Formal Methods in Software Development
Course 13. Predicate Transformers

Mădălina Eraşcu

Content based on the book Leino, K. Rustan M. Program Proofs. MIT Press, 2023; lecture Formal Methods in Software Development by Wolfgang Schreiner, Johannes Kepler University, Linz, Austria

Thanks to Costel Anghel, 3rd year Bachelor student, Applied Informatics

June 5, 2024

Recalling from Previous Lecture

Sum of first *n* natural numbers. Verify it in Dafny.

Recalling from Previous Lecture

```
method MyMethod(x: int) returns (y: int)
requires x >= 20
ensures y >= 50
(0)
{ (1)
var a := x - 1;
(2)
var b := 31;
(3)
y := a + b;
(4)
}
(5)
```

Analyzing the program in a **forward direction**:

$$
\begin{aligned}
&(0) x \geq 20 \\
&(1) x \geq 20 \\
&(2) x \geq 20 \wedge a = x - 1 \\
&(3) x \geq 20 \wedge a = x - 1 \wedge b = 31 \\
&(4) x \geq 20 \wedge a = x - 1 \wedge b = 31 \wedge y = a + b \\
&(5) y \geq 50
\end{aligned}
$$

Moreover, the postcondition must be shown true, i.e.

$$x \geq 20 \;\wedge\; a = x - 1 \;\wedge\; b = 31 \;\wedge\; y = a + b \quad\Longrightarrow\quad y \geq 50$$

We can also analyze it **backwards**.

Strongest Postcondition

Example

Consider the Hoare triple $\{\!\{x == 0\}\!\} y := x + 3 \{\!\{Q\}\!\}$. There are many Q's which make the Hoare triple valid, for example:

- $\{\!\{x == 0\}\!\} y := x + 3 \{\!\{y < 100\}\!\}$
- $\{\!\{x == 0\}\!\} y := x + 3 \{\!\{x == 0\}\!\}$
- $\{\!\{x == 0\}\!\} y := x + 3 \{\!\{0 <= x \&\& y == 3\}\!\}$
- $\{\!\{x == 0\}\!\} y := x + 3 \{\!\{3 <= y\}\!\}$
- $\{\!\{x == 0\}\!\} y := x + 3 \{\!\{\mathit{true}\}\!\}$

In a **forward derivation**, we want to compute the *strongest (most precise) post-state predicate*, i.e. strongest postcondition.

Remark

If two Hoare triples $\{\!\{P\}\!\} S \{\!\{Q_0\}\!\}$ and $\{\!\{P\}\!\} S \{\!\{Q_1\}\!\}$ are both valid, then also $\{\!\{P\}\!\} S \{\!\{Q_0 \&\& Q_1\}\!\}$ is valid.

Weakest Precondition

Example

Consider the Hoare triple $\{\{P\}\} y := x + 3 \{\{y <= 80\}\}$. There are many P's which make the Hoare triple valid, for example:

- $\{\{x <= 70\}\} y := x + 3 \{\{y <= 80\}\}$
- $\{\{x == 65 \;\&\&\; y < 21\}\} y := x + 3 \{\{y <= 80\}\}$
- $\{\{x <= 77\}\} y := x + 3 \{\{y <= 80\}\}$
- $\{\{x * x + y * y <= 2500\}\} y := x + 3 \{\{y <= 80\}\}$
- $\{\{\mathit{false}\}\} y := x + 3 \{\{y <= 80\}\}$

In a **backward derivation**, we want to compute the *weakest (most general) pre-state predicate*, i.e. weakest precondition.

Remark

If two Hoare triples $\{\{P_0\}\} S \{\{Q\}\}$ and $\{\{P_1\}\} S \{\{Q\}\}$ are both valid, then also $\{\{P_0 || P_1\}\} S \{\{Q\}\}$ is valid.

Remark

Even counterintuitive, computing the weakeast precondition is easier than computing the strongest post-condition.

