aligned}
 & [X = 1, X = 1] \\
 & \supseteq \{ \text{SKIP Law} \} \\
 & \text{SKIP}
 \end{aligned}$$

- An alternative refinement:

$$\begin{aligned}
 & [Y = 1, X = 1] \\
 & \supseteq \{ \text{Precondition Weakening } \vdash Y = 1 \Rightarrow 1 = 1 \} \\
 & [1 = 1, X = 1] \\
 & \supseteq \{ \text{Assignment Law} \} \\
 & X := 1
 \end{aligned}$$

- Another example:

$$\begin{aligned}
 & [T, R = X] \\
 & \supseteq \{ \text{Precondition Weakening } \vdash T \Rightarrow X = X \} \\
 & [X = X, R = X] \\
 & \supseteq \{ \text{Assignment Law} \} \\
 & R := X
 \end{aligned}$$

Derived Assignment Law**Derived Assignment Law**

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

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

- Derivation:

$$\begin{aligned}
& [P, Q] \\
& \supseteq \{ \text{Precondition Weakening } \vdash P \Rightarrow Q[E/V] \} \\
& [Q[E/V], Q] \\
& \supseteq \{ \text{Assignment Law} \} \\
& V := E
\end{aligned}$$

- Example:

$$\begin{aligned}
& [T, R = X] \\
& \supseteq \{ \text{Derived Assignment } \vdash T \Rightarrow X = X \} \\
& R := X
\end{aligned}$$

Sequencing**The Sequencing Law**

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

- Derivation of Sequencing Law:

$$\begin{aligned}
& C \supseteq [P, R]; [R, Q] \\
& \Leftrightarrow \{ \text{Definition of } c_1; c_2 \} \\
& C \in \{C_1; C_2 \mid C_1 \in [P, R] \wedge C_2 \in [R, Q]\} \\
& \Leftrightarrow \{ \text{Definition of } [P, R] \text{ and } [R, Q] \} \\
& C \in \{C_1; C_2 \mid \vdash [P] C_1 [R] \wedge \vdash [R] C_2 [Q]\} \\
& \Rightarrow \{ \text{Sequencing Rule} \} \\
& C \in \{C_1; C_2 \mid \vdash [P] C_1; C_2 [Q]\} \\
& \Rightarrow (\text{Definition of } [P, Q]) \\
& \vdash [P] C [Q] \\
& \Leftrightarrow C \in [P, Q]
\end{aligned}$$

Sequencing Example

$$\begin{aligned}
& [T, R = X \wedge Q = 0] \\
& \supseteq \{ \text{Sequencing Law} \} \\
& [T, R = X]; [R = X, R = X \wedge Q = 0] \\
& \supseteq \{ \text{Derived Assignment } \vdash T \Rightarrow X = X \} \\
& R := X; [R = X, R = X \wedge Q = 0] \\
& \supseteq \{ \text{Derived Assignment } \vdash R = X \Rightarrow R = X \wedge 0 = 0 \} \\
& R := X; Q := 0
\end{aligned}$$

6.5 Refining to Code

Creating Different Programs

- By applying the laws in a different way, we obtain different programs
- In the previous example, by using a different assertion with the sequencing law, we could create a program with the assignments reversed:

$$\begin{aligned}
& [T, R = X \wedge Q = 0] \\
& \supseteq \{ \text{Sequencing Law} \} \\
& [T, Q = 0]; [Q = 0, R = X \wedge Q = 0] \\
& \supseteq \{ \text{Derived Assignment } \vdash T \Rightarrow 0 = 0 \} \\
& Q := 0; [Q = 0, R = X \wedge Q = 0] \\
& \supseteq \{ \text{Derived Assignment } \vdash Q = 0 \Rightarrow X = X \wedge Q = 0 \} \\
& Q := 0; R := X
\end{aligned}$$

Inefficient Programs

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

$$\begin{aligned}
& [T, R = X \wedge Q = 0] \\
& \supseteq \{ \text{Sequencing} \} \\
& [T, R = X \wedge Q = 0]; [R = X \wedge Q = 0, R = X \wedge Q = 0] \\
& \supseteq \{ \text{as previous example} \} \\
& Q := 0; R := X; [R = X \wedge Q = 0, R = X \wedge Q = 0] \\
& \supseteq \{ \text{SKIP Law} \} \\
& Q := 0; R := X; \text{SKIP}
\end{aligned}$$

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:

$$\begin{aligned}
& [X = x \wedge Y = y, X = y \wedge Y = x] \\
& \supseteq \{ \text{Sequencing Law} \} \\
& [X = x \wedge Y = y, X = x \wedge Y = x]; \\
& [X = x \wedge Y = x, X = y \wedge Y = x] \\
& \supseteq \{ \text{Derived Assignment} \\
& \quad \vdash X = x \wedge Y = y \Rightarrow X = x \wedge X = x \} \\
& Y := X; [X = x \wedge Y = x, X = y \wedge Y = x] \\
& \supseteq \{ \text{Sequencing Law} \} \\
& Y := X; [X = x \wedge Y = x, Y = y \wedge Y = x]; \\
& \quad [Y = y \wedge Y = x, X = y \wedge Y = x] \\
& \supseteq \{ \text{Assignment Law} \} \\
& Y := X; [X = x \wedge Y = x, Y = y \wedge Y = x]; X := Y
\end{aligned}$$

