# Introduction to Program Verification

Laura Kovács

# Program Verification: Programs and Specifications

## Example – Maximum of Two Natural Numbers

Given two natural numbers $x$ and $y$.
Compute the maximum value of $x$ and $y$.

The maximum of $x$ and $y$ is $x$ iff $x \geq y$.
Otherwise, the maximum of $x$ and $y$ is $y$.

Computing the maximum (*max*) of $x$ and $y$:

```
if (x ≥ y)
   then max := x
   else max := y
```

## Example – Maximum of Two Natural Numbers

Given two natural numbers $x$ and $y$.
Compute the maximum value of $x$ and $y$.

The maximum of $x$ and $y$ is $x$ iff $x \geq y$.
Otherwise, the maximum of $x$ and $y$ is $y$.

Computing the maximum (*max*) of $x$ and $y$:

```
if (x ≥ y)
   then max := x
   else max := y
```

## Example – Maximum of Two Natural Numbers

Given two natural numbers $x$ and $y$.
Compute the maximum value of $x$ and $y$.

The maximum of $x$ and $y$ is $x$ iff $x \geq y$.
Otherwise, the maximum of $x$ and $y$ is $y$.

Computing the maximum ($max$) of $x$ and $y$:

$$\begin{array}{l} \underline{\text{if}}\ (x \geq y) \\ \quad \underline{\text{then}}\ max := x \\ \quad \underline{\text{else}}\ max := y \end{array}$$

## Example – Maximum of Two Natural Numbers

Given two natural numbers $x$ and $y$.

**REQUIREMENT ON PROGRAM'S INPUT**

The maximum of $x$ and $y$ is $x$ iff $x \geq y$.
Otherwise, the maximum of $x$ and $y$ is $y$.

**REQUIREMENT ON PROGRAM'S OUTPUT**

Computing the maximum (*max*) of $x$ and $y$:

```
if (x ≥ y)
  then max := x
  else max := y
```

**PROGRAM**

## Example – Maximum of Two Natural Numbers

Given two natural numbers $x$ and $y$.

**Requirement on Program's Input**

**Precondition**

The maximum of $x$ and $y$ is $x$ iff $x \geq y$.
Otherwise, the maximum of $x$ and $y$ is $y$.

**Requirement on Program's Output**

**Postcondition**

Computing the maximum (*max*) of $x$ and $y$:

<u>if</u> $(x \geq y)$
  <u>then</u> $max := x$
  <u>else</u> $max := y$

**Program**

## Example – Maximum of Two Natural Numbers

Given two natural numbers $x$ and $y$.

$(x \geq 0 \land y \geq 0)$

**REQUIREMENT ON PROGRAM'S INPUT**

**PRECONDITION**

The maximum of $x$ and $y$ is $x$ iff $x \geq y$.
Otherwise, the maximum of $x$ and $y$ is $y$.

$(max \geq x) \land (max \geq y) \land (max = x \lor max = y)$

**REQUIREMENT ON PROGRAM'S OUTPUT**

**POSTCONDITION**

Computing the maximum ($max$) of $x$ and $y$:

<u>if</u> $(x \geq y)$
  <u>then</u> $max := x$
  <u>else</u> $max := y$

**PROGRAM**

## Example – Maximum of Two Natural Numbers

Given two natural numbers $x$ and $y$.
$P : (x \geq 0 \wedge y \geq 0)$

**REQUIREMENT ON PROGRAM'S INPUT**

**PRECONDITION $P$**

The maximum of $x$ and $y$ is $x$ iff $x \geq y$.
Otherwise, the maximum of $x$ and $y$ is $y$.
$Q : (max \geq x) \wedge (max \geq y) \wedge (max = x \vee max = y)$

**REQUIREMENT ON PROGRAM'S OUTPUT**

**POSTCONDITION $Q$**

Computing the maximum ($max$) of $x$ and $y$:

```
if (x ≥ y)
   then max := x
   else max := y
```

**PROGRAM**

# Program Verification: Programs and Specifications

## Example – Maximum of Two Natural Numbers

Given two natural numbers $x$ and $y$.
$P : (x \geq 0 \land y \geq 0)$

**REQUIREMENT ON PROGRAM'S INPUT**

**PRECONDITION $P$**

The maximum of $x$ and $y$ is $x$ iff $x \geq y$.
Otherwise, the maximum of $x$ and $y$ is $y$.
$Q : (max \geq x) \land (max \geq y) \land (max = x \lor max = y)$

**REQUIREMENT ON PROGRAM'S OUTPUT**

**POSTCONDITION $Q$**

Computing the maximum ($max$) of $x$ and $y$:

```
if (x ≥ y)
   then max := x
   else max := y
```

**PROGRAM**

# Program Verification: Programs and Specifications

## Program Verification:

program satisfies its requirements (specification $P$, $Q$) (Vorbedingung $P$, Endbedingung $Q$)

## Example.

Given two natural numbers $x$ and $y$.
Compute the maximum value($max$) of $x$ and $y$.

Precondition $P$: $(x \geq 0) \wedge (y \geq 0)$ INITIAL STATE

Postcondition $Q$: $(max \geq x) \wedge (max \geq y) \wedge (max = x \vee max = y)$ FINAL STATE

Program (code) $S$:

```
if (x ≥ y)
   then max := x
   else max := y
```

HOW?

**Hoare triple (correctness formula):** $\{P\}\ S\ \{Q\}$

T. Hoare (1969)

# Program Verification: Programs and Specifications

## Program Verification:

$\underbrace{\text{program satisfies its requirements (specification } P, Q)}_{\text{PROGRAM CORRECTNESS}}$

## Example.

Given two natural numbers $x$ and $y$.
Compute the maximum value($max$) of $x$ and $y$.

Precondition $P$: $(x \geq 0) \wedge (y \geq 0)$ INITIAL STATE

Postcondition $Q$: $(max \geq x) \wedge (max \geq y) \wedge (max = x \vee max = y)$ FINAL STATE

Program (code) $S$: <u>if</u> $(x \geq y)$
<u>then</u> $max := x$
<u>else</u> $max := y$

HOW?

**Hoare triple (correctness formula)**: $\{P\}\ S\ \{Q\}$

T. Hoare (1969)

# Program Verification: Programs and Specifications

## Program Verification:

program satisfies its requirements (specification $P$, $Q$)

**PROGRAM CORRECTNESS**

## Example.

Given two natural numbers $x$ and $y$.
Compute the maximum value($max$) of $x$ and $y$.

Precondition $P$: $(x \geq 0) \wedge (y \geq 0)$ — INITIAL STATE

Postcondition $Q$: $(max \geq x) \wedge (max \geq y) \wedge (max = x \vee max = y)$ — FINAL STATE

Program (code) $S$:

```
if (x ≥ y)
   then max := x
   else max := y
```

HOW?