Computing the weakest precondition. Examples

Remark

For computing the weakest precondition of an assignment statement, we just replace the value of the assigned variable in the postcondition. For example, P in $\{\{P\}\} y := a + b \{\{25 <= y\}\}$ is $25 <= a + b$

Let the statements sequence which swaps the values stored in x and y.

```
var tmp := x;
x := y;
y := tmp;
```

Computing the weakest precondition. Examples

Remark

For computing the weakest precondition of an assignment statement, we just replace the value of the assigned variable in the postcondition. For example, P in $\{\{P\}\} y := a + b \{\{25 <= y\}\}$ is $25 <= a + b$

Let the statements sequence which swaps the values stored in x and y.

```
var tmp := x;
x := y;
y := tmp;
```

- $\{\{x := Y \&\& y == X\}\}$ (postconditie)

Computing the weakest precondition. Examples

Remark

For computing the weakest precondition of an assignment statement, we just replace the value of the assigned variable in the postcondition. For example, P in $\{\{P\}\} y := a + b \{\{25 <= y\}\}$ is $25 <= a + b$

Let the statements sequence which swaps the values stored in x and y.

```
var tmp := x;
x := y;
y := tmp;
```

- $y := tmp$
- $\{\{x := Y \&\& y == X\}\}$ (postconditie)

Computing the weakest precondition. Examples

Remark

For computing the weakest precondition of an assignment statement, we just replace the value of the assigned variable in the postcondition. For example, P in $\{\{P\}\}y := a + b\{\{25 <= y\}\}$ is $25 <= a + b$

Let the statements sequence which swaps the values stored in x and y.

```
var tmp := x;
x := y;
y := tmp;
```

- $\{\{x := Y\&\&tmp == X\}\}$
- $y := tmp$
- $\{\{x := Y\&\&y == X\}\}$ (postconditie)

Computing the weakest precondition. Examples

Remark

For computing the weakest precondition of an assignment statement, we just replace the value of the assigned variable in the postcondition. For example, P in $\{\{P\}\} y := a + b \{\{25 <= y\}\}$ is $25 <= a + b$

Let the statements sequence which swaps the values stored in x and y.

```
var tmp := x;
x := y;
y := tmp;
```

- $x := y$
- $\{\{x := Y \&\& tmp == X\}\}$
- $y := tmp$
- $\{\{x := Y \&\& y == X\}\}$ (postconditie)

Computing the weakest precondition. Examples

Remark

For computing the weakest precondition of an assignment statement, we just replace the value of the assigned variable in the postcondition. For example, P in $\{\{P\}\} y := a + b \{\{25 <= y\}\}$ is $25 <= a + b$

Let the statements sequence which swaps the values stored in x and y.

```
var tmp := x;
x := y;
y := tmp;
```

- $\{\{y := Y \&\& tmp == X\}\}$
- $x := y$
- $\{\{x := Y \&\& tmp == X\}\}$
- $y := tmp$
- $\{\{x := Y \&\& y == X\}\}$ (postconditie)

Computing the weakest precondition. Examples

Remark

For computing the weakest precondition of an assignment statement, we just replace the value of the assigned variable in the postcondition. For example, P in $\{\{P\}\} y := a + b \{\{25 <= y\}\}$ is $25 <= a + b$

Let the statements sequence which swaps the values stored in x and y.

```
var tmp := x;
x := y;
y := tmp;
```

- var tmp :=x
- $\{\{y := Y \&\& tmp == X\}\}$
- $x := y$
- $\{\{x := Y \&\& tmp == X\}\}$
- $y := tmp$
- $\{\{x := Y \&\& y == X\}\}$ (postconditie)

Computing the weakest precondition. Examples

Remark

For computing the weakest precondition of an assignment statement, we just replace the value of the assigned variable in the postcondition. For example, P in $\{\{P\}\} y := a + b \{\{25 <= y\}\}$ is $25 <= a + b$

Let the statements sequence which swaps the values stored in x and y.

```
var tmp := x;
x := y;
y := tmp;
```

- $\{\{y := Y \&\& x == X\}\}$
- var tmp :=x
- $\{\{y := Y \&\& tmp == X\}\}$
- $x := y$
- $\{\{x := Y \&\& tmp == X\}\}$
- $y := tmp$
- $\{\{x := Y \&\& y == X\}\}$ (postconditie)

Computing the weakest precondition. Examples

Remark

For computing the weakest precondition of an assignment statement, we just replace the value of the assigned variable in the postcondition. For example, P in $\{\{P\}\} y := a + b \{\{25 <= y\}\}$ is $25 <= a + b$

Let the statements sequence which swaps the values stored in x and y.

```
var tmp := x;
x := y;
y := tmp;
```

- $\{\{x := X \&\& y == Y\}\}$ (preconditie)
- $\{\{y := Y \&\& x == X\}\}$
- var tmp :=x
- $\{\{y := Y \&\& tmp == X\}\}$
- $x := y$
- $\{\{x := Y \&\& tmp == X\}\}$
- $y := tmp$
- $\{\{x := Y \&\& y == X\}\}$ (postconditie)

