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

\[
E ::= N \mid V \mid E_1 + E_2 \mid E_1 - E_2 \mid E_1 \times E_2 \mid \ldots
\]
\[
B ::= T \mid F \mid E_1 = E_2 \mid E_1 \leq E_2 \mid \ldots
\]
\[
C ::= \text{SKIP} \mid V := E \mid C_1 ; C_2 \mid \text{IF } B \text{ THEN } C_1 \text{ ELSE } C_2 \mid \text{BEGIN VAR } V_1 ; \ldots \text{ VAR } V_n ; C \text{ END} \mid \text{WHILE } B \text{ DO } C \mid [P, Q]
\]

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:
  \[
  R := X;
  Q := 0;
  \text{WHILE } Y \leq R \text{ DO }
  \]
  \[
  | X = R + Y \times Q \land Y \leq R, X = R + Y \times Q |
  \]
  - Think of a specification as defining the set of implementations:
    \([P, Q] = \{C \mid \vdash [P] C [Q]\}\)
  - For example:
    \([T, X = 1] = \{"X := 1", \text{"IF } X \neq 1 \text{ THEN } X := 1", \text{"X := 2; X := X - 1",} \ldots\}\)

Notation for Combining Sets of Commands

- Let \(c, c_1, c_2\) etc. denote sets of commands
- Define:
\[ c_1; \ldots; c_n = \{ C_1; \ldots; C_n \mid C_1 \in c_1 \land \cdots \land C_n \in c_n \} \]

\[
\begin{align*}
\text{BEGIN VAR } & V_1; \ldots \text{ VAR } V_n; c \text{ END} \\
& = \{ \text{BEGIN VAR } V_1; \ldots \text{ VAR } V_n; C \text{ END} \mid C \in c \}
\end{align*}
\]

\[
\begin{align*}
\text{IF } S \text{ THEN } c_1 \text{ ELSE } c_2 &= \{ \text{IF } S \text{ THEN } C_1 \text{ ELSE } C_2 \mid C_1 \in c_1 \land C_2 \in c_2 \}
\end{align*}
\]

\[
\begin{align*}
\text{WHILE } S \text{ DO } c &= \{ \text{WHILE } S \text{ DO } C \mid C \in c \}
\end{align*}
\]

- 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:
  \[ \text{spec} \supseteq \text{mixed}_1 \supseteq \cdots \supseteq \text{mixed}_n \supseteq \text{code} \]

**Monotonicity**

- Sets of commands are monotonic w.r.t. \( \supseteq \)
  - if \( c \supseteq c' \), \( c_1 \supseteq c'_1 \), \ldots, \( c_n \supseteq c'_n \)
  - then:
    \[
    \begin{align*}
    c_1; \ldots; c_n &\supseteq c'_1; \ldots; c'_n \\
    \text{BEGIN VAR } V_1; \ldots \text{ VAR } V_n; c \text{ END} &\supseteq \text{BEGIN VAR } V_1; \ldots \text{ VAR } V_n; c' \text{ END} \\
    \text{IF } S \text{ THEN } c_1 \text{ ELSE } c_2 &\supseteq \text{IF } S \text{ THEN } c'_1 \text{ ELSE } c'_2 \\
    \text{WHILE } S \text{ DO } c &\supseteq \text{WHILE } S \text{ DO } c'
    \end{align*}
    \]
- Monotonicity shows that a command can be refined by separately refining its constituents
- Laws of refinement now follow
6.4 Refinement Laws

**SKIP Law**

\[
[P, P] \supseteq \{\text{SKIP}\}
\]

- **Derivation:**
  \[C \in \{\text{SKIP}\} \iff C = \text{SKIP} \implies \{\text{SKIP Axiom}\} \vdash [P] C [P] \iff \{\text{Definition of } [P, P]\} \implies C \in [P, P]\]

- **Examples:**
  \[\{X = 1, X = 1\} \supseteq \{\text{SKIP}\} \quad \{T, T\} \supseteq \{\text{SKIP}\} \quad \{X = R + Y \times Q, X = R + Y \times Q\} \supseteq \{\text{SKIP}\}\]

**Notational Convention**

- Omit \{ and \} around individual commands
- Skip law becomes: \([P, P]\) \supseteq \text{SKIP}
- Examples become: \([X = 1, X = 1]\) \supseteq X := Y \quad \{X + 1 = n + 1, X = n + 1\} \supseteq X := X + 1

**Assignment Law**

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

- **Derivation:**
  \[C \in \{V := E\} \iff C = V := E \implies \{\text{Assignment Axiom}\} \vdash [P[E/V]] C [P] \iff \{\text{Definition of } [P[E/V], P]\} \implies C \in [P[E/V], P]\]

- **Examples (using bracket omitting convention):**
  \[\{Y = 1, X = 1\} \supseteq X := Y \quad \{X + 1 = n + 1, X = n + 1\} \supseteq X := X + 1\]