**Hoare triple (correctness formula)**: $\{P\}\ S\ \{Q\}$

T. Hoare (1969)

# Program Verification: Programs and Specifications

## Program Verification:

program satisfies its requirements (specification $P$, $Q$)

**PROGRAM CORRECTNESS**

Program . . . HOW to compute
using program statements $S$

Specifications . . . WHAT to compute
using predicate logic formulas $P$, $Q$ (assertions, Zusicherungen)

Program state . . . every program variable has a value

**Hoare triple (correctness formula)**: $\{P\}\ S\ \{Q\}$

T. Hoare (1969)

# Program Verification: Programs and Specifications

## Program Verification:

program satisfies its requirements (specification $P$, $Q$)

**PROGRAM CORRECTNESS**

| | | |
|---|---|---|
| Program | ... | HOW to compute<br>using program statements $S$ |
| Specifications | ... | WHAT to compute<br>using predicate logic formulas $P$, $Q$ (assertions, Zusicherungen) |
| Program state | ... | every program variable has a value |

**Hoare triple (correctness formula)**: $\{P\}\ S\ \{Q\}$

T. Hoare (1969)

# Programs

## Program statements and their meaning (semantics):

(Zuweisung, Sequenz, Konditional, Schleife)

- **Assignments:** $var := A$, where *var* is a program variable (scalar $x$ or array $a[x]$), and $A$ is an arithmetic expression;

  variable *var* receives (is updated by) the value $A$

  $A \stackrel{def}{=} n \mid x \mid a[n] \mid a[x] \mid A_1 + A_2 \mid A_1 - A_2 \mid A_1 * A_2$, where $n \in \mathbb{N}$; $x$ is a *scalar* variable with values from $\mathbb{N}$;
  $a$ is an *array* variable; $A_1$ , $A_2$ are arithmetic expressions

- **Sequencing:** $s_1 ; \; s_2$, where $s_1$ and $s_2$ are program statements;

  execution of statement $s_1$ is followed by execution of statement $s_2$

- **Conditionals:** <u>if</u> $(B)$ <u>then</u> $s_1$ <u>else</u> $s_2$, where $B$ is a boolean expression;

  if $B$ holds then $s_1$ is executed, otherwise $s_2$ is executed

- **Loops:** <u>while</u> $(B)$ <u>do</u> $s$ <u>end while</u>, where $s$ is a program statement.

  until $B$ holds, statement $s$ is executed

## Program $S$ is a finite sequence of statements:

$$S = s_1 ; s_2 ; \ldots ; s_{n-1} ; s_n$$

**NOTE:** LOOPS MAY NOT TERMINATE! (infinite loop)

# Programs

## Program statements and their meaning (semantics):

(Zuweisung, Sequenz, Konditional, Schleife)

- **Assignments:** $var := A$, where *var* is a program variable (scalar $x$ or array $a[x]$), and $A$ is an arithmetic expression;

  variable *var* receives (is updated by) the value $A$

  $A \stackrel{def}{=} n \mid x \mid a[n] \mid a[x] \mid A_1 + A_2 \mid A_1 - A_2 \mid A_1 * A_2$, where $n \in \mathbb{N}$; $x$ is a *scalar* variable with values from $\mathbb{N}$;

  $a$ is an *array* variable; $A_1$ , $A_2$ are arithmetic expressions

- **Sequencing:** $s_1 ;\ s_2$, where $s_1$ and $s_2$ are program statements;

  execution of statement $s_1$ is followed by execution of statement $s_2$

- **Conditionals:** if $(B)$ then $s_1$ else $s_2$, where $B$ is a boolean expression;

  if $B$ holds then $s_1$ is executed, otherwise $s_2$ is executed

  $B \stackrel{def}{=} True \mid False \mid \neg B_1 \mid B_1 \wedge B_2 \mid B_1 \vee B_2 \mid A_1 \leq A_2$, where $B_1$, $B_2$ are boolean expressions;

  $A_1$, $A_2$ are arithmetic expressions

- **Loops:** while $(B)$ do $s$ end while, where $s$ is a program statement.

  until $B$ holds, statement $s$ is executed

## Program $S$ is a finite sequence of statements:

$$S = s_1 ; s_2 ; \ldots ; s_{n-1} ; s_n$$

**NOTE:** LOOPS MAY NOT TERMINATE! (infinite loop)

# Programs

## Program statements and their meaning (semantics):

(Zuweisung, Sequenz, Konditional, Schleife)

- **Assignments:** $var := A$, where *var* is a program variable (scalar $x$ or array $a[x]$), and $A$ is an arithmetic expression;

  variable *var* receives (is updated by) the value $A$

  $A \stackrel{def}{=} n \mid x \mid a[n] \mid a[x] \mid A_1 + A_2 \mid A_1 - A_2 \mid A_1 * A_2$, where $n \in \mathbb{N}$; $x$ is a *scalar* variable with values from $\mathbb{N}$;
  $a$ is an *array* variable; $A_1$ , $A_2$ are arithmetic expressions

- **Sequencing:** $s_1 ;\ s_2$, where $s_1$ and $s_2$ are program statements;

  execution of statement $s_1$ is followed by execution of statement $s_2$

- **Conditionals:** if $(B)$ then $s_1$ else $s_2$, where $B$ is a boolean expression;

  if $B$ holds then $s_1$ is executed, otherwise $s_2$ is executed

  $B \stackrel{def}{=} True \mid False \mid \neg B_1 \mid B_1 \wedge B_2 \mid B_1 \vee B_2 \mid A_1 \leq A_2$, where $B_1$ , $B_2$ are boolean expressions;
  $A_1$ , $A_2$ are arithmetic expressions

- **Loops:** while $(B)$ do $s$ end while, where $s$ is a program statement.

  until $B$ holds, statement $s$ is executed

## Program $S$ is a finite sequence of statements:

$$S = s_1 ; s_2 ; \ldots ; s_{n-1} ; s_n$$

NOTE: LOOPS MAY NOT TERMINATE! (infinite loop)

# Programs

## Program statements and their meaning (semantics):

(Zuweisung, Sequenz, Konditional, Schleife)

- **Assignments:** $var := A$, where *var* is a program variable (scalar $x$ or array $a[x]$), and $A$ is an arithmetic expression;

  variable *var* receives (is updated by) the value $A$

  $A \overset{def}{=} n \mid x \mid a[n] \mid a[x] \mid A_1 + A_2 \mid A_1 - A_2 \mid A_1 * A_2$, where $n \in \mathbb{N}$; $x$ is a *scalar* variable with values from $\mathbb{N}$;

  $a$ is an *array* variable; $A_1$, $A_2$ are arithmetic expressions

- **Sequencing:** $s_1;\ s_2$, where $s_1$ and $s_2$ are program statements;

  execution of statement $s_1$ is followed by execution of statement $s_2$

- **Conditionals:** if $(B)$ then $s_1$ else $s_2$, where $B$ is a boolean expression;

  if $B$ holds then $s_1$ is executed, otherwise $s_2$ is executed

  $B \overset{def}{=} True \mid False \mid \neg B_1 \mid B_1 \wedge B_2 \mid B_1 \vee B_2 \mid A_1 \leq A_2$, where $B_1$, $B_2$ are boolean expressions; $A_1$, $A_2$ are arithmetic expressions

- **Loops:** while $(B)$ do $s$ end while, where $s$ is a program statement.

  until $B$ holds, statement $s$ is executed

## Program $S$ is a finite sequence of statements:

$$S = s_1; s_2; \ldots; s_{n-1}; s_n$$

**NOTE:** Loops may not terminate! (infinite loop)

# Programs

## Program statements and their meaning (semantics):

(Zuweisung, Sequenz, Konditional, Schleife)

- **Assignments:** $var := A$, where *var* is a program variable (scalar $x$ or array $a[x]$), and $A$ is an arithmetic expression;

  variable *var* receives (is updated by) the value $A$

  $A \stackrel{def}{=} n \mid x \mid a[n] \mid a[x] \mid A_1 + A_2 \mid A_1 - A_2 \mid A_1 * A_2$, where $n \in \mathbb{N}$; $x$ is a *scalar* variable with values from $\mathbb{N}$;

  $a$ is an *array* variable; $A_1$ , $A_2$ are arithmetic expressions

- **Sequencing:** $s_1;\ s_2$, where $s_1$ and $s_2$ are program statements;

  execution of statement $s_1$ is followed by execution of statement $s_2$

- **Conditionals:** if $(B)$ then $s_1$ else $s_2$, where $B$ is a boolean expression;

  if $B$ holds then $s_1$ is executed, otherwise $s_2$ is executed

  $B \stackrel{def}{=} True \mid False \mid \neg B_1 \mid B_1 \wedge B_2 \mid B_1 \vee B_2 \mid A_1 \leq A_2$, where $B_1$ , $B_2$ are boolean expressions; $A_1$ , $A_2$ are arithmetic expressions

- **Loops:** while $(B)$ do $s$ end while, where $s$ is a program statement.

