1 Introduction

1. Verification vs Synthesis
2. Synthesis
3. Synthesis of Alternative Commands
4. Synthesis of Loops
5. Examples

1.1 Verification

Verification problem: Is the implementation correct wrt specification?
Problem Instance: A Program, P
Specification: A property, \( \phi \)
Question: Does \( P \models \phi \) ?
Answer: Verification yields the answer Yes/No

1.2 Synthesis

Synthesis problem: Generate a provably correct implementation from the specification.
Specification: A Precondition \( Q \) and a Postcondition \( R \)
Implementation: Synthesis generates the implementation \( S \).

1.2.1 Coffee Can Problem

Problem: We have a can of beans, black and white. Take two beans and If they are of the

1. same colour, throw them both away and put a black bean in the can
2. different colour, throw away the black bean and put the white bean back in the can

**Question**: What is the colour of the final bean based on the number of white beans and black beans initially in the can?

### 1.2.2 Idea of Solution

1. Start with the PostCondition $R$. Either the last bean is black or white. If it is black, then there is an odd number of black beans in the can. Is this an invariant?

2. Both actions increase/decrease the number of black beans by one. So "odd number of black beans" cannot be an invariant property.

3. What about "even number of white beans"? Since, the number of white beans are either decreased by two or unchanged by the actions, "even number of white beans" is an invariant property.

4. If the last bean is white, similarly, the invariant is "odd number of white beans".

### 2 Synthesis Principle

Before attempting to solve the problem, make absolutely sure

- you know what the problem is.

That means before developing a program,

- make precise pre- and postconditions
- refine them.

#### 2.1 Synthesis of Alternate Commands

- Given a precondition $Q$ and a postcondition $R$, construct a program $IF$ such that $\{Q\}IF\{R\}$ holds.

1. Find a command $C$ whose execution will establish the postcondition $R$ in at least some cases

2. Find a boolean condition $B$ satisfying $Q \land B \Rightarrow wp("C", R)$ and put them together to form $B \rightarrow C$

3. Repeat above two steps until $Q \Rightarrow BB$
Example 1 Write a program that interchanges the values of $x$ and $y$ such that $x \leq y$.

Precondition $Q : x = X \land y = Y$

Postcondition $R : x \leq y \land ((x = X \land y = Y) \lor (x = Y \land y = X))$

Solution :

1. Find a command that establishes $R$ in some cases. Since $Q$ is a part of the second conjunct in $R$, \texttt{skip} is a good command.

2. Find $B_1$ such that $Q \land B_1 \Rightarrow wp("\texttt{skip}", R)$. Since $wp("\texttt{skip}", R) = R$ and $Q$ implies the second conjunct of $R$, choose $B_1 = x \leq y$(the first conjunct).

3. $Q \Rightarrow B_1$ ? No, find a new command.

4. $x,y := y,x$ would also satisfy the second conjunct of $R$.

5. Find $B_2$ such that $Q \land B_2 \Rightarrow wp("x := y,x", R)$. 

\[
wp("x := y,x", R) = y \leq x \land ((y = X \land x = Y) \lor (y = Y \land x = X))
\]

$Q$ implies the second conjunct of the weakest precondition, choose $B_2 = y \leq x$(the first conjunct).

6. $Q \Rightarrow (B_1 \lor B_2)$ ?

Assume $Q$.

$B_1 \lor B_2 = x \leq y \lor y \leq x = T$. Done !!

if x \leq y \rightarrow \texttt{skip} 
\hfill \Box y \leq x \rightarrow x,y := y,x 
fi

2.2 Synthesis of Loops

Given a precondition $Q$ and a postcondition $R$, construct a program $LOOP$ such that \{Q\}LOOP\{R\} holds.

Produce $LOOP$ in two steps.

1. From $Q$ and $R$, we derive an invariant $P$ and a bound function $t$.

2. From $Q, R, P$ and $t$, we derive a program $LOOP$. 
2.2.1 Developing Loops from Invariants and bounds

Given

- a precondition $Q$
- a postcondition $R$
- an invariant $P$
- a bound function $t$

Construct a program $LOOP$ such that $\{Q\} LOOP \{R\}$ holds.

