Predicate Transformers

(Based on [Dijkstra 1976; Gries 1981; Morgan 1994])

Yih-Kuen Tsay

Department of Information Management
National Taiwan University

Basic Idea

- The execution of a sequential program, if terminating, transforms the initial state into some final state.
- If, for any given postcondition, we know

  > *the weakest precondition that guarantees termination of the program in a state satisfying the postcondition,*

  then we have fully understood the meaning of the program.

Note: the weakest precondition is the weakest in the sense that it identifies all the desired initial states and nothing else.

The Predicate Transformer *wp*

- For a program S and a predicate (or an assertion) Q, let $wp(S, Q)$ denote the aforementioned weakest precondition.
- Therefore, we can see a program as a *predicate transformer* $wp(S, \cdot)$, transforming a postcondition Q (a predicate) into its weakest precondition $wp(S, Q)$.
- If the execution of S starts in a state satisfying $wp(S, Q)$, it is guaranteed to terminate and result in a state satisfying Q.

Note: there is a weaker variant of *wp*, called *wlp* (weakest liberal precondition), which is defined almost identical to *wp* except that termination is not guaranteed.

Notational Conventions

- $\Rightarrow$ vs. $\rightarrow$
  - $A \Rightarrow B$ (A entails B) states a relation between two formulae A and B: in every state, if A is true then B is true.
  - $A \rightarrow B$ is a formula. When "$A \rightarrow B$" stands alone, it usually means $A \rightarrow B$ is true in every state (model).
- $\equiv$ vs. $\leftrightarrow$
  - $A \equiv B$ (A is equivalent to B) states a relation between two formulae A and B: in every state, if A is true if and only if B is true.
  - $A \leftrightarrow B$ is a formula. When "$A \leftrightarrow B$" stands alone, it usually means $A \leftrightarrow B$ is true in every state (model).

Hoare Triples in Terms of *wp*

- When total correctness is meant, $\{P\}\ S\ \{Q\}$ can be understood as saying $P \Rightarrow wp(S, Q)$.
- In fact, with a suitable formal definition, *wp* provides a semantic foundation for the Hoare logic.
- The precondition P here may be as weak as $wp(S, Q)$, but often a stronger and easier-to-find P is all that is needed.

Properties of *wp*

Fundamental Properties (Axioms):

- **Law of the Excluded Miracle**: $wp(S, false) \equiv false$
- **Distributivity of Conjunction**:
  $wp(S, Q_1) \wedge wp(S, Q_2) \equiv wp(S, Q_1 \wedge Q_2)$
- **Distributivity of Disjunction** for deterministic S:
  $wp(S, Q_1) \vee wp(S, Q_2) \equiv wp(S, Q_1 \vee Q_2)$

Derived Properties:

- **Law of Monotonicity**: if $Q_1 \Rightarrow Q_2$, then
  $wp(S, Q_1) \Rightarrow wp(S, Q_2)$
- **Distributivity of Disjunction** for nondeterministic S:
  $wp(S, Q_1) \vee wp(S, Q_2) \Rightarrow wp(S, Q_1 \vee Q_2)$

Predicate Calculation

- Equivalence is preserved by substituting equals for equals
- Example:

$$
\begin{array}{ll}
 & (A \vee B) \rightarrow C \\
\equiv & \{\ A \rightarrow B \ \equiv\ \neg A \vee B\ \} \\
 & \neg(A \vee B) \vee C \\
\equiv & \{\ \text{de Morgan's law}\ \} \\
 & (\neg A \wedge \neg B) \vee C \\
\equiv & \{\ \text{distributive law}\ \} \\
 & (\neg A \vee C) \wedge (\neg B \vee C) \\
\equiv & \{\ A \rightarrow B \ \equiv\ \neg A \vee B\ \} \\
 & (A \rightarrow C) \wedge (B \rightarrow C)
\end{array}
$$

Predicate Calculation (cont.)