  until $B$ holds, statement $s$ is executed

## Program $S$ is a finite sequence of statements:

$$S = s_1; s_2; \ldots; s_{n-1}; s_n$$

**NOTE:** Loops may not terminate! (infinite loop)

# Programs

## Program statements and their meaning (semantics):

(Zuweisung, Sequenz, Konditional, Schleife)

- **Assignments:** $var := A$, where *var* is a program variable (scalar $x$ or array $a[x]$), and $A$ is an arithmetic expression;

  variable *var* receives (is updated by) the value $A$

  $A \stackrel{def}{=} n \mid x \mid a[n] \mid a[x] \mid A_1 + A_2 \mid A_1 - A_2 \mid A_1 * A_2$, where $n \in \mathbb{N}$; $x$ is a *scalar* variable with values from $\mathbb{N}$;
  $a$ is an *array* variable; $A_1$ , $A_2$ are arithmetic expressions

- **Sequencing:** $s_1 ; \; s_2$, where $s_1$ and $s_2$ are program statements;

  execution of statement $s_1$ is followed by execution of statement $s_2$

- **Conditionals:** <u>if</u> $(B)$ <u>then</u> $s_1$ <u>else</u> $s_2$, where $B$ is a boolean expression;

  if $B$ holds then $s_1$ is executed, otherwise $s_2$ is executed

  $B \stackrel{def}{=} True \mid False \mid \neg B_1 \mid B_1 \wedge B_2 \mid B_1 \vee B_2 \mid A_1 \leq A_2$, where $B_1$ , $B_2$ are boolean expressions;
  $A_1$ , $A_2$ are arithmetic expressions

- **Loops:** <u>while</u> $(B)$ <u>do</u> $s$ <u>end while</u>, where $s$ is a program statement.

  until $B$ holds, statement $s$ is executed

## Program $S$ is a finite sequence of statements:

$$S = s_1; s_2; \ldots ; s_{n-1}; s_n$$

NOTE: LOOPS MAY NOT TERMINATE! (infinite loop)

# Programs

## Program statements and their meaning (semantics):

(Zuweisung, Sequenz, Konditional, Schleife)

- **Assignments:** $var := A$, where *var* is a program variable (scalar $x$ or array $a[x]$), and $A$ is an arithmetic expression;

  variable *var* receives (is updated by) the value $A$

  $A \stackrel{def}{=} n \mid x \mid a[n] \mid a[x] \mid A_1 + A_2 \mid A_1 - A_2 \mid A_1 * A_2$, where $n \in \mathbb{N}$; $x$ is a *scalar* variable with values from $\mathbb{N}$;
  $a$ is an *array* variable; $A_1$ , $A_2$ are arithmetic expressions

- **Sequencing:** $s_1;\ s_2$, where $s_1$ and $s_2$ are program statements;

  execution of statement $s_1$ is followed by execution of statement $s_2$

- **Conditionals:** if $(B)$ then $s_1$ else $s_2$, where $B$ is a boolean expression;

  if $B$ holds then $s_1$ is executed, otherwise $s_2$ is executed

  $B \stackrel{def}{=} True \mid False \mid \neg B_1 \mid B_1 \wedge B_2 \mid B_1 \vee B_2 \mid A_1 \leq A_2$, where $B_1$ , $B_2$ are boolean expressions;
  $A_1$ , $A_2$ are arithmetic expressions

- **Loops:** while $(B)$ do $s$ end while, where $s$ is a program statement.

  until $B$ holds, statement $s$ is executed

## Program $S$ is a finite sequence of statements:

$$S = s_1; s_2; \ldots ; s_{n-1}; s_n$$

NOTE: LOOPS MAY NOT TERMINATE! (infinite loop)

# Programs

## Program statements and their meaning (semantics):

(Zuweisung, Sequenz, Konditional, Schleife)

- **Assignments:** $var := A$, where *var* is a program variable (scalar $x$ or array $a[x]$), and $A$ is an arithmetic expression;

  variable *var* receives (is updated by) the value $A$

  $A \stackrel{def}{=} n \mid x \mid a[n] \mid a[x] \mid A_1 + A_2 \mid A_1 - A_2 \mid A_1 * A_2$, where $n \in \mathbb{N}$; $x$ is a *scalar* variable with values from $\mathbb{N}$;

  $a$ is an *array* variable; $A_1$, $A_2$ are arithmetic expressions

- **Sequencing:** $s_1;\ s_2$, where $s_1$ and $s_2$ are program statements;

  execution of statement $s_1$ is followed by execution of statement $s_2$

- **Conditionals:** if $(B)$ then $s_1$ else $s_2$, where $B$ is a boolean expression;

  if $B$ holds then $s_1$ is executed, otherwise $s_2$ is executed

  $B \stackrel{def}{=} True \mid False \mid \neg B_1 \mid B_1 \wedge B_2 \mid B_1 \vee B_2 \mid A_1 \leq A_2$, where $B_1$, $B_2$ are boolean expressions; $A_1$, $A_2$ are arithmetic expressions

- **Loops:** while $(B)$ do $s$ end while, where $s$ is a program statement.

  until $B$ holds, statement $s$ is executed

## Program $S$ is a finite sequence of statements:

$$S = s_1; s_2; \ldots ; s_{n-1}; s_n$$

**NOTE:** LOOPS MAY NOT TERMINATE! (infinite loop)

# Example: Integer Division

## Example.

Given two natural numbers $x$ and $y$, with $y$ being non zero.

Compute:

the quotient (*quo*) and the remainder (*rem*) of the integer division of $x$ by $y$.

Precondition $P$: $(x \geq 0) \wedge (y > 0)$

Postcondition $Q$: $(quo * y + rem = x) \wedge (0 \leq rem < y)$

Program (code) $S$:

```
quo := 0; rem := x;
while y ≤ rem do
   rem := rem − y;  quo := quo + 1
end while
```

**Hoare triple (correctness formula):** $\{P\}\ S\ \{Q\}$

# Example: Integer Division

## Example.

Given two natural numbers $x$ and $y$, with $y$ being non zero.

Compute:

the quotient (*quo*) and the remainder (*rem*) of the integer division of $x$ by $y$.

Precondition $P$: $(x \geq 0) \wedge (y > 0)$

Postcondition $Q$: $(quo * y + rem = x) \wedge (0 \leq rem < y)$

Program (code) $S$:

$quo := 0; rem := x;$
<u>while</u> $y \leq rem$ <u>do</u>
  $rem := rem - y;\ quo := quo + 1$
<u>end while</u>

Hoare triple (correctness formula): $\{P\}\ S\ \{Q\}$

# Example: Integer Division

## Example.

Given two natural numbers $x$ and $y$, with $y$ being non zero.

Compute:

the quotient (*quo*) and the remainder (*rem*) of the integer division of $x$ by $y$.

Precondition $P$: $(x \geq 0) \wedge (y > 0)$

Postcondition $Q$: $(quo * y + rem = x) \wedge (0 \leq rem < y)$

Program (code) $S$:

$quo := 0; rem := x;$
<u>while</u> $y \leq rem$ <u>do</u>
  $rem := rem - y;\ \ quo := quo + 1$
<u>end while</u>

**Hoare triple (correctness formula)**: $\{P\}\ S\ \{Q\}$

# Program Correctness

Partial correctness (partiell/teilweise korrekt) of $\{P\}\ S\ \{Q\}$:

Every execution of $S$ that:

- starts in a state satisfying $P$ and
- is terminating,

ends in a state satisfying $Q$.

# Program Correctness

**Partial correctness** (partiell/teilweise korrekt) of $\{P\}\ S\ \{Q\}$:

Every execution of $S$ that:

- starts in a state satisfying $P$ and
- is terminating,

ends in a state satisfying $Q$.

**Total correctness** (total/vollständig korrekt) of $\{P\}\ S\ \{Q\}$:

Every execution of $S$ that:

- starts in a state satisfying $P$,

terminates in a state satisfying $Q$.