**Developing Loops : Method 1**

Given $Q, R, P$ and $t$, a program of the following form will be constructed.

\[
S_{init}; \\
\text{do } B \rightarrow S_{inv}; S_t \text{ od}
\]

1. Find $B$ such that $P \land \neg B \Rightarrow R$
2. Find a statement $S_t$ that decreases $t$
3. Find a statement $S_{inv}$ such that $P$ is preserved. That means find out if
   \[(P \land B \Rightarrow wp("S_{inv}; S_t", P)] = (P \land B \Rightarrow wp("S_{inv}", wp("S_t", P)))
   \]
   holds. To find $S_{inv}$
   (a) Compute $P' = wp("S_t", P)$
   (b) Compare $P'$ with $P \land B$ to find $S_{inv}$ such that $P \land B \Rightarrow wp("S_{inv}", P')$.
   Prove that finally using $B$ as guard $P \land B \Rightarrow wp("S_{inv}", P')$ holds.
4. Find a statement $S_{init}$ such that $P$ holds initially. Prove $q \Rightarrow wp("S_{init}", P)$.

**Example 2** Write a program that computes $a^n$ using repeated multiplication.

\[Precondition \quad Q : \quad n \geq 0\]
\[Postcondition \quad R : \quad r = a^n\]
\[Invariant \quad P : \quad r = a^i \land 0 \leq i \leq n\]
\[Bound t : \quad n - i \]

**Solution:**

1. Find $B$ such that $P \land \neg B \Rightarrow R$. From the bound function, we know that $n - i \leq 0$ on termination. Therefore, $\neg B = (i = n)$, i.e. $B = (i \neq n)$
2. Find a statement $S_t$ that decreases $t$. $i := i + 1$
3. Find a statement $S_{\text{inv}}$ such that $P$ is preserved.

\[ P' = \text{wp}("S_1", P) \]
\[ = \text{wp}("i := i + 1", r = a^i \& 0 \leq i \leq n) \]
\[ = r = a^{i+1} \& 0 \leq i + 1 \leq n \]

$S_{\text{inv}} = r := r \times a$. Prove that $P \land B \Rightarrow \text{wp}("S_{\text{inv}}", P')$.

4. Find a statement $S_{\text{init}}$ such that $P$ holds initially. $r, i := 1, 0$. Prove $Q \Rightarrow \text{wp}("S_{\text{init}}", P)$.

\[
\begin{align*}
r, i &:= 1, 0 \\
\text{do} & \\
& i \neq n \rightarrow r := r \times a; i := i + 1 \\
\text{od}
\end{align*}
\]

Developing Loops : Method 2

1. Find a statement $S$ which decreases $t$.

2. Find a guard $B$ such that $B \rightarrow S$ preserves the invariant, i.e $P \land B \Rightarrow \text{wp}("S", P)$.

3. $P \land \neg B \Rightarrow R$?

   **YES**: finished!
   **NO**: repeat steps 1-3.

4. Find a statement $S_{\text{init}}$ such that $P$ holds initially.

Example 3 Create a program that computes the gcd of two numbers, $x_0$ and $y_0$.

\[
\begin{align*}
\text{Precondition } Q : & \quad 0 < x_0 \land 0 < y_0 \\
\text{Postcondition } R : & \quad y = \text{gcd}(x_0, y_0) \\
\text{Invariant } P : & \quad 0 < x \land 0 < y \quad \land \text{gcd}(x, y) = \text{gcd}(x_0, y_0) \\
\text{Bound } t : & \quad x + y
\end{align*}
\]

1. $\text{gcd}(x, y) = \text{gcd}(x + y, y)$
2. $\text{gcd}(x, y) = \text{gcd}(x, y + x)$
3. $\text{gcd}(x, y) = \text{gcd}(x - y, y)$
4. $\text{gcd}(x, y) = \text{gcd}(x, y - x)$
5. $\text{gcd}(x, y) = \text{gcd}(-x, y)$
6. $\text{gcd}(x, y) = \text{gcd}(x, -y)$
7. $\text{gcd}(x, 0) = x$
8. \( \gcd(0, y) = y \)

9. \( \gcd(x, x) = x \)

**Solution:**

1. Find a statement that decreases the bound function, use properties of \( \gcd \). You cannot use 1, 2, 5, 6. 1 and 2 increases the bound function and 5, 6 violates the invariant property "0 < x, y". Use property 3, i.e, \( S_1 = x := x - y \)

2. Find guard \( B_1 \) s.t \( B_1 \rightarrow S_1 \) preserves the invariant, i.e \( P \land B_1 \Rightarrow wp("x := x - y", P) \)

\[
wp("x := x - y", P) \\
= 0 < x - y \land 0 < y \land \gcd(x - y, y) = \gcd(x_0, y_0) \\
= y < x \land 0 < y \land \gcd(x - y, y) = \gcd(x_0, y_0)
\]

Choose \( B_1 = y < x \).

3. \( P \land \neg B_1 \Rightarrow R? \) NO!