**Laws of Consequence**

**Precondition Weakening**

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

provided \(\vdash P \Rightarrow R\)

**Postcondition Strengthening**

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

provided \(\vdash R \Rightarrow Q\)
• 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:
  \[ C \in [R, Q] \]
  \[ \iff \]
  \[ \vdash [R] C [Q] \]
  \[ \Rightarrow \]
  \[ \vdash [P] C [Q] \]
  \[ \iff \]
  \[ \vdash \text{Precondition Strengthening} \vdash P \Rightarrow R \]
  \[ \vdash [P] C [Q] \]
  \[ \iff \]
  \[ \vdash \text{Definition of } [P, Q] \]
  \[ C \in [P, Q] \]

• Derivation of Postcondition Strengthening:
  \[ C \in [P, R] \]
  \[ \iff \]
  \[ \vdash [P] C [R] \]
  \[ \Rightarrow \]
  \[ \vdash [P] C [Q] \]
  \[ \iff \]
  \[ \vdash \text{Postcondition Weakening} \vdash R \Rightarrow Q \]
  \[ \vdash [P] C [Q] \]
  \[ \iff \]
  \[ \vdash \text{Definition of } [P, Q] \]
  \[ C \in [P, Q] \]

Examples

• A previous example:
  \[ [X = 1, X = 1] \]
  \[ \supseteq \{ \text{SKIP Law} \} \]
  SKIP

• An alternative refinement:
  \[ [Y = 1, X = 1] \]
  \[ \supseteq \{ \text{Precondition Weakening} \vdash Y = 1 \Rightarrow 1 = 1 \} \]
  \[ [1 = 1, X = 1] \]
  \[ \supseteq \{ \text{Assignment Law} \} \]
  \[ X := 1 \]

• Another example:
  \[ [T, R = X] \]
  \[ \supseteq \{ \text{Precondition Weakening} \vdash X = X \} \]
  \[ [X = X, R = X] \]
  \[ \supseteq \{ \text{Assignment Law} \} \]
  \[ R := X \]
Derived Assignment Law

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

provided \( \vdash P \Rightarrow Q[E/V] \)

- Derivation:
  \[
  [P, Q]
  \supseteq \{ \text{Precondition Weakening} \vdash P \Rightarrow Q[E/V] \}
  [Q[E/V], Q]
  \supseteq \{ \text{Assignment Law} \}
  V := E
  \]

- Example:
  \[
  [T, R = X]
  \supseteq \{ \text{Derived Assignment} \vdash T \Rightarrow X = X \}
  R := X
  \]

Sequencing

The Sequencing Law

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

- Derivation of Sequencing Law:
  \[
  C \supseteq [P, R]; [R, Q]
  \iff
  \{ \text{Definition of } c_1; c_2 \}
  C \in \{ C_1; C_2 \mid C_1 \in [P, R] \land C_2 \in [R, Q] \}
  \iff
  \{ \text{Definition of } [P, R] \text{ and } [R, Q] \}
  C \in \{ C_1; C_2 \mid \vdash [P] C_1 [R] \land \vdash [R] C_2 [Q] \}
  \iff
  \{ \text{Sequencing Rule} \}
  C \in \{ C_1; C_2 \mid \vdash [P] C_1; C_2 [Q] \}
  \iff
  (\text{Definition of } [P, Q])
  \vdash [P] C [Q]
  \iff
  C \in [P, Q]
  \]

Sequencing Example

\[
[T, R = X \land Q = 0]
\supseteq \{ \text{Sequencing Law} \}
[T, R = X]; [R = X, R = X \land Q = 0]
\supseteq \{ \text{Derived Assignment} \vdash T \Rightarrow X = X \}
R := X; [R = X, R = X \land Q = 0]
\supseteq \{ \text{Derived Assignment} \vdash R = X \Rightarrow R = X \land 0 = 0 \}
R := X; Q := 0
\]
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:

\[
[T, R = X \land Q = 0] \supseteq \{ \text{Sequencing Law} \}
\]

\[
[T, Q = 0]; [Q = 0, R = X \land Q = 0] \supseteq \{ \text{Derived Assignment} \vdash T \Rightarrow 0 = 0 \}
\]

\[
Q := 0; [Q = 0, R = X \land Q = 0] \supseteq \{ \text{Derived Assignment} \vdash Q = 0 \Rightarrow X = X \land Q = 0 \}
\]

\[
Q := 0; R := X
\]

Inefficient Programs

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

- Example:

\[
[T, R = X \land Q = 0] \supseteq \{ \text{Sequencing} \}
\]

\[
[T, R = X \land Q = 0]; [R = X \land Q = 0, R = X \land Q = 0] \supseteq \{ \text{as previous example} \}
\]

\[
Q := 0; R := X; [R = X \land Q = 0, R = X \land Q = 0] \supseteq \{ \text{SKIP Law} \}
\]

\[
Q := 0; R := X; \text{SKIP}
\]