6.6 More Refinement Laws

Blocks

The Block Law

$$[P, Q] \supseteq \text{BEGIN VAR } V_1; \dots V_n; [P, Q] \text{ END}$$

where none of the variables V_1, \dots, V_n occur in P or Q

- Example:

$$[X = x \wedge Y = y, X = y \wedge Y = x]$$

$$\supseteq \{ \text{Block Law} \}$$

$$\text{BEGIN VAR R; } [X = x \wedge Y = y, X = y \wedge Y = x] \text{ END}$$

$$\supseteq \{ \text{Sequencing Law and Derived Assignment} \}$$

$$\text{BEGIN VAR R; R := X; X := Y; Y := R END}$$

Conditionals

The Conditional Law

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

- 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!

Example

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

$$\supseteq \{ \text{Conditional Law} \}$$

$$\text{IF } X \geq Y$$

$$\text{THEN } [T \wedge X \geq Y, M = \max(X, Y)]$$

$$\text{ELSE } [T \wedge \neg(X \geq Y), M = \max(X, Y)]$$

$$\supseteq \{ \text{Derived Assignment} \}$$

$$\vdash T \wedge X \geq Y \Rightarrow X = \max(X, Y) \}$$

$$\text{IF } X \geq Y$$

$$\text{THEN } M := X$$

$$\text{ELSE } [T \wedge \neg(X \geq Y), M = \max(X, Y)]$$

$$\supseteq \{ \text{Derived Assignment} \}$$

$$\vdash T \wedge \neg(X \geq Y) \Rightarrow Y = \max(X, Y) \}$$

$$\text{IF } X \geq Y \text{ THEN } M := X \text{ ELSE } M := Y$$

WHILE

The WHILE Law

$$\begin{aligned} & [R, R \wedge \neg S] \supseteq \\ \text{WHILE } S \text{ DO } & [R \wedge S \wedge (E = n), R \wedge (E < n)] \\ & \text{provided } \vdash R \wedge S \Rightarrow E \geq 0 \end{aligned}$$

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

- Example:

$$\begin{aligned} & [X = R + Y \times Q \wedge Y > 0, X = R + Y \times Q \wedge Y > 0 \wedge \neg(Y \leq R)] \\ & \supseteq \{ \text{WHILE Law} \\ & \quad \vdash X = R + Y \times Q \wedge Y > 0 \wedge Y \leq R \Rightarrow R \geq 0 \} \\ \text{WHILE } Y \leq R \text{ DO} \\ & [X = R + Y \times Q \wedge Y > 0 \wedge Y \leq R \wedge R = n, \\ & X = R + Y \times Q \wedge Y > 0 \wedge R < n] \end{aligned}$$

6.7 Examples**Example**

$$\begin{aligned} & [Y > 0, X = R + Y \times Q \wedge R \leq Y] \\ & \supseteq \{ \text{Block Law} \} \\ \text{BEGIN} \\ & [Y > 0, X = R + Y \times Q \wedge R \leq Y] \\ \text{END} \\ & \supseteq \{ \text{Sequencing Law} \} \\ \text{BEGIN} \\ & [Y > 0, R = X \wedge Y > 0]; \\ & [R = X \wedge Y > 0, X = R + Y \times Q \wedge R \leq Y] \\ \text{END} \\ & \supseteq \{ \text{Derived Assignment} \\ & \quad \vdash Y > 0 \Rightarrow X = X \wedge Y > 0 \} \\ \text{BEGIN} \\ & R := X; \\ & [R = X \wedge Y > 0, X = R + Y \times Q \wedge R \leq Y] \\ \text{END} \end{aligned}$$

Example (Continued)

$$\begin{aligned} & \supseteq \{ \text{Sequencing Law} \} \\ \text{BEGIN} \\ & R := X; \\ & [R = X \wedge Y > 0, R = X \wedge Y > 0 \wedge Q = 0]; \\ & [R = X \wedge Y > 0 \wedge Q = 0, X = R + Y \times Q \wedge R \leq Y] \\ \text{END} \\ & \supseteq \{ \text{Derived Assignment} \\ & \quad \vdash R = X \wedge Y > 0 \Rightarrow R = X \wedge Y > 0 \wedge 0 = 0 \} \\ \text{BEGIN} \\ & R := X; \\ & Q := 0; \\ & [R = X \wedge Y > 0 \wedge Q = 0, X = R + Y \times Q \wedge R \leq Y] \\ \text{END} \end{aligned}$$

Example (Continued)

$$\supseteq \{ \text{Precondition Weakening} \\ \vdash R = X \wedge Y > 0 \wedge Q = 0 \Rightarrow \\ X = R + Y \times Q \wedge Y > 0 \}$$

```

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

```

$$\supseteq \{ \text{Postcondition Strengthening} \\ \vdash X = R + Y \times Q \wedge Y > 0 \wedge \neg(Y \leq R) \Rightarrow \\ X = R + Y \times Q \wedge R \leq Y \}$$

```

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

