Local Reasoning and Separation Logic ✓

- Want to just reason about just those locations being modified
  - assume all other locations unchanged
- Solution: separation logic
  - small and forward assignment axioms + separating conjunction
  - small means just applies to fragment of heap (*footprint*)
  - forward means Floyd-style forward rules that support symbolic execution
  - non-faulting semantics of Hoare triples
  - symbolic execution used by tools like smallfoot
  - separating conjunction solves frame problem - like rule of constancy for heap
- Need new kinds of assertions to state separation logic axioms

Summary of pointer manipulating axioms

Store assignment axiom

$$\vdash \{V \doteq v\}\, V \texttt{:=} E\, \{V \doteq E[v/V]\}$$

where v is an auxiliary variable not occurring in E.

Fetch assignment axiom

$$\vdash \{(V = v_1) \wedge E \mapsto v_2\}\, V \texttt{:=[}E\texttt{]}\, \{(V = v_2) \wedge E[v_1/V] \mapsto v_2\}$$

where v_1, v_2 are auxiliary variables not occurring in E.

Heap assignment axiom

$$\vdash \{E \mapsto \_\}\, \texttt{[}E\texttt{]:=}F\, \{E \mapsto F\}$$

Allocation assignment axiom

$$\vdash \{V \doteq v\}\, V \texttt{:=cons(}E_1, \ldots, E_n\texttt{)}\, \{V \mapsto E_1[v/V], \ldots, E_n[v/V]\}$$

where v is an auxiliary variable not equal to V or occurring in E_1,. . .,E_n

Dispose axiom

$$\vdash \{E \mapsto \_\}\, \texttt{dispose(}E\texttt{)}\, \{\texttt{emp}\}$$

The frame rule ✓

The rule of constancy

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

where no variable modified by C occurs free in R.

- **Rule of constancy is not valid for heap assignments**

  $\vdash \{\mathtt{X} \mapsto \_\}\ \mathtt{[X]:=0}\ \{\mathtt{X} \mapsto 0\}$

  but not (c.f. arrays)

  $\{\mathtt{X} \mapsto \_ \ \wedge\ \mathtt{Y} \mapsto 1\}\ \mathtt{[X]:=0}\ \{\mathtt{X} \mapsto 0 \ \wedge\ \mathtt{Y} \mapsto 1\}$

  as `X` and `Y` could initially both be bound to the same location

- **Using $\star$ instead of $\wedge$ forces `X` and `Y` to point to different locations**

The frame rule

$$\frac{\vdash \{P\}\,C\,\{Q\}}{\vdash \{P \star R\}\,C\,\{Q \star R\}}$$

where no variable modified by C occurs free in R.

- **Soundness a little tricky due to faulting**

Compound command rules ✓

- **Following rules apply to both Hoare logic and separation logic**

The sequencing rule

$$\frac{\vdash \{P\}\ C_1\ \{Q\}, \qquad \vdash \{Q\}\ C_2\ \{R\}}{\vdash \{P\}\ C_1 ; C_2\ \{R\}}$$

The conditional rule

$$\frac{\vdash \{P \wedge S\}\ C_1\ \{Q\}, \qquad \vdash \{P \wedge \neg S\}\ C_2\ \{Q\}}{\vdash \{P\}\ \texttt{IF}\ S\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2\ \{Q\}}$$

The `WHILE`-rule

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

- **For separation logic, need to think about faulting**

$\{$*contents of pointers* $\mathtt{X}$ *and* $\mathtt{Y}$ *are equal*$\}$ $\mathtt{X:=[X]\,;\ Y:=[Y]}$ $\{\mathtt{X} = \mathtt{Y}\}$ ✓

- **Proof:**

$\vdash \{(\mathtt{X} = x) \wedge \mathtt{X} \mapsto v\}\ \mathtt{X:=[X]}\ \{(\mathtt{X} = v) \wedge x \mapsto v\}$ fetch assignment axiom

$\vdash \{(\mathtt{Y} = y) \wedge \mathtt{Y} \mapsto v\}\ \mathtt{Y:=[Y]}\ \{(\mathtt{Y} = v) \wedge y \mapsto v\}$ fetch assignment axiom

$\vdash \{((\mathtt{X} = x) \wedge \mathtt{X} \mapsto v) \star ((\mathtt{Y} = y) \wedge \mathtt{Y} \mapsto v)\}$ frame rule
$\mathtt{X:=[X]}$
$\{((\mathtt{X} = v) \wedge x \mapsto v) \star (((\mathtt{Y} = y) \wedge \mathtt{Y} \mapsto v))\}$

$\vdash \{((\mathtt{Y} = y) \wedge \mathtt{Y} \mapsto v) \star ((\mathtt{X} = v) \wedge x \mapsto v)\}$ frame rule
$\mathtt{Y:=[Y]}$
$\{((\mathtt{Y} = v) \wedge y \mapsto v) \star ((\mathtt{X} = v) \wedge x \mapsto v)\}$