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:

\[
[X = x \land Y = y, X = y \land Y = x] \supseteq \{ \text{Sequencing Law} \}
\]

\[
[X = x \land Y = y, X = x \land Y = x] \supseteq \{ \text{Derived Assignment} \vdash X = x \land X = x \}
\]

\[
Y := X; [X = x \land Y = x, X = y \land Y = x] \supseteq \{ \text{Sequencing Law} \}
\]

\[
Y := X; [X = x \land Y = x, Y = y \land Y = x] \supseteq \{ \text{Assignment Law} \}
\]

\[
Y := X; [X = x \land Y = x, Y = y \land Y = x]; X := Y
\]
6.6 More Refinement Laws

Blocks

The Block Law

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

where none of the variables \( V_1, \ldots, V_n \) occur in \( P \) or \( Q \)

- Example:

\[ [X = x \land Y = y, X = y \land Y = x] \supseteq \{ \text{Block Law} \} \]

BEGIN VAR R; \[X = x \land Y = y, X = y \land Y = x]\] END

\[ \supseteq \{ \text{Sequencing Law and Derived Assignment} \} \]

BEGIN VAR R; \( R := X; X := Y; Y := R \) END

Conditionals

The Conditional Law

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

- The Conditional Law can be used to refine \textit{any} specification and \textit{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

\[ [T, M = \max(X, Y)] \supseteq \{ \text{Conditional Law} \} \]

IF \( X \geq Y \)
THEN \[T \land X \geq Y, M = \max(X, Y)]\]
ELSE \[T \land \neg(X \geq Y), M = \max(X, Y)]\]
\[ \supseteq \{ \text{Derived Assignment} \}
\[ \vdash T \land X \geq Y \Rightarrow X = \max(X, Y) \} \]

IF \( X \geq Y \)
THEN M := X
ELSE \[T \land \neg(X \geq Y), M = \max(X, Y)]\]
\[ \supseteq \{ \text{Derived Assignment} \}
\[ \vdash T \land \neg(X \geq Y) \Rightarrow Y = \max(X, Y) \} \]

IF \( X \geq Y \) THEN M := X ELSE M := Y

WHILE
The WHILE Law

\[ [R, R \land \neg S] \supset\text{WHILE } S \text{ DO } [R \land S \land (E = n), R \land (E < n)] \]

provided \( \vdash R \land S \Rightarrow E \geq 0 \)

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

6.7 Examples

Example

\[ [Y > 0, X = R + Y \times Q \land R \leq Y] \]

\( \supset \) \{ Block Law \}
BEGIN
\[ [Y > 0, X = R + Y \times Q \land R \leq Y] \]
END
\( \supset \) \{ Sequencing Law \}
BEGIN
\[ [Y > 0, R = X \land Y > 0];\]
\[ [R = X \land Y > 0, X = R + Y \times Q \land R \leq Y] \]
END
\( \supset \) \{ Derived Assignment \}
BEGIN
\[ R := X;\]
\[ R = X \land Y > 0, X = R + Y \times Q \land R \leq Y] \]
END

Example (Continued)

\( \supset \) \{ Sequencing Law \}
BEGIN
\[ R := X;\]
\[ R = X \land Y > 0, R = X \land Y > 0 \land Q = 0];\]
\[ R = X \land Y > 0 \land Q = 0, X = R + Y \times Q \land R \leq Y] \]
END
\( \supset \) \{ Derived Assignment \}
BEGIN
\[ R := X;\]
\[ R = X \land Y > 0 \Rightarrow R = X \land Y > 0 \land 0 = 0 \}
BEGIN
\[ R := X;\]
\[ Q := 0;\]
\[ R = X \land Y > 0 \land Q = 0, X = R + Y \times Q \land R \leq Y] \]
END
Example (Continued)
\[\{\text{Precondition Weakening}\}
\]

\[\Gamma R = X \land Y > 0 \land Q = 0 \Rightarrow
X = R + Y \times Q \land Y > 0\]

\begin{verbatim}
BEGIN
R := X;
Q := 0;
[X = R + Y \times Q \land Y > 0, X = R + Y \times Q \land R \leq Y]
\end{verbatim}

\[\{\text{Postcondition Strengthening}\}
\]

\[\Gamma X = R + Y \times Q \land Y > 0 \land \neg (Y \leq R) \Rightarrow
X = R + Y \times Q \land R \leq Y\]

\begin{verbatim}
BEGIN
R := X;
Q := 0;
[X = R + Y \times Q \land Y > 0, X = R + Y \times Q \land Y > 0 \land \neg (Y \leq R)]
\end{verbatim}