Computing the weakest precondition. Examples

Remark

For computing the weakest precondition of an assignment statement, we just replace the value of the assigned variable in the postcondition. For example, P in $\{\{P\}\}y := a + b\{\{25 <= y\}\}$ is $25 <= a + b$

Let the statements sequence which swaps the values stored in x and y.

```
var tmp := x;
x := y;
y := tmp;
```

- $\{\{x := X\&\&y == Y\}\}$ (preconditie)
- $\{\{y := Y\&\&x == X\}\}$
- var tmp :=x
- $\{\{y := Y\&\&tmp == X\}\}$
- $x := y$
- $\{\{x := Y\&\&tmp == X\}\}$
- $y := tmp$
- $\{\{x := Y\&\&y == X\}\}$ (postconditie)

Write in Dafny a program which swaps the values of 2 integer numbers. Proves it is correct.

Computing the weakest precondition. Examples

Remark

Dafny allows simultaneous assignments, For example,

```
x, y := 10, 3
```

sets x to 10 and y to 3 at the same time.

Quiz! Write the Dafny code for swapping the values of two integers using simultaneous assignments. Write appropriate postcondition and by using backward reasoning (weakest precondition) show the correctness of the code.

Recalling the Hoare logic from the previous lecture

- `skip` command

$$\{\!\{P\}\!\}\ \texttt{skip}\ \{\!\{P\}\!\}$$

- `abort` command

$$\{\!\{\mathit{true}\}\!\}\ \texttt{abort}\ \{\!\{\mathit{false}\}\!\}$$

- Scalar assignment

$$\{\!\{Q[e/x]\}\!\}\ x := e\ \{\!\{Q\}\!\}$$

- Array assignment

$$\{\!\{Q[a[i \rightarrowtail e]/a]\}\!\}\ a[i] := e\ \{\!\{Q\}\!\}$$

- Command Sequences

$$\frac{\{\!\{P\}\!\} c_1 \{\!\{R\}\!\} \qquad \{\!\{R\}\!\} c_2 \{\!\{Q\}\!\}}{\{\!\{P\}\!\} c_1 ; c_2 \{\!\{Q\}\!\}}$$

- Conditionals

$$\frac{\{\!\{P \wedge b\}\!\}\ c_1\ \{\!\{Q\}\!\} \qquad \{\!\{P \wedge \neg b\}\!\}\ c_2\ \{\!\{Q\}\!\}}{\{\!\{P\}\!\}\ \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2\ \{\!\{Q\}\!\}}$$

$$\frac{\{\!\{P \wedge b\}\!\}\ c\ \{\!\{Q\}\!\} \qquad \{\!\{P \wedge \neg b\}\!\} \Longrightarrow \{\!\{Q\}\!\}}{\{\!\{P\}\!\}\ \textbf{if } b \textbf{ then } c\ \{\!\{Q\}\!\}}$$

- Loops (partial correctness)

$$\frac{P \Longrightarrow I \qquad \{\!\{I \wedge b\}\!\}\ c\ \{\!\{I\}\!\} \qquad (I \wedge \neg b) \Longrightarrow Q}{\{\!\{P\}\!\}\ \textbf{while } b \textbf{ do } c\ \{\!\{Q\}\!\}}$$

- Loops (total correctness)

$$\frac{P \Longrightarrow I \quad I \Longrightarrow t \geq 0 \quad \{\!\{I \wedge b \wedge t = N\}\!\}\ c\ \{\!\{I \wedge t < N\}\!\}\ (I \wedge \neg b) \Longrightarrow Q}{\{\!\{P\}\!\}\ \textbf{while } b \textbf{ do } c\ \ \{\!\{Q\}\!\}}$$

Weakest Precondition

A calculus for *backward reasoning* (E.W. Dijkstra)

- Predicate transformer `WP`
  - `WP` takes as arguments a command c and a postcondition Q and returns a precondition.
  - Read $\mathtt{WP}(c, Q)$ as *the weakest precondition of c w.r.t. Q*.

Weakest Precondition

A calculus for *backward reasoning* (E.W. Dijkstra)

- Predicate transformer $\mathtt{WP}$
  - $\mathtt{WP}$ takes as arguments a command c and a postcondition Q and returns a precondition.
  - Read $\mathtt{WP}(c, Q)$ as *the weakest precondition of c w.r.t. Q*.
- $\mathtt{WP}(c, Q)$ is a precondition for c that ensures Q as a postcondition.
  - Must satisfy $\{\!\{\mathtt{WP}(c, Q)\}\!\}\ c\ \{\!\{Q\}\!\}$.