Total correctness = Partial correctness + Termination

# Program Correctness

**Partial correctness** (partiell/teilweise korrekt) of $\{P\}\ S\ \{Q\}$:

Every execution of $S$ that:

- starts in a state satisfying $P$ and
- is terminating,

ends in a state satisfying $Q$.

**Total correctness** (total/vollständig korrekt) of $\{P\}\ S\ \{Q\}$:

Every execution of $S$ that:

- starts in a state satisfying $P$,

terminates in a state satisfying $Q$.

Total correctness = Partial correctness + Termination

# Program Correctness

Partial correctness (partiell/teilweise korrekt) of $\{P\}\ S\ \{Q\}$:

Every execution of $S$ that:

- starts in a state satisfying $P$ and
- is terminating,

ends in a state satisfying $Q$.

Total correctness (total/vollständig korrekt) of $\{P\}\ S\ \{Q\}$:

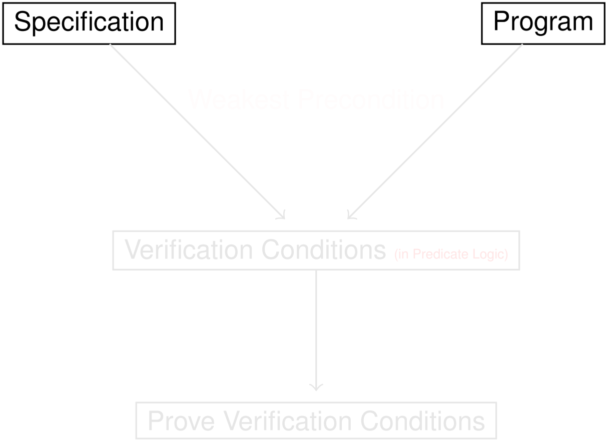
Every execution of $S$ that:

- starts in a state satisfying $P$,

terminates in a state satisfying $Q$.

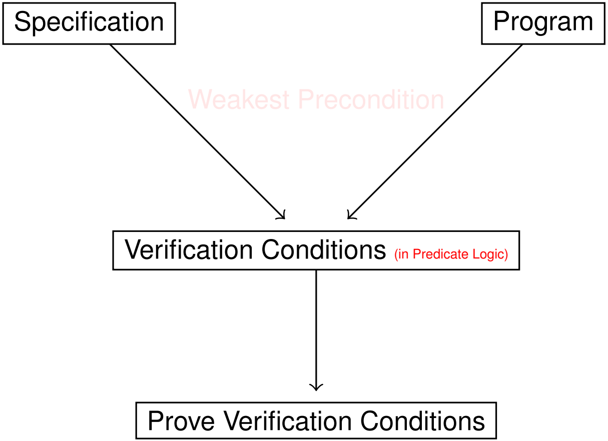
Total correctness = Partial correctness + Termination

# Verifying Program Correctness – the Process of Program Verification



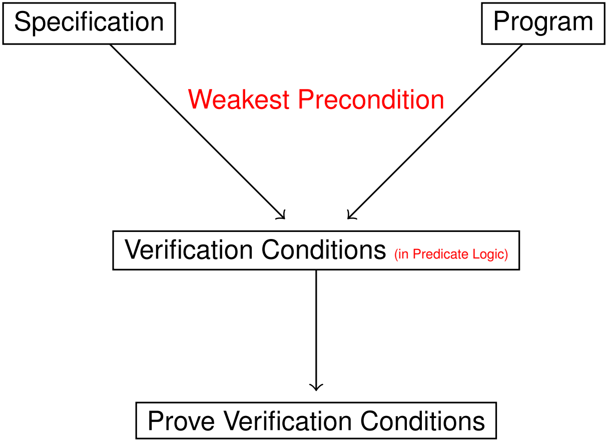

E. W. Dijsktra (1975)

# Verifying Program Correctness – the Process of Program Verification




E. W. Dijsktra (1975)

# Verifying Program Correctness – the Process of Program Verification




E. W. Dijsktra (1975)

# Weakest Precondition (WP) Strategy

Formula $P$ is **weaker** (schwächer) than formula $R$ iff $R \implies P$.

Weakest Precondition $\mathrm{wp}(S, Q)$ (schwächste Vorbedingung) for $S$ with $Q$:

for any $\{R\}\ S\ \{Q\}$ we have $R \implies \mathrm{wp}(S, Q)$.

Note: $\{\mathrm{wp}(S, Q)\}\ S\ \{Q\}$.

## Verification of $\{P\}\ S\ \{Q\}$:

$S = s_1; \ldots; s_{n-1}; s_n$

1. Compute $\mathrm{wp}(S, Q)$;
2. Prove $P \implies \mathrm{wp}(S, Q)$

# Weakest Precondition (WP) Strategy

Formula $P$ is **weaker** (schwächer) than formula $R$ iff $R \implies P$.

Weakest Precondition $\mathrm{wp}(S, Q)$ (schwächste Vorbedingung) for $S$ with $Q$:

for any $\{R\}\ S\ \{Q\}$ we have $R \implies \mathrm{wp}(S, Q)$.

Note: $\{\mathrm{wp}(S, Q)\}\ S\ \{Q\}$.

**Verification of $\{P\}\ S\ \{Q\}$:**

$S = s_1; \ldots; s_{n-1}; s_n$

1. Compute $\mathrm{wp}(S, Q)$;
2. Prove $P \implies \mathrm{wp}(S, Q)$

# Weakest Precondition (WP) Strategy

Formula $P$ is **weaker** (schwächer) than formula $R$ iff $R \implies P$.

Weakest Precondition $\mathrm{wp}(S, Q)$ (schwächste Vorbedingung) for $S$ with $Q$:

for any $\{R\}\ S\ \{Q\}$ we have $R \implies \mathrm{wp}(S, Q)$.

Note: $\{\mathrm{wp}(S, Q)\}\ S\ \{Q\}$.

**VERIFICATION OF $\{P\}\ S\ \{Q\}$:**

$S = s_1; \ldots; s_{n-1}; s_n$

1. Compute $\mathrm{wp}(S, Q)$;
2. Prove $P \implies \mathrm{wp}(S, Q)$

$\{P\}$

$s_1;$

$\vdots$

$s_{n-1};$

$s_n$

$\{Q\}$

# Weakest Precondition (WP) Strategy

Formula $P$ is **weaker** (schwächer) than formula $R$ iff $R \implies P$.

Weakest Precondition $\mathrm{wp}(S, Q)$ (schwächste Vorbedingung) for $S$ with $Q$:

for any $\{R\}\ S\ \{Q\}$ we have $R \implies \mathrm{wp}(S, Q)$.

Note: $\{\mathrm{wp}(S, Q)\}\ S\ \{Q\}$.

**Verification of $\{P\}\ S\ \{Q\}$:**

$S = s_1; \ldots; s_{n-1}; s_n$

1. Compute $\mathrm{wp}(S, Q)$;
2. Prove $P \implies \mathrm{wp}(S, Q)$

$\{P\}$

$s_1;$

$\vdots$

$s_{n-1};$

$\leftarrow \mathrm{wp}(s_n, Q)$

$s_n$

$\{Q\}$

# Weakest Precondition (WP) Strategy

Formula $P$ is **weaker** (schwächer) than formula $R$ iff $R \implies P$.

Weakest Precondition $\mathrm{wp}(S, Q)$ (schwächste Vorbedingung) for $S$ with $Q$:

for any $\{R\}\ S\ \{Q\}$ we have $R \implies \mathrm{wp}(S, Q)$.

Note: $\{\mathrm{wp}(S, Q)\}\ S\ \{Q\}$.

**Verification of $\{P\}\ S\ \{Q\}$:**

$S = s_1; \ldots; s_{n-1}; s_n$

1. Compute $\mathrm{wp}(S, Q)$;
2. Prove $P \implies \mathrm{wp}(S, Q)$

$\{P\}$

$s_1;$

$\vdots$

$\leftarrow \mathrm{wp}(s_{n-1}, \mathrm{wp}(s_n, Q))$

$s_{n-1};$

$\leftarrow \mathrm{wp}(s_n, Q)$

$s_n$

$\{Q\}$

# Weakest Precondition (WP) Strategy

Formula $P$ is **weaker** (schwächer) than formula $R$ iff $R \implies P$.