- Entailment distributes over conjunction, disjunction, quantification, and the consequence of an implication.
- Example:

$$
\begin{array}{ll}
 & \forall x(A \rightarrow B) \wedge \forall xA \\
\Rightarrow & \quad \{\ \forall x(A \rightarrow B) \ \Rightarrow \ (\forall xA \rightarrow \forall xB)\ \} \\
 & (\forall xA \rightarrow \forall xB) \wedge \forall xA \\
\equiv & (\neg\forall xA \vee \forall xB) \wedge \forall xA \\
\equiv & (\neg\forall xA \wedge \forall xA) \vee (\forall xB \wedge \forall xA) \\
\equiv & \quad \{\ \neg A \wedge A \ \equiv \ \mathit{false}\ \} \\
 & \mathit{false} \vee (\forall xB \wedge \forall xA) \\
\equiv & \quad \{\ \mathit{false} \vee A \ \equiv \ A\ \} \\
 & \forall xB \wedge \forall xA \\
\Rightarrow & \forall xB
\end{array}
$$

Some Laws for Predicate Calculation

- Equivalence is commutative and associative
  - $A \leftrightarrow B \quad \equiv \quad B \leftrightarrow A$
  - $A \leftrightarrow (B \leftrightarrow C) \quad \equiv \quad (A \leftrightarrow B) \leftrightarrow C$
- $\mathit{false} \vee A \quad \equiv \quad A \vee \mathit{false} \quad \equiv \quad A$
- $\neg A \wedge A \quad \equiv \quad \mathit{false}$
- $A \rightarrow B \quad \equiv \quad \neg A \vee B$
- $A \rightarrow \mathit{false} \quad \equiv \quad \neg A$
- $(A \vee B) \rightarrow C \quad \equiv \quad (A \rightarrow C) \wedge (B \rightarrow C)$
- $A \rightarrow (B \rightarrow C) \quad \equiv \quad (A \wedge B) \rightarrow C$
- $A \rightarrow B \quad \equiv \quad A \leftrightarrow (A \wedge B)$
- $A \wedge B \quad \Rightarrow \quad A$

Some Laws for Predicate Calculation (cont.)

- $\forall x(x = E \rightarrow A) \;\equiv\; A[E/x] \;\equiv\; \exists x(x = E \wedge A)$, if x is not free in E.
- $\forall x(A \wedge B) \;\equiv\; \forall x A \wedge \forall x B$
- $\forall x(A \rightarrow B) \;\Rightarrow\; \forall x A \rightarrow \forall x B$
- $\forall x(A \rightarrow B) \;\equiv\; A \rightarrow \forall x B$, if x is not free in A.
- $\exists x(A \wedge B) \;\equiv\; A \wedge \exists x B$, if x is not free in A.

"Extreme" Programs

- $wp(\textbf{skip}, Q) \triangleq Q$
- $wp(\textbf{choose } x, x \in \mathrm{Dom}(x)) \triangleq true$
- $wp(\textbf{choose } x, Q) \triangleq Q$, if x is not free in Q
- $wp(\textbf{abort}, Q) \triangleq false$

The Assignment Statement

- Syntax: $x := E$
  Note: this becomes a multiple assignment, if we view x as a list of distinct variables and E as a list of expressions.
- Semantics: $wp(x := E, Q) \triangleq Q[E/x]$.

Sequencing

- Syntax: $S_1; S_2$
- Semantics: $wp(S_1; S_2, Q) \triangleq wp(S_1, wp(S_2, Q))$.

Abbreviation of Conjunctions/Disjunctions

- Conjunction:
  - Original Form: $B_1 \land B_2 \land \cdots \land B_n$
  - Abbreviation: $\forall i : 1 \leq i \leq n : B_i$
- Disjunction:
  - Original Form: $B_1 \lor B_2 \lor \cdots \lor B_n$
  - Abbreviation: $\exists i : 1 \leq i \leq n : B_i$
- This applies to conjuctions/disjunctions of first-order formulae, Hoare triples, etc.