$\vdash \{((\mathtt{X} = x) \wedge \mathtt{X} \mapsto v) \star ((\mathtt{Y} = y) \wedge \mathtt{Y} \mapsto v)\}$ sequencing rule and commutativity of $\star$
$\mathtt{X:=[X]\,;Y:=[Y]}$
$\{((\mathtt{X} = v) \wedge x \mapsto v) \star ((\mathtt{Y} = v) \wedge y \mapsto v)\}$

$\vdash \{\exists v\ x\ y.\ ((\mathtt{X} = x) \wedge \mathtt{X} \mapsto v) \star ((\mathtt{Y} = y) \wedge \mathtt{Y} \mapsto v)\}$ exists-introduction (3 times)
$\mathtt{X:=[X]\,;Y:=[Y]}$
$\{\exists v\ x\ y.\ ((\mathtt{X} = v) \wedge x \mapsto v) \star ((\mathtt{Y} = v) \wedge y \mapsto v)\}$

$\vdash \{\exists v.\ \mathtt{X} \mapsto v \star \mathtt{Y} \mapsto v\}\ \mathtt{X:=[X]\,;\ Y:=[Y]}\ \{\mathtt{X} = \mathtt{Y}\}$ rules of consequence (see next slide)

Current research and the future ✓

- **Extending separation logic to cover practical language features**
  - **various concurrency idioms**
  - **objects**
- **Building tools to mechanise separation logic**
  - **much work on** *shape analysis*, **e.g.:**

    $\{\exists x.\ \texttt{list}\ x\ X\}$
    `Y:=nil;`
    `WHILE ¬(X = nil) DO (Z:=[X+1]; [X+1]:=Y; Y:=X; X:=Z)`
    $\{\exists x.\ \texttt{list}\ x\ Y\}$

    automatically finds memory usage errors
- **Finally, something to think about:**
  should we be verifying code in old fashioned languages (pramatism)
  or creating new methods to create correct software (idealism)?

"The tension between idealism and pragmatism is as profound (almost) as that between good and evil (and just as pervasive)."
[Tony Hoare]

New Topic: 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**

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

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

$C ::=$ `SKIP` **(does nothing, `SKIP`-Axiom is** $\vdash [P]$ `SKIP` $[P]$**)**
$\mid V$ `:=` E
$\mid C_1$ `;` C_2
$\mid$ `IF` B `THEN` C_1 `ELSE` C_2
$\mid$ `BEGIN VAR` V_1 `;` .. `VAR` V_1 `;` C `END`
$\mid$ `WHILE` B `DO` C
$\mid [P,\ Q]$

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**

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

- **Don't confuse use of $\{\cdots\}$ as set brackets and in Hoare triples**

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:**

$$c_1\,;\ \cdots\ ;c_n = \{\ C \mid \exists C_1\ \cdots\ C_n.\ C = C_1\,;\ \cdots\ ;C_n \wedge C_1 \in c_1 \wedge \cdots \wedge C_n \in c_n\ \}$$

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

$$\texttt{BEGIN VAR}\ V_1\texttt{;}\ \cdots\ \texttt{VAR}\ V_n\texttt{;}\ c\ \texttt{END} = \{\ \texttt{BEGIN VAR}\ V_1\texttt{;}\ \cdots\ \texttt{VAR}\ V_n\texttt{;}\ C\ \texttt{END} \mid C \in c\ \}$$

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

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

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

$$spec \supseteq mixed_1 \supseteq \ldots \supseteq mixed_n \supseteq code$$

Skip Law ✓

The Skip Law

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

- **Derivation:**

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

- **Examples**

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

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

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

Notational Convention

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

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

- **Examples become:**

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

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

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

Assignment Law ✓

The Assignment Law

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

- **Derivation**

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

- **Examples**

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

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

Laws of Consequence ✓

Precondition Weakening

$$[P,\ \ Q] \supseteq [R,\ \ Q]$$
$$\textbf{provided} \vdash\ \ P \Rightarrow R$$

Postcondition Strengthening

$$[P,\ \ Q] \supseteq [P,\ \ R]$$
$$\textbf{provided} \vdash\ \ R \Rightarrow Q$$

- **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'**

Derived Assignment Law ✓

Derived Assignment Law

$$[P,\ \ Q] \supseteq \{V :=\! E\}$$
$$\textbf{provided } \vdash\ P\ \Rightarrow\ Q[E/V]$$

- **Derivation**

  $[P,\ \ Q]$

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

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

  $\supseteq$ (Assignment)

  $V :=\! E$

- **Example**

  [T, R=X]

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

  R := X

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]
⊇ (Sequencing)
[T,  Q=0] ; [Q=0,  R=X∧Q=0]
⊇ (Derived Assignment  ⊢  T ⇒ 0=0)
Q:=0; [Q=0,  R=X∧Q=0]
⊇ (Derived Assignment  ⊢  Q=0 ⇒ X=X ∧ 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];        (no way to refine this!)
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**
- **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!**

While ✓

The While Law

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

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**

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

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

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

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

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

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**

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

abbreviates

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

Exercise: check the refinement on this slide ✓

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

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

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