Weakest Precondition $\mathrm{wp}(S, Q)$ (schwächste Vorbedingung) for $S$ with $Q$:

for any $\{R\}\ S\ \{Q\}$ we have $R \implies \mathrm{wp}(S, Q)$.

Note: $\{\mathrm{wp}(S, Q)\}\ S\ \{Q\}$.

**Verification of $\{P\}\ S\ \{Q\}$:**

$S = s_1; \ldots; s_{n-1}; s_n$

1. Compute $\mathrm{wp}(S, Q)$;
2. Prove $P \implies \mathrm{wp}(S, Q)$

$$
\begin{array}{ll}
\{P\} & \\
 & \leftarrow \underbrace{\mathrm{wp}(s_1, \mathrm{wp}(\ldots, \mathrm{wp}(s_n, Q)))}_{\mathrm{wp}(S,Q)} \\
s_1; & \\
\vdots & \\
 & \leftarrow \mathrm{wp}(s_{n-1}, \mathrm{wp}(s_n, Q)) \\
s_{n-1}; & \\
 & \leftarrow \mathrm{wp}(s_n, Q) \\
s_n & \\
\{Q\} &
\end{array}
$$

# WP Rules

- **Scalar Assignments** ($x$ is a scalar variable, $A$ is arithmetic expression):

$$\mathrm{wp}(x := A,\ Q) \quad = \quad Q_{x \leftarrow A}$$

formula $Q_{x \leftarrow A}$ results from $Q$ by substituting every occurence of $x$ by $A$

$$\mathrm{wp}(x := 5,\ x + y = 6) \quad = \quad 5 + y = 6$$

$$\mathrm{wp}(x := x + 1,\ x + y = 6) \quad = \quad x + 1 + y = 6$$

- **Array Assignments** ($a$ is an array variable, $x$ is a scalar variable, $A$ is arithmetic expression):

$$\mathrm{wp}(a[x] := A,\ Q) \quad = \quad Q_{a \leftarrow a'}$$

formula $Q_{a \leftarrow a'}$ results from $Q$ by substituting every occurence of $a$ by array $a'$,

where $a'$ results from $a$ by replacing the $x$th element by $A$

$$\mathrm{wp}(a[1] := x + 1,\ a[1] = a[2]) \quad = \quad a'[1] = a'[2]$$

where $a'[1] = x + 1$ and $a'[i] = a[i]$ for every $i \neq 1$

$$= \quad x + 1 = a[2]$$

# WP Rules

- Scalar Assignments ($x$ is a scalar variable, $A$ is arithmetic expression):

$$\mathrm{wp}(x := A,\ Q) \quad = \quad Q_{x \leftarrow A}$$

formula $Q_{x \leftarrow A}$ results from $Q$ by substituting every occurence of $x$ by $A$

$$\mathrm{wp}(x := \underline{5},\ \underline{x} + y = 6) \quad = \quad \underline{5} + y = 6$$

$$\mathrm{wp}(x := \underline{x + 1},\ \underline{x} + y = 6) \quad = \quad \underline{x + 1} + y = 6$$

- Array Assignments ($a$ is an array variable, $x$ is a scalar variable, $A$ is arithmetic expression):

$$\mathrm{wp}(a[x] := A,\ Q) \quad = \quad Q_{a \leftarrow a'}$$

formula $Q_{a \leftarrow a'}$ results from $Q$ by substituting every occurence of $a$ by array $a'$,

where $a'$ results from $a$ by replacing the $x$th element by $A$

$$\mathrm{wp}(a[1] := \underline{x + 1},\ \underline{a[1]} = a[2]) \quad = \quad \underline{a'[1]} = \underline{a'[2]}$$

where $a'[1] = x + 1$ and $a'[i] = a[i]$ for every $i \neq 1$

$$= \quad x + 1 = a[2]$$

# WP Rules

- Scalar Assignments ($x$ is a scalar variable, $A$ is arithmetic expression):

$$\text{wp}(x := A,\ Q) \quad = \quad Q_{x \leftarrow A}$$

formula $Q_{x \leftarrow A}$ results from $Q$ by substituting every occurence of $x$ by $A$

$$\text{wp}(x := \underline{5},\ \underline{x} + y = 6) \quad = \quad \underline{5} + y = 6$$

$$\text{wp}(x := \underline{x + 1},\ \underline{x} + y = 6) \quad = \quad \underline{x + 1} + y = 6$$

- Array Assignments ($a$ is an array variable, $x$ is a scalar variable, $A$ is arithmetic expression):

$$\text{wp}(a[x] := A,\ Q) \quad = \quad Q_{a \leftarrow a'}$$

formula $Q_{a \leftarrow a'}$ results from $Q$ by substituting every occurence of $a$ by array $a'$,

where $a'$ results from $a$ by replacing the $x$th element by $A$

$$\text{wp}(a[1] := \underline{x + 1},\ \underline{a[1]} = a[2]) \quad = \quad \underline{a'[1]} = \underline{a'[2]}$$

where $a'[1] = x + 1$ and $a'[i] = a[i]$ for every $i \neq 1$

$$= \quad \underline{x + 1} = \underline{a[2]}$$

# WP Rules

- Scalar Assignments (*x* is a scalar variable, *A* is arithmetic expression):

$$\mathrm{wp}(x := A,\ Q) = Q_{x \leftarrow A}$$

formula $Q_{x \leftarrow A}$ results from $Q$ by substituting every occurence of $x$ by $A$

$$\mathrm{wp}(x := \underline{5},\ \underline{x} + y = 6) = \underline{5} + y = 6$$

$$\mathrm{wp}(x := \underline{x + 1},\ \underline{x} + y = 6) = \underline{x + 1} + y = 6$$

- Array Assignments (*a* is an array variable, *x* is a scalar variable, *A* is arithmetic expression):

$$\mathrm{wp}(a[x] := A,\ Q) = Q_{a \leftarrow a'}$$

formula $Q_{a \leftarrow a'}$ results from $Q$ by substituting every occurence of $a$ by array $a'$,

where $a'$ results from $a$ by replacing the $x$th element by $A$

$$\mathrm{wp}(a[1] := \underline{x + 1},\ \underline{a[1]} = a[2]) = \underline{a'[1]} = \underline{a'[2]}$$

where $a'[1] = x + 1$ and $a'[i] = a[i]$ for every $i \neq 1$

$$= \underline{x + 1} = a[2]$$

# WP Rules

- Scalar Assignments ($x$ is a scalar variable, $A$ is arithmetic expression):

$$\mathrm{wp}(x := A,\ Q) \quad = \quad Q_{x \leftarrow A}$$

formula $Q_{x \leftarrow A}$ results from $Q$ by substituting every occurence of $x$ by $A$

$$\mathrm{wp}(x := \underline{5},\ \underline{x} + y = 6) \quad = \quad \underline{5} + y = 6$$

$$\mathrm{wp}(x := \underline{x+1},\ \underline{x} + y = 6) \quad = \quad \underline{x+1} + y = 6$$

- Array Assignments ($a$ is an array variable, $x$ is a scalar variable, $A$ is arithmetic expression):

$$\mathrm{wp}(a[x] := A,\ Q) \quad = \quad Q_{a \leftarrow a'}$$

formula $Q_{a \leftarrow a'}$ results from $Q$ by substituting every occurence of $a$ by array $a'$,

where $a'$ results from $a$ by replacing the $x$th element by $A$

$$\mathrm{wp}(a[1] := \underline{x+1},\ \underline{a[1]} = a[2]) \quad = \quad \underline{a'[1]} = \underline{a'[2]}$$

where $a'[1] = x + 1$ and $a'[i] = a[i]$ for every $i \neq 1$

$$= \quad \underline{x+1} = a[2]$$

# WP Rules

- Sequencing:

$$\mathrm{wp}(s_1;\ s_2,\ Q) \quad = \quad \mathrm{wp}(s_1,\ \mathrm{wp}(s_2,\ Q))$$

$$\begin{aligned}
\mathrm{wp}(x := x+1; y := y+x,\ y > 10) \quad &= \quad \mathrm{wp}(x := x+1,\ \mathrm{wp}(y := \underline{y+x},\ \underline{y} > 10)) \\
&= \quad \mathrm{wp}(x := \underline{x+1},\ \ y + \underline{x} > 10) \\
&= \quad y + x + 1 > 10
\end{aligned}$$