The Alternative Statement

IM NTU

- Syntax:

  IF: **if** $B_1 \rightarrow S_1$
  $[]\, B_2 \rightarrow S_2$
  $\ldots$
  $[]\, B_n \rightarrow S_n$
  fi

  Each of the "$B_i \rightarrow S_i$"s is a guarded command, where B_i is the guard (a boolean expression) and S_i the command (body).
- Informal description: One of the guarded commands, whose guard evaluates to true, is nondeterministically selected and the corresponding command executed. If none of the guards evaluates to true, then the execution aborts.

The Alternative Statement (cont.)

- Syntax:

  $$\begin{array}{l} \mathrm{IF}:\ \textbf{if}\ B_1 \rightarrow S_1 \\ \quad\ [\,]\ B_2 \rightarrow S_2 \\ \quad\ \ldots \\ \quad\ [\,]\ B_n \rightarrow S_n \\ \quad\ \textbf{fi} \end{array}$$

- Semantics:

  $$\begin{array}{rcl} wp(\mathrm{IF}, Q) & \triangleq & (\exists i : 1 \leq i \leq n : B_i) \\ & \wedge & (\forall i : 1 \leq i \leq n : B_i \rightarrow wp(S_i, Q)) \end{array}$$

- The case of simple IF:

  $$wp(\textbf{if}\ B \rightarrow S\ \textbf{fi}, Q) \triangleq B \wedge (B \rightarrow wp(S, Q))$$

The Alternative Statement (cont.)

Suppose there exists a predicate P such that

1. $P \Rightarrow (\exists i : 1 \leq i \leq n : B_i)$ and
2. $\forall i : 1 \leq i \leq n : P \wedge B_i \Rightarrow wp(S_i, Q)$.

Then $P \Rightarrow wp(\mathrm{IF}, Q)$.

The less obvious part is $P \Rightarrow (\forall i : 1 \leq i \leq n : B_i \rightarrow wp(S_i, Q))$.

$$
\begin{array}{ll}
 & \forall i : 1 \leq i \leq n : (P \wedge B_i) \rightarrow wp(S_i, Q) \\
\equiv & \forall i : 1 \leq i \leq n : P \rightarrow (B_i \rightarrow wp(S_i, Q)) \\
\equiv & P \rightarrow (\forall i : 1 \leq i \leq n : B_i \rightarrow wp(S_i, Q))
\end{array}
$$

The Alternative Statement (cont.)

- Inference rule in the Hoare logic:

$$\frac{P \Rightarrow (\exists i : 1 \leq i \leq n : B_i) \qquad \forall i : 1 \leq i \leq n : \{P \wedge B_i\}\ S_i\ \{Q\}}{\{P\}\ \mathrm{IF}:\ \textbf{if}\ B_1 \rightarrow S_1 [] \cdots []\ B_n \rightarrow S_n\ \textbf{fi}\ \{Q\}}$$

- This rule follows from the preceding theorem.
- The case of simple IF:

$$\frac{P \Rightarrow B \qquad \{P \wedge B\}\ S\ \{Q\}}{\{P\}\ \textbf{if}\ B \rightarrow S\ \textbf{fi}\ \{Q\}}$$

The Iterative Statement

- Syntax:

  $$\begin{array}{ll} \mathrm{DO}: & \textbf{do}\ B_1 \rightarrow S_1 \\ & \ \ [\,]\ \ B_2 \rightarrow S_2 \\ & \ \ \cdots \\ & \ \ [\,]\ \ B_n \rightarrow S_n \\ & \textbf{od} \end{array}$$

  Each of the "$B_i \rightarrow S_i$"s is a guarded command.
- Informal description: Choose (nondeterministically) a guard B_i that evaluates to true and execute the corresponding command S_i. If none of the guards evaluates to true, then the execution terminates.
- The usual "**while** B **do** S **od**" can be defined as this simple *while*-loop: "**do** $B \rightarrow S$ **od**".

The Iterative Statement (cont.)