Example (Continued)
\[\{\text{WHILE Law}\}
\]

\[\Gamma X = R + Y \times Q \land Y > 0 \land Y \leq R \Rightarrow R \geq 0\]

\begin{verbatim}
BEGIN
R := X;
Q := 0;
WHILE Y \leq R DO
[X = R + Y \times Q \land Y > 0 \land Y \leq R \land R = n,
X = R + Y \times Q \land Y > 0 \land R < n]
\end{verbatim}

Example (Continued)
\[\{\text{Block Law}\}
\]

\begin{verbatim}
BEGIN
R := X;
Q := 0;
WHILE Y \leq R DO
BEGIN
[X = R + Y \times Q \land Y > 0 \land Y \leq R \land R = n,
X = R + Y \times Q \land Y > 0 \land R < n]
END
\end{verbatim}

Example (Continued)
\[\{\text{Sequence Law}\}
\]

\begin{verbatim}
BEGIN
R := X;
Q := 0;
WHILE Y \leq R DO
BEGIN
[X = R + Y \times Q \land Y > 0 \land Y \leq R \land R = n,
X = (R - Y) + Y \times Q \land Y > 0 \land (R - Y) < n];
[X = (R - Y) + Y \times Q \land Y > 0 \land (R - Y) < n,
X = R + Y \times Q \land Y > 0 \land R < n]
END
\end{verbatim}
Example (Continued)

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

Example (Continued)

\[ \{ \text{Derived Assignment} \} \]
BEGIN
R := X;
Q := 0;
WHILE Y \leq R DO
BEGIN
Q := Q + 1;
R := R - Y
END
END

More Notation

- The notation:
  \[ [P_1, P_2, P_3, \ldots, P_{n-1}, P_n] \]
will be used to abbreviate:
  \[ [P_1, P_2]; [P_2, P_3]; \ldots; [P_{n-1}, P_n] \]
- Brackets around specifications \{C\} can be omitted
- If \( C \) is a set of commands, then:
  R := X; C
    abbreviates:
    \[ \{R := X\}; C \]

Example

- Let \( I \) stand for \( X = R + (Y \times Q) \), then:
  \[ [Y > 0, I \land R \leq Y] \]
  \[ \{ \text{Sequencing Law} \} \]
  \[ [Y > 0, R = X \land Y > 0, I \land R \leq Y] \]
  \[ \{ \text{Assignment Law} \} \]
  R := X;
  \[ [R = X \land Y > 0, I \land R \leq Y] \]
  \[ \{ \text{Sequencing Law} \} \]
  R := X;
\[ R = X \land Y > 0, R = X \land Y > 0 \land Q = 0, I \land R \leq Y \]
\[ \supset \{ \text{Assignment Law} \} \]
\[ R := X; \]
\[ Q := 0; \]
\[ |R = X \land Y > 0 \land Q = 0, I \land R \leq Y| \]

**Example (Continued)**
\[ \supset \{ \text{Precondition Weakening} \} \]
\[ R := X; \]
\[ Q := 0; \]
\[ |I \land Y > 0, I \land R \leq Y| \]
\[ \supset \{ \text{Postcondition Strengthening} \} \]
\[ R := X; \]
\[ Q := 0; \]
\[ |I \land Y > 0, I \land Y > 0 \land \neg (Y \leq R)| \]
\[ \supset \{ \text{WHILE Law} \} \]
\[ R := X; \]
\[ Q := 0; \]
\[ \text{WHILE } Y \leq R \text{ DO } \]
\[ |I \land Y > 0 \land Y \leq R \land R = n, I \land Y > 0 \land R < n| \]

**Example (Continued)**
\[ \supset \{ \text{Sequencing Law} \} \]
\[ R := X; \]
\[ Q := 0; \]
\[ \text{WHILE } Y \leq R \text{ DO } \]
\[ |I \land Y > 0 \land Y \leq R \land R = n, \]
\[ X = (R - Y) + (Y \times Q) \land Y > 0 \land (R - Y) < n, \]
\[ I \land Y > 0 \land R < n| \]
\[ \supset \{ \text{Derived Assignment} \} \]
\[ R := X; \]
\[ Q := 0; \]
\[ \text{WHILE } Y \leq R \text{ DO } \]
\[ |I \land Y > 0 \land Y \leq R \land R = n, \]
\[ X = (R - Y) + (Y \times Q) \land Y > 0 \land (R - Y) < n; \]
\[ R := R - Y \]

**Example (Continued)**
\[ \supset \{ \text{Derived Assignment} \} \]
\[ R := X; \]
\[ Q := 0; \]
\[ \text{WHILE } Y \leq R \text{ DO } \]
\[ Q := Q + 1; \]
\[ R := R - Y \]

- **Exercise:** Develop a factorial program from the specification:

\[ |X = n \land X > 0, Y = n!| \]
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