# WP Rules

- Sequencing:

$$\mathrm{wp}(s_1;\ s_2,\ Q) \quad = \quad \mathrm{wp}(s_1,\ \mathrm{wp}(s_2,\ Q))$$

$$\begin{aligned}
\mathrm{wp}(x := x + 1; y := y + x,\ y > 10) &= \mathrm{wp}(x := x + 1,\ \mathrm{wp}(y := \underline{y + x},\ \underline{y} > 10)) \\
&= \mathrm{wp}(x := \underline{x + 1},\ y + \underline{x} > 10) \\
&= y + x + 1 > 10
\end{aligned}$$

# WP Rules

- Sequencing:

$$\mathrm{wp}(s_1;\ s_2,\ Q) \quad = \quad \mathrm{wp}(s_1,\ \mathrm{wp}(s_2,\ Q))$$

$$\begin{aligned}
\mathrm{wp}(x := x+1; y := y+x,\ y > 10) \quad &= \quad \mathrm{wp}(x := x+1,\ \mathrm{wp}(y := \underline{y+x},\ \underline{y} > 10)) \\
&= \quad \mathrm{wp}(x := \underline{x+1},\ \ y + \underline{x} > 10) \\
&= \quad y + x + 1 > 10
\end{aligned}$$

# WP Rules

- Sequencing:

$$\text{wp}(s_1;\ s_2,\ Q) \quad = \quad \text{wp}(s_1,\ \text{wp}(s_2,\ Q))$$

$$\begin{aligned}
\text{wp}(x := x + 1; y := y + x,\ y > 10) \quad &= \quad \text{wp}(x := x + 1,\ \text{wp}(y := \underline{y + x},\ \underline{y} > 10)) \\
&= \quad \text{wp}(x := \underline{x + 1},\ \ y + \underline{x} > 10) \\
&= \quad y + x + 1 > 10
\end{aligned}$$

# WP Rules

- Sequencing:

$$\text{wp}(s_1;\ s_2,\ Q) \quad = \quad \text{wp}(s_1,\ \text{wp}(s_2,\ Q))$$

$$\begin{aligned}
\text{wp}(x := x+1; y := y+x,\ y > 10) &= \text{wp}(x := x+1,\ \text{wp}(y := \underline{y+x},\ \underline{y} > 10)) \\
&= \text{wp}(x := \underline{x+1},\ \ y + \underline{x} > 10) \\
&= y + x + 1 > 10
\end{aligned}$$

# WP Rules

- Conditionals:

$$\text{wp}(\underline{\text{if}}\ (B)\ \underline{\text{then}}\ s_1\ \underline{\text{else}}\ s_2,\ Q) \quad = \quad (B \implies \text{wp}(s_1,\ Q)) \wedge (\neg B \implies \text{wp}(s_2,\ Q))$$

# WP Rules

- Conditionals:

$$\text{wp}(\underline{\text{if}}\ (B)\ \underline{\text{then}}\ s_1\ \underline{\text{else}}\ s_2,\ Q) \quad = \quad (B \implies \text{wp}(s_1,\ Q)) \wedge (\neg B \implies \text{wp}(s_2,\ Q))$$

Special Case:

$$\text{wp}(\underline{\text{if}}\ (B)\ \underline{\text{then}}\ s_1,\ Q) \quad = \quad (B \implies \text{wp}(s_1,\ Q)) \wedge (\neg B \implies Q)$$

# WP Rules

- Conditionals:

$$\text{wp}(\underline{\text{if}}\ (B)\ \underline{\text{then}}\ s_1\ \underline{\text{else}}\ s_2,\ Q) \quad = \quad (B \implies \text{wp}(s_1,\ Q)) \wedge (\neg B \implies \text{wp}(s_2,\ Q))$$

## Example revisited: Maximum of Two Natural Numbers

Postcondition $Q$: $(max \geq x) \wedge (max \geq y) \wedge (max = x \vee max = y)$

$$\text{wp}(\underline{\text{if}}\ x \geq y\ \underline{\text{then}}\ max := x\ \underline{\text{else}}\ max := y,\ Q) \ =$$

$$(x \geq y \implies \text{wp}(max := \underline{x},\ Q)) \wedge (x < y \implies \text{wp}(max := \underline{y},\ Q)) \ =$$

$$(x \geq y \implies Q_{max \leftarrow x}) \wedge (x < y \implies Q_{max \leftarrow y}) =$$

$$\Big(x \geq y \implies ((\underline{x} \geq x) \wedge (\underline{x} \geq y) \wedge (\underline{x} = x \vee \underline{x} = y))\Big)$$
$$\wedge$$
$$\Big((x < y \implies ((\underline{y} \geq x) \wedge (\underline{y} \geq y) \wedge (\underline{y} = x \vee \underline{y} = y))\Big)$$

# WP Rules

- Conditionals:

$$\text{wp}(\underline{\text{if}}\ (B)\ \underline{\text{then}}\ s_1\ \underline{\text{else}}\ s_2,\ Q) \quad = \quad (B \implies \text{wp}(s_1,\ Q)) \wedge (\neg B \implies \text{wp}(s_2,\ Q))$$

## Example revisited: Maximum of Two Natural Numbers

Postcondition $Q$: $(max \geq x) \wedge (max \geq y) \wedge (max = x \vee max = y)$

$$\text{wp}(\underline{\text{if}}\ x \geq y\ \underline{\text{then}}\ max := x\ \underline{\text{else}}\ max := y,\ Q) \ =$$

$$(x \geq y \implies \text{wp}(max := \underline{x},\ Q)) \wedge (x < y \implies \text{wp}(max := \underline{y},\ Q)) \ =$$

$$(x \geq y \implies Q_{max \leftarrow x}) \wedge (x < y \implies Q_{max \leftarrow y}) =$$

$$\Big(x \geq y \implies ((\underline{x} \geq x) \wedge (\underline{x} \geq y) \wedge (\underline{x} = x \vee \underline{x} = y))\Big)$$

$$\wedge$$

$$\Big((x < y \implies ((\underline{y} \geq x) \wedge (\underline{y} \geq y) \wedge (\underline{y} = x \vee \underline{y} = y))\Big)$$

# WP Rules

- Conditionals:

$$\text{wp}(\underline{\text{if}}\ (B)\ \underline{\text{then}}\ s_1\ \underline{\text{else}}\ s_2,\ Q) \quad = \quad (B \implies \text{wp}(s_1,\ Q)) \land (\neg B \implies \text{wp}(s_2,\ Q))$$

## Example revisited: Maximum of Two Natural Numbers

Postcondition $Q$: $(max \geq x) \land (max \geq y) \land (max = x \lor max = y)$

$$\text{wp}(\underline{\text{if}}\ x \geq y\ \underline{\text{then}}\ max := x\ \underline{\text{else}}\ max := y,\ Q) \ =$$

$$(x \geq y \implies \text{wp}(max := \underline{x},\ Q)) \land (x < y \implies \text{wp}(max := \underline{y},\ Q)) \ =$$

$$(x \geq y \implies Q_{max \leftarrow x}) \land (x < y \implies Q_{max \leftarrow y}) \ =$$

$$\Big(x \geq y \implies ((\underline{x} \geq x) \land (\underline{x} \geq y) \land (\underline{x} = x \lor \underline{x} = y))\Big)$$

$$\land$$

$$\Big((x < y \implies ((\underline{y} \geq x) \land (\underline{y} \geq y) \land (\underline{y} = x \lor \underline{y} = y))\Big)$$