- Let BB denote $\exists i : 1 \leq i \leq n : B_i$, i.e., $B_1 \vee B_2 \vee \cdots \vee B_n$.
- The DO statement is equivalent to

  do $\mathrm{BB} \rightarrow$ **if** $B_1 \rightarrow S_1$
  $\quad\quad\quad\quad\quad\quad\; [] \, B_2 \rightarrow S_2$
  $\quad\quad\quad\quad\quad\quad\; \ldots$
  $\quad\quad\quad\quad\quad\quad\; [] \, B_n \rightarrow S_n$
  $\quad\quad\quad\quad\quad\quad\;$ **if**
  od

  or simply **do** $\mathrm{BB} \rightarrow \mathrm{IF}$ **od**.
- This suggests that we could have got by with just the simple *while*-loop.

The Iterative Statement (cont.)

- Again, let BB denote $\exists i : 1 \leq i \leq n : B_i$.
- Let $H_k(Q)$, $k \geq 0$, be defined as follows.

$$\begin{cases} H_0(Q) & \triangleq & \neg\mathrm{BB} \wedge Q \\ H_k(Q) & \triangleq & H_0(Q) \vee wp(\mathrm{IF}, H_{k-1}(Q)) \quad \text{for } k > 0 \end{cases}$$

- The predicate $H_0(Q)$ represents the set of states where execution of DO terminates immediately (0 iteration).
- The predicate $H_k(Q)$, for $k > 0$, represents the set of states where execution of DO terminates after at most k iterations.
- Semantics of DO:

$$wp(\mathrm{DO}, Q) \triangleq (\exists k : 0 \leq k : H_k(Q))$$

A More Useful Theorem for DO

Suppose there exist a predicate P and an integer-valued expression t such that

1. $\forall i : 1 \leq i \leq n : P \wedge B_i \Rightarrow wp(S_i, P)$,
2. $P \Rightarrow (t \geq 0)$, and
3. $\forall i : 1 \leq i \leq n : P \wedge B_i \wedge (t = t_0) \Rightarrow wp(S_i,\ t < t_0)$, where t_0 is a rigid variable.

Then $P \Rightarrow wp(\mathrm{DO}, P \wedge \neg \mathrm{BB})$.

$$
\begin{array}{llll}
P & \equiv & P \wedge (\exists k : 0 \leq k : t \leq k) & (t \text{ is finite}) \\
 & \equiv & \exists k : 0 \leq k : P \wedge t \leq k & (k \text{ is not free in } P) \\
 & \Rightarrow & \exists k : 0 \leq k : H_k(P \wedge \neg \mathrm{BB}) & (P \wedge t \leq k \Rightarrow H_k(P \wedge \neg \mathrm{BB})) \\
 & \equiv & wp(\mathrm{DO}, P \wedge \neg \mathrm{BB}) & (\text{def. of } \mathrm{DO})
\end{array}
$$

A More Useful Theorem for DO (cont.)

- Proof of $P \wedge t \leq k \Rightarrow H_k(P \wedge \neg \mathrm{BB})$ is by induction on k.
- Will do this for the case of simple DO.

A Simplified Theorem for Simple DO

IM NTU

Suppose there exist a predicate P and an integer-valued expression t such that

1. $P \wedge B \Rightarrow wp(S, P)$,
2. $P \Rightarrow (t \geq 0)$, and
3. $P \wedge B \wedge (t = t_0) \Rightarrow wp(S,\ t < t_0)$, where t_0 is a rigid variable.

Then $P \Rightarrow wp(\mathbf{do}\ B \rightarrow S\ \mathbf{od}, P \wedge \neg B)$.

This is to be contrasted by

$$\frac{\{P \wedge B\}\ S\ \{P\} \qquad \{P \wedge B \wedge t = Z\}\ S\ \{t < Z\} \qquad P \Rightarrow (t \geq 0)}{\{P\}\ \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{od}\ \{P \wedge \neg B\}}$$

A Simplified Theorem for Simple DO (cont.)

Proof of $P \wedge t \leq k \Rightarrow H_k(P \wedge \neg B)$ is by induction on k.