Weakest Precondition

A calculus for *backward reasoning* (E.W. Dijkstra)

- Predicate transformer $\mathtt{WP}$
  - $\mathtt{WP}$ takes as arguments a command c and a postcondition Q and returns a precondition.
  - Read $\mathtt{WP}(c, Q)$ as *the weakest precondition of c w.r.t. Q*.
- $\mathtt{WP}(c, Q)$ is a precondition for c that ensures Q as a postcondition.
  - Must satisfy $\{\!\{\mathtt{WP}(c, Q)\}\!\}\ c\ \{\!\{Q\}\!\}$.
- $\mathtt{WP}(c, Q)$ is the weakest such precondition.
  - Take any P such that $\{\!\{P\}\!\}\ c\ \{\!\{Q\}\!\}$.
  - Then $P \Rightarrow \mathtt{WP}(c, Q)$.

Weakest Precondition

A calculus for *backward reasoning* (E.W. Dijkstra)

- Predicate transformer $\texttt{WP}$
  - $\texttt{WP}$ takes as arguments a command c and a postcondition Q and returns a precondition.
  - Read $\texttt{WP}(c, Q)$ as *the weakest precondition of c w.r.t. Q*.
- $\texttt{WP}(c, Q)$ is a precondition for c that ensures Q as a postcondition.
  - Must satisfy $\{\!\{\texttt{WP}(c, Q)\}\!\}\ c\ \{\!\{Q\}\!\}$.
- $\texttt{WP}(c, Q)$ is the weakest such precondition.
  - Take any P such that $\{\!\{P\}\!\}\ c\ \{\!\{Q\}\!\}$.
  - Then $P \Rightarrow \texttt{WP}(c, Q)$.
- Consequence: $\{\!\{P\}\!\}\ c\ \{\!\{Q\}\!\}$ *iff* $(P \Rightarrow \texttt{WP}(c, Q))$.
  - We want to prove $\{\!\{P\}\!\}\ c\ \{\!\{Q\}\!\}$.
  - We may prove $P \Rightarrow \texttt{WP}(c, Q)$ instead.

Weakest Precondition

A calculus for *backward reasoning* (E.W. Dijkstra)

- Predicate transformer $\mathtt{WP}$
  - $\mathtt{WP}$ takes as arguments a command c and a postcondition Q and returns a precondition.
  - Read $\mathtt{WP}(c, Q)$ as *the weakest precondition of c w.r.t. Q*.
- $\mathtt{WP}(c, Q)$ is a precondition for c that ensures Q as a postcondition.
  - Must satisfy $\{\!\{\mathtt{WP}(c, Q)\}\!\}\ c\ \{\!\{Q\}\!\}$.
- $\mathtt{WP}(c, Q)$ is the weakest such precondition.
  - Take any P such that $\{\!\{P\}\!\}\ c\ \{\!\{Q\}\!\}$.
  - Then $P \Rightarrow \mathtt{WP}(c, Q)$.
- Consequence: $\{\!\{P\}\!\}\ c\ \{\!\{Q\}\!\}$ *iff* $(P \Rightarrow \mathtt{WP}(c, Q))$.
  - We want to prove $\{\!\{P\}\!\}\ c\ \{\!\{Q\}\!\}$.
  - We may prove $P \Rightarrow \mathtt{WP}(c, Q)$ instead.

Verification is reduced to the calculation of weakest preconditions.

Weakest Precondition Calculus

Here we have the weakest precondition for each program construct:

- $\mathtt{WP}(\textbf{skip}, Q) = Q$
- $\mathtt{WP}(\textbf{abort}, Q) = \mathit{true}$
- $\mathtt{WP}(x := e, Q) = Q[e/x]$
- $\mathtt{WP}(c_1; c_2, Q) = \mathtt{WP}(c_1, \mathtt{WP}(c_2, Q))$
- $\mathtt{WP}(\textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2,\ Q) = (b \Longrightarrow \mathtt{WP}(c_1, Q)) \wedge (\neg b \Longrightarrow \mathtt{WP}(c_2, Q))$
- $\mathtt{WP}(\textbf{if } b \textbf{ then } c, Q) \iff (b \Longrightarrow \mathtt{WP}(c, Q)) \wedge (\neg b \Longrightarrow Q)$
- $\mathtt{WP}(\textbf{while } b \textbf{ then } c, Q) = \ldots$

Remark

Computing $\mathtt{WP}(\textbf{while } b \textbf{ then } c, Q)$ requires advanced formal methods and computational logic knowledge which will be introduced into a lecture at master studies.