# WP Rules

- Conditionals:

$$\text{wp}(\underline{\text{if}}\ (B)\ \underline{\text{then}}\ s_1\ \underline{\text{else}}\ s_2,\ Q) \quad = \quad (B \implies \text{wp}(s_1,\ Q)) \wedge (\neg B \implies \text{wp}(s_2,\ Q))$$

## Example revisited: Maximum of Two Natural Numbers

Postcondition $Q$: $(max \geq x) \wedge (max \geq y) \wedge (max = x \vee max = y)$

$$\text{wp}(\underline{\text{if}}\ x \geq y\ \underline{\text{then}}\ max := x\ \underline{\text{else}}\ max := y,\ Q) \ =$$

$$(x \geq y \implies \text{wp}(max := \underline{x},\ Q)) \wedge (x < y \implies \text{wp}(max := \underline{y},\ Q)) \ =$$

$$(x \geq y \implies Q_{max \leftarrow x}) \wedge (x < y \implies Q_{max \leftarrow y}) =$$

$$\Big(x \geq y \implies ((\underline{x} \geq x) \wedge (\underline{x} \geq y) \wedge (\underline{x} = x \vee \underline{x} = y))\Big)$$

$$\wedge$$

$$\Big((x < y \implies ((\underline{y} \geq x) \wedge (\underline{y} \geq y) \wedge (\underline{y} = x \vee \underline{y} = y))\Big)$$

# WP Rules

- Loops $L$ $\equiv$ while ($B$) do $s$ end while :

$$wp(\underline{\text{while}}\ (B)\ \underline{\text{do}}\ \ s\ \ \underline{\text{end while}},\ Q) \quad = \quad I$$

where $I$ is a loop invariant ($I$ is invariant/remains unchanged) (Schlaufen-Invariant)

# WP Rules

- Loops $L \equiv$ while ($B$) do $s$ end while :

$$\text{wp}(\text{while } (B) \text{ do } s \text{ end while}, Q) = I$$

where $I$ is a loop invariant ($I$ is invariant/remains unchanged) (Schlaufen-Invariant)

use conditional together with loop: || instead of a single loop:

{wp($L$, $Q$)}

if ($B$) then $s$;

while ($B$) do $s$ end while

while ($B$) do $s$ end while

{$Q$}

# WP Rules

- Loops $L \equiv$ while $(B)$ do $s$ end while :

$$wp(\text{while } (B) \text{ do } s \text{ end while}, Q) = I$$

where $I$ is a loop invariant ($I$ is invariant/remains unchanged) (Schlaufen-Invariant)

use conditional together with loop: || instead of a single loop:

$\{wp(L, Q)\}$

$\{wp(L, Q)\}$

| |
|---|
| if $(B)$ then $s$; |
| while $(B)$ do $s$ end while |

$\{Q\}$

while $(B)$ do $s$ end while

# WP Rules

- Loops $L \equiv$ while $(B)$ do $s$ end while :

$$\text{wp}(\text{while } (B) \text{ do } s \text{ end while}, Q) = I$$

where $I$ is a loop invariant ($I$ is invariant/remains unchanged) (Schlaufen-Invariant)

use conditional together with loop: || instead of a single loop:

$\{\text{wp}(L, Q)\}$

$\{\text{wp}(L, Q)\}$

| if $(B)$ then $s$; |
| --- |
| if $(B)$ then $s$;<br>while $(B)$ do $s$ end while |

$$\frac{\text{while } (B) \text{ do } s \text{ end while}}{\text{while } (B) \text{ do } s \text{ end while}}$$

$\{Q\}$

# WP Rules

- Loops $L \equiv$ while $(B)$ do $s$ end while :

$$\text{wp}(\text{while } (B) \text{ do } s \text{ end while}, Q) = I$$

where $I$ is a loop invariant ($I$ is invariant/remains unchanged) (Schlaufen-Invariant)

1. $I \wedge B \implies I'$, where $I' = \text{wp}(S, I)$;
2. $I \wedge \neg B \implies Q$.

**LOOP INVARIANTS (INDUCTIVE ASSERTIONS):**

evaluate to true before and after each loop iteration

$I$ is an invariant for $\{P\}$ while $(B)$ do $s$ end while $\{Q\}$ iff:

0. initial condition: $P \implies I$;
1. iterative (inductive) condition: $\{I \wedge B\}\ s\ \{I\}$;
2. final condition: $I \wedge \neg B \implies Q$

# WP Rules

- Loops $L \equiv$ while $(B)$ do $s$ end while :

$$\text{wp}(\text{while } (B) \text{ do } s \text{ end while}, \; Q) \;=\; I$$

where $I$ is a loop invariant ($I$ is invariant/remains unchanged) (Schlaufen-Invariant)

1. $I \wedge B \implies I'$, where $I' = \text{wp}(S, I)$;
2. $I \wedge \neg B \implies Q$.

**Loop Invariants (Inductive Assertions):**

evaluate to true before and after each loop iteration

$I$ is an invariant for $\{P\}$ while $(B)$ do $s$ end while $\{Q\}$ iff:

0. initial condition: $P \implies I$;
1. iterative (inductive) condition: $\{I \wedge B\}\; s\; \{I\}$;
2. final condition: $I \wedge \neg B \implies Q$

# WP Rules

- Loops $L \equiv$ while $(B)$ do $s$ end while :

$$\text{wp}(\text{while } (B) \text{ do } s \text{ end while}, \; Q) \quad = \quad I$$

where $I$ is a loop invariant ($I$ is invariant/remains unchanged) (Schlaufen-Invariant)

and VERIFICATION CONDITIONS:

1. $I \wedge B \implies I'$, where $I' = \text{wp}(s, \; I)$;
2. $I \wedge \neg B \implies Q$.

## LOOP INVARIANTS (INDUCTIVE ASSERTIONS):

evaluate to true before and after each loop iteration

$I$ is an invariant for $\{P\}$ while $(B)$ do $s$ end while $\{Q\}$ iff:

0. initial condition: $P \implies I$;
1. iterative (inductive) condition: $\{I \wedge B\}\; s\; \{I\}$;
2. final condition: $I \wedge \neg B \implies Q$

# WP Rules

- Loops $L \equiv$ while $(B)$ do $s$ end while :

$$\text{wp}(\text{while } (B) \text{ do } s \text{ end while}, \ Q) = I$$

where $I$ is a loop invariant ($I$ is invariant/remains unchanged) (Schlaufen-Invariant)

and VERIFICATION CONDITIONS:

1. $I \wedge B \implies I'$, where $I' = \text{wp}(s, \ I)$;
2. $I \wedge \neg B \implies Q$.

**VERIFICATION OF** $\{P\}$ WHILE $(B)$ DO $s$ END WHILE $\{Q\}$ :

- Compute wp(while $(B)$ do $s$ end while , $Q$) $= I$;
- Prove VERIFICATION CONDITIONS:

0. $P \implies I$;
1. $I \wedge B \implies I'$, where $I' = \text{wp}(s, \ I)$;
2. $I \wedge \neg B \implies Q$.