```

Example (Continued)

$$\supseteq \{ \text{WHILE Law} \\ \vdash X = R + Y \times Q \wedge Y > 0 \wedge Y \leq R \Rightarrow R \geq 0 \}$$

```

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 (Continued)

$$\supseteq \{ \text{Block Law} \}$$

```

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

```

Example (Continued)

$$\supseteq \{ \text{Sequence Law} \}$$

```

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 (Continued)

```

 $\supseteq \{ \text{Assignment Law} \}$ 
BEGIN
  R := X;
  Q := 0;
  WHILE Y  $\leq$  R DO
    BEGIN
      [X = R + Y  $\times$  Q  $\wedge$  Y > 0  $\wedge$  Y  $\leq$  R  $\wedge$  R = n,
       X = (R - Y) + Y  $\times$  Q  $\wedge$  Y > 0  $\wedge$  (R - Y) < n];
      R := R - Y
    END
  END
END

```

Example (Continued)

```

 $\supseteq \{ \text{Derived Assignment} \}$ 
 $\vdash X = R + Y \times Q \wedge Y > 0 \wedge Y \leq R \wedge R = n \Rightarrow$ 
 $X = (R - Y) + Y \times (Q + 1) \wedge Y > 0 \wedge (R - Y) < n\}$ 
BEGIN
  R := X;
  Q := 0;
  WHILE Y  $\leq$  R DO
    BEGIN
      Q := Q + 1;
      R := R - Y
    END
  END
END

```

More Notation

- The notation:

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

will be used to abbreviate:

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

- Brackets around specifications $\{C\}$ can be omitted
- If C is a set of commands, then:

$$R := X; C$$

abbreviates:

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

Example

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

```

[Y > 0,  $\mathcal{I} \wedge R \leq Y$ ]
 $\supseteq \{ \text{Sequencing Law} \}$ 
[Y > 0, R = X  $\wedge$  Y > 0,  $\mathcal{I} \wedge R \leq Y$ ]
 $\supseteq \{ \text{Assignment Law} \}$ 
R := X;
[R = X  $\wedge$  Y > 0,  $\mathcal{I} \wedge R \leq Y$ ]
 $\supseteq \{ \text{Sequencing Law} \}$ 
R := X;

```

$$\begin{aligned}
& [R = X \wedge Y > 0, R = X \wedge Y > 0 \wedge Q = 0, \mathcal{I} \wedge R \leq Y] \\
& \supseteq \{ \text{Assignment Law} \} \\
& R := X; \\
& Q := 0; \\
& [R = X \wedge Y > 0 \wedge Q = 0, \mathcal{I} \wedge R \leq Y]
\end{aligned}$$
Example (Continued)

$$\begin{aligned}
& \supseteq \{ \text{Precondition Weakening} \} \\
& R := X; \\
& Q := 0; \\
& [\mathcal{I} \wedge Y > 0, \mathcal{I} \wedge R \leq Y] \\
& \supseteq \{ \text{Postcondition Strengthening} \} \\
& R := X; \\
& Q := 0; \\
& [\mathcal{I} \wedge Y > 0, \mathcal{I} \wedge Y > 0 \wedge \neg(Y \leq R)] \\
& \supseteq \{ \text{WHILE Law} \} \\
& R := X; \\
& Q := 0; \\
& \text{WHILE } Y \leq R \text{ DO} \\
& \quad [\mathcal{I} \wedge Y > 0 \wedge Y \leq R \wedge R = n, \mathcal{I} \wedge Y > 0 \wedge R < n]
\end{aligned}$$
Example (Continued)

$$\begin{aligned}
& \supseteq \{ \text{Sequencing Law} \} \\
& R := X; \\
& Q := 0; \\
& \text{WHILE } Y \leq R \text{ DO} \\
& \quad [\mathcal{I} \wedge Y > 0 \wedge Y \leq R \wedge R = n, \\
& \quad X = (R - Y) + (Y \times Q) \wedge Y > 0 \wedge (R - Y) < n, \\
& \quad \mathcal{I} \wedge Y > 0 \wedge R < n] \\
& \supseteq \{ \text{Derived Assignment} \} \\
& R := X; \\
& Q := 0; \\
& \text{WHILE } Y \leq R \text{ DO} \\
& \quad [\mathcal{I} \wedge Y > 0 \wedge Y \leq R \wedge R = n, \\
& \quad X = (R - Y) + (Y \times Q) \wedge Y > 0 \wedge (R - Y) < n]; \\
& \quad R := R - Y
\end{aligned}$$
Example (Continued)

$$\begin{aligned}
& \supseteq \{ \text{Derived Assignment} \} \\
& R := X; \\
& Q := 0; \\
& \text{WHILE } Y \leq R \text{ DO} \\
& \quad Q := Q + 1; \\
& \quad R := R - Y
\end{aligned}$$

- 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:

$$[X = n \wedge X > 0, Y = n!]$$

6.8 Data Refinement

Data Refinement

- So far we have given laws to refine commands
- 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
- Details can be found in Morgan's book