Recall, for simple DO,

$$\begin{cases} H_0(Q) & \triangleq & \neg B \wedge Q \\ H_k(Q) & \triangleq & H_0(Q) \vee wp(\textbf{if } B \rightarrow S \textbf{ fi}, H_{k-1}(Q)) \quad \text{for } k > 0 \end{cases}$$

A Simplified Theorem for Simple DO (cont.)

- Base case: $P \wedge t \leq 0 \Rightarrow H_0(P \wedge \neg B)$, which is equivalent to $P \wedge t \leq 0 \Rightarrow P \wedge \neg B$.

  Since $P \Rightarrow (t \geq 0)$, it suffices to show that $P \wedge t = 0 \Rightarrow \neg B$.

$$
\begin{array}{ll}
 & P \wedge t = 0 \wedge B \\
\equiv & (P \wedge B) \wedge (P \wedge B \wedge t = 0) \\
\Rightarrow & wp(S, P) \wedge wp(S, t < 0) \\
\equiv & wp(S, P \wedge t < 0) \\
\equiv & wp(S, \mathit{false}) \\
\equiv & \mathit{false}
\end{array}
$$

A Simplified Theorem for Simple DO (cont.)

- Inductive step ($k > 0$): $P \wedge t \leq k \Rightarrow H_k(P \wedge \neg B)$, i.e.,
  $P \wedge t \leq k \Rightarrow H_0(P \wedge \neg B) \vee wp(\textbf{if } B \rightarrow S \textbf{ fi}, H_{k-1}(P \wedge \neg B))$.

  Split $P \wedge t \leq k$ into three cases:
  - $P \wedge (t \leq k - 1)$
  - $P \wedge B \wedge (t = k)$

$$
\begin{array}{ll}
\Rightarrow & B \wedge (B \rightarrow wp(S, P)) \wedge B \wedge (B \rightarrow wp(S, t < k)) \\
\Rightarrow & wp(\textbf{if } B \rightarrow S \textbf{ fi}, P) \wedge wp(\textbf{if } B \rightarrow S \textbf{ fi}, t < k) \\
\equiv & wp(\textbf{if } B \rightarrow S \textbf{ fi}, P \wedge t < k) \\
\equiv & wp(\textbf{if } B \rightarrow S \textbf{ fi}, P \wedge (t \leq k - 1)) \\
\Rightarrow & \quad \{ \text{ Ind. Hypothesis and Monotonicity of } wp \ \} \\
 & wp(\textbf{if } B \rightarrow S \textbf{ fi}, H_{k-1}(P \wedge \neg B)) \\
\Rightarrow & H_0(P \wedge \neg B) \vee wp(\textbf{if } B \rightarrow S \textbf{ fi}, H_{k-1}(P \wedge \neg B))
\end{array}
$$

  - $P \wedge \neg B \wedge (t = k)$

Refinement

- Syntax:

$$prog_1 \sqsubseteq prog_2$$

  which is read as "$prog_1$ *is refined by* $prog_2$" or "$prog_2$ *refines* $prog_1$" ($prog_2 \sqsupseteq prog_1$).
- Informal description: intuitively, the refinement relation conveys the concept of program $prog_2$ being better than $prog_1$. Program $prog_2$ is better in the sense that it is more accurate, applies in more situations, or runs more efficiently.
- A program may be derived through a series of refinement steps.

Specifications

- Syntax:

$$w : [pre, post]$$

  where *pre* is the precondition, *post* is postcondition, and the "w" part is called the *frame*.
- Informal description: the specification describes an *abstract* program such that if the initial state satisfies the precondition *pre*, then it *changes only variables listed in the frame* and terminates in a final state satisfying the postcondition *post*.
- Examples:
  - $y : [0 \leq x \leq 9, y^2 = x]$
  - $y : [0 \leq x, y^2 = x \wedge y \geq 0]$
  - $x : [true, x = x_0 + 1 \vee x = x_0 - 1]$ (x_0 denotes the initial value of x)