# Example revisited: Integer Division ANNOTATED with Invariant

Precondition $P$: $(x \geq 0) \wedge (y > 0)$

Postcondition $Q$: $(quo * y + rem = x) \wedge (0 \leq rem < y)$

Loop *DivLoop*:

Invariant $I$ : $(quo * y + rem = x) \wedge (0 \leq rem) \wedge (0 < y) \wedge (x \geq 0)$

<u>while</u> $(y \leq rem)$ <u>do</u>

  $rem := rem - y;\ quo := quo + 1$

<u>end while</u>

$$\mathbf{wp}(DivLoop, Q) \quad = \quad \underbrace{(quo * y + rem = x) \wedge (0 \leq rem) \wedge (0 < y) \wedge (x \geq 0)}_{I}$$

VERIFICATION CONDITIONS:

$P \implies I$

$I \wedge (y \leq rem) \implies ((quo + 1) * y + (rem - y) = x) \wedge (0 \leq rem - y) \wedge (0 < y) \wedge (x \geq 0)$

$I \wedge (y > rem) \implies Q$

## Example revisited: Integer Division ANNOTATED with invariant

Precondition $P$: $(x \geq 0) \wedge (y > 0)$

Postcondition $Q$: $(quo * y + rem = x) \wedge (0 \leq rem < y)$

Loop *DivLoop*:

*Invariant I* : $(quo * y + rem = x) \wedge (0 \leq rem) \wedge (0 < y) \wedge (x \geq 0)$
<u>while</u> $(y \leq rem)$ <u>do</u>
    $rem := rem - y; \; quo := quo + 1$
<u>end while</u>

$$\mathbf{wp}(DivLoop, Q) \quad = \quad \underbrace{(quo * y + rem = x) \wedge (0 \leq rem) \wedge (0 < y) \wedge (x \geq 0)}_{I}$$

VERIFICATION CONDITIONS:

$P \implies I$

$I \wedge (y \leq rem) \implies ((quo + 1) * y + (rem - y) = x) \wedge (0 \leq rem - y) \wedge (0 < y) \wedge (x \geq 0)$

$I \wedge (y > rem) \implies Q$

## Example revisited: Integer Division ANNOTATED with invariant

Precondition $P$: $(x \geq 0) \wedge (y > 0)$

Postcondition $Q$: $(quo * y + rem = x) \wedge (0 \leq rem < y)$

Loop *DivLoop*:

*Invariant I* : $(quo * y + rem = x) \wedge (0 \leq rem) \wedge (0 < y) \wedge (x \geq 0)$
while $(y \leq rem)$ do
    $rem := rem - y;\ quo := quo + 1$
end while

$$\mathbf{wp}(DivLoop, Q) \quad = \quad \underbrace{(quo * y + rem = x) \wedge (0 \leq rem) \wedge (0 < y) \wedge (x \geq 0)}_{I}$$

VERIFICATION CONDITIONS:

$P \implies I$

$I \wedge (y \leq rem) \implies ((quo + 1) * y + (rem - y) = x) \wedge (0 \leq rem - y) \wedge (0 < y) \wedge (x \geq 0)$

$I \wedge (y > rem) \implies Q$

## Example revisited: Integer Division ANNOTATED with invariant

Precondition $P$: $(x \geq 0) \wedge (y > 0)$

Postcondition $Q$: $(quo * y + rem = x) \wedge (0 \leq rem < y)$

Loop *DivLoop*:

*Invariant I* : $(quo * y + rem = x) \wedge (0 \leq rem) \wedge (0 < y) \wedge (x \geq 0)$
while $(y \leq rem)$ do
  $rem := rem - y;\ quo := quo + 1$
end while

$$\mathbf{wp}(DivLoop, Q) \quad = \quad \underbrace{(quo * y + rem = x) \wedge (0 \leq rem) \wedge (0 < y) \wedge (x \geq 0)}_{I}$$

VERIFICATION CONDITIONS:

$$P \implies I$$

$$I \wedge (y \leq rem) \implies ((quo + 1) * y + (rem - y) = x) \wedge (0 \leq rem - y) \wedge (0 < y) \wedge (x \geq 0)$$

$$I \wedge (y > rem) \implies Q$$

# Weakest Precondition Strategy – Revised Summary

## VERIFICATION OF $\{P\}\ S\ \{Q\}$:

$S = s_1; \ldots; s_{n-1}; s_n$

1. Compute $\text{wp}(S, Q)$;
2. Prove:
   - $P \implies \text{wp}(S, Q)$ ;
   - **additional verification conditions**

$\{P\}$

$\quad \leftarrow \underbrace{\text{wp}(s_1, \text{wp}(\ldots, \text{wp}(s_n, Q)))}_{\text{wp}(S,Q)}$

$s_1;$

$\vdots$

$\quad \leftarrow \text{wp}(s_{n-1}, \text{wp}(s_n, Q))$ $\qquad \uparrow$ verification conditions

$s_{n-1};$

$\quad \leftarrow \text{wp}(s_n, Q)$

$s_n$

$\{Q\}$

# Example

## Example (Integer Division.)

Verify the partial correctness of the *annotated* $\{P\}\ S\ \{Q\}$, where:

$P$: $(x \geq 0) \wedge (y > 0)$

$Q$: $(quo * y + rem = x) \wedge (0 \leq rem < y)$

*Annotated S* (S annotated with invariant):

$quo := 0;\ \ rem := x;$

<u>invariant</u> $(quo * y + rem = x) \wedge (0 \leq rem) \wedge (0 < y) \wedge (x \geq 0)$

<u>while</u> $(y \leq rem)$ <u>do</u>

  $rem := rem - y;\ \ quo := quo + 1$

<u>end while</u>

Verification Conditions:

$(x \geq 0) \wedge (y > 0) \implies$
$(x = x) \wedge x \geq 0 \wedge x \geq 0 \wedge y > 0$

$(x = rem + y * quo) \wedge x \geq 0 \wedge rem \geq 0 \wedge y > 0 \wedge y \leq rem \implies$
$(x = (rem - y) + y * (quo + 1)) \wedge x \geq 0 \wedge rem - y \geq 0 \wedge y > 0$

$(x = rem + y * quo) \wedge x \geq 0 \wedge rem \geq 0 \wedge y > 0 \wedge y > rem \implies$
$(x = rem + y * quo) \wedge 0 \leq rem < y$

# Example

## Example (Integer Division.)

Verify the partial correctness of the *annotated* $\{P\}\ S\ \{Q\}$, where:

$P$: $(x \geq 0) \wedge (y > 0)$

$Q$: $(quo * y + rem = x) \wedge (0 \leq rem < y)$

*Annotated* $S$ ($S$ annotated with invariant):

$quo := 0;\ rem := x;$
invariant $(quo * y + rem = x) \wedge (0 \leq rem) \wedge (0 < y) \wedge (x \geq 0)$
while $(y \leq rem)$ do
  $rem := rem - y;\ quo := quo + 1$
end while

### Verification Conditions:

$$(x \geq 0) \wedge (y > 0) \implies (x = x) \wedge x \geq 0 \wedge x \geq 0 \wedge y > 0$$

$$(x = rem + y * quo) \wedge x \geq 0 \wedge rem \geq 0 \wedge y > 0 \wedge y \leq rem \implies (x = (rem - y) + y * (quo + 1)) \wedge x \geq 0 \wedge rem - y \geq 0 \wedge y > 0$$

$$(x = rem + y * quo) \wedge x \geq 0 \wedge rem \geq 0 \wedge y > 0 \wedge y > rem \implies (x = rem + y * quo) \wedge 0 \leq rem < y$$

## Exercise (1)

*Is the Hoare triple* $\{x := 1\}\ x := x + 1; y := x + 1\ \{y \geq 2\}$ *correct?*

## Exercise (2)

*Compute:* $wp(t := x; x := y; y := t,\ x = Y \wedge y = X)$.

## Exercise (3)

*Verify the partial correctness of the annotated* $\{P\}\ S\ \{Q\}$, *where:*

*P:* $x = 0 \wedge y = 0$

*Q:* $x = 10 \wedge y = 10$

*Annotated S:* invariant $(x = y) \wedge (x \leq 10)$
while $(x < 10)$ do $x := x + 1;\ y := y + 1$ end while

## Exercise (4)

*Consider the Hoare triple* $\{P\}\ S\ \{Q\}$, *where:*

*P:* $x = 0$

*Q:* $x = 5$

*S:* while $(x < 5)$ do $x := x + 1$ end while

- *Is* $x \leq 5$ *an invariant?*
- *Is* $x < 5$ *an invariant?*
- *Is* $x = 5$ *an invariant?*