Some Laws for Refinement

- strengthen postcondition: If $post' \Rightarrow post$, then

$$w : [pre, post] \sqsubseteq w : [pre, post']$$

  Example:
  $y : [0 \leq x \leq 9, y^2 = x] \sqsubseteq y : [0 \leq x \leq 9, y^2 = x \wedge y \geq 0]$

- weaken precondition: If $pre \Rightarrow pre'$, then

$$w : [pre, post] \sqsubseteq w : [pre', post]$$

  Example:
  $y : [0 \leq x \leq 9, y^2 = x \wedge y \geq 0] \sqsubseteq y : [0 \leq x, y^2 = x \wedge y \geq 0]$

- Combining the two refinements,

$$y : [0 \leq x \leq 9, y^2 = x] \sqsubseteq y : [0 \leq x, y^2 = x \wedge y \geq 0]$$

Some Laws for Refinement (cont.)

- assignment: If $pre \Rightarrow post[E/x]$, then

$$w, x : [pre, post] \sqsubseteq x := E$$

  Note: w may (but not necessarily) be changed.

- sequential composition: For any predicate mid,

$$w : [pre, post] \sqsubseteq w : [pre, mid]; w : [mid, post]$$

Semantics of Specification

- Syntax: $w : [pre, post]$
- Semantics:

$$wp(w : [pre, post], Q) \triangleq pre \wedge (\forall w(post \rightarrow Q))[v/v_0]$$

where the substitution $[v/v_0]$ replaces all "initial" variables, i.e., v_0, by corresponding final variables.
Note: initial variables v_0 do not occur in Q.
- Example: $wp(x := x \pm 1, Q) \equiv Q[x + 1/x] \wedge Q[x - 1/x]$

Semantics of Specification (cont.)

$$
\begin{array}{ll}
 & wp(x := x \pm 1, Q) \\
\equiv & wp(x : [true, x = x_0 + 1 \vee x = x_0 - 1], Q) \\
\equiv & \quad \{ \text{ def. of specification } \} \\
 & true \wedge \forall x((x = x_0 + 1 \vee x = x_0 - 1) \rightarrow Q)[x/x_0] \\
\equiv & \forall x((x = x_0 + 1 \rightarrow Q) \wedge (x = x_0 - 1 \rightarrow Q))[x/x_0] \\
\equiv & (\forall x(x = x_0 + 1 \rightarrow Q) \wedge \forall x(x = x_0 - 1 \rightarrow Q))[x/x_0] \\
\equiv & \forall x(x = x_0 + 1 \rightarrow Q)[x/x_0] \wedge \forall x(x = x_0 - 1 \rightarrow Q)[x/x_0] \\
\equiv & \quad \{ \forall x(x = E \rightarrow A) \ \equiv \ A[E/x] \} \\
 & (Q[x_0 + 1/x])[x/x_0] \wedge (Q[x_0 - 1/x])[x/x_0] \\
\equiv & \quad \{ \text{ Q does not contain } x_0 \} \\
 & Q[x + 1/x] \wedge Q[x - 1/x]
\end{array}
$$

Semantics of Refinement

- Syntax: $prog_1 \sqsubseteq prog_2$
- Semantics: for all Q,

$$wp(prog_1, Q) \Rightarrow wp(prog_2, Q)$$

- Examples:
  - $x := x \pm 1 \sqsubseteq x := x + 1$

$$\begin{array}{ll} & wp(x := x \pm 1, Q) \\ \equiv & Q[x+1/x] \wedge Q[x-1/x] \\ \Rightarrow & Q[x+1/x] \\ \equiv & wp(x := x+1, Q) \end{array}$$

  - $x := x \pm 1 \sqsubseteq x := x - 1$

References

- E.W. Dijkstra. *A Discipline of Programming*, Prentice-Hall, 1976.
- D. Gries. *The Science of Programming*, Springer-Verlag, 1981.
- C. Morgan. *Programming from Specifications, 2nd Edition*, Prentice-Hall, 1994.