4. Find a new statement that decreases the bound function. Use property 4, i.e, \( S_2 = y := y - x \)

5. Find guard \( B_2 \) s.t \( B_2 \rightarrow S_2 \) preserves the invariant, i.e \( P \land B_2 \Rightarrow wp("y := y - x", P) \)

\[
wp("y := y - x", P) \\
= 0 < x \land 0 < y - x \land \gcd(x - x, y - x) = \gcd(x_0, y_0) \\
= 0 < x \land x < y \land \gcd(x - y, y) = \gcd(x_0, y_0)
\]

Choose \( B_2 = x < y \).

6. \( P \land \neg(B_1 \lor B_2) \Rightarrow R? \) YES! (Prove)

The program is:

\[
\begin{align*}
\text{\texttt{\textbf{do}}} & \quad y < x \rightarrow x := x - y \\
\text{\texttt{\textbf{\box}}}& \quad x < y \rightarrow y := y - x \\
\text{\texttt{od}} \\
\{R : y = \gcd(x_0, y_0)\}
\end{align*}
\]
3 Developing Invariants

Given a postcondition \( R \), find \( P \) and \( t \) and an initial state \( IS \). The initial state should be easy to establish.

To develop an invariant, weaken \( R \) to include an initial state \( IS \) such that \( IS \) is easy to establish.

Different strategies of weakening \( R \) exists.

1. Delete a conjunct
2. Replace a constant with a variable
3. Enlarge the range of a variable
4. Add a disjunct (Too hard!)

Delete a Conjunct

Example 4 Linear Search

Given an array \( b[0:m] \), find the first value \( x \) in \( b[0:m] \), the value is known to exist.

Precondition \( Q : 0 < m \land x \in b[0:m] \)

Postcondition \( R : 0 \leq i \leq m \land x \notin b[0:i-1] \land x = b[i] \)

Find \( P \) and \( t \) and develop a program that satisfies the specification. Investigate \( R \). A suitable initial state can be \( i:=0 \) and \( R \) includes the initial state. \( x = b[i] \) is the tough one and we delete it.

\[
P : \quad 0 \leq i < m \land x \notin b[0:i-1] \\
t : \quad m - i
\]

Develop the program using the method 1.

1. Find \( B \) such that \( P \land \neg B \Rightarrow R \). We note that \( P \land \neg B \Rightarrow R \) is true if \( P \land \neg B = R \), i.e \( \neg BB \) is the removed conjunct. Use \( BB \), the negation of the removed conjunct as guard. Therefore, \( \neg B = (x = b[i]) \), i.e, \( B = (i \neq b[i]) \)

2. Find a statement \( S_t \) that decreases \( t \). \( i := i + 1 \)

3. Find a statement \( S_{inv} \) such that \( P \) is preserved.

\[
P' = \quad wp("S_t", P) \\
= \quad wp("i := i + 1", P) \\
= \quad 0 \leq i + 1 < m \land x \notin b[0 : i]
\]

Compare \( P' \) with \( P \land B \) to find \( S_{inv} \) such that \( P \land B \Rightarrow wp("S_{inv}", P') \).

Since, \( P \land B \Rightarrow P' (Homework!) \), \( S_{inv} = \text{skip} \) will do.
4. Find a statement $S_{init}$ such that $P$ holds initially. $i := 0$

The program:

$$
i := 0
\begin{array}{c}
do \quad x \notin b[i] \rightarrow i := i + 1 \\
d\end{array}
$$

Replace a constant with a variable

**Example 5** Given an array $b[0:m]$, develop a program that computes the sum of all the elements of $b[0:m]$.

Precondition $Q : m + 1 \geq 0$

Postcondition $R : r = (\Sigma i : 0 \leq i \leq m + 1 : b[i])$

Find $P$ and $t$ and develop a program that satisfies the specification.

**Solution:** Weaken $R$ by replacing the constant $m + 1$ with a variable $j$ and puts bound on $j$.

$$
P : r = (\Sigma i : 0 \leq i < j : b[i]) \land 0 \leq j \leq m + 1
$$

$$
t : m + 1 - j
$$

Develop the program using the method 1.

1. Find $B$ such that $P \land \neg B \Rightarrow R$. We note that the first conjunct of $P$ is equal to $R$ if the new variable has the value of the removed constant, i.e., let $\neg BB$ be the equality $j = m + 1$. Therefore, $\neg B = (j = m + 1)$, i.e., $B = (j \neq m + 1)$

2. Find a statement $S_t$ that decreases $t$. $j := j + 1$

3. Find a statement $S_{inv}$ such that $P$ is preserved.

$$
P' = wp("S_t", P)
$$

$$
= wp("j := j + 1", P)
$$

$$
= r = (\Sigma i : 0 \leq i < j + 1 : b[i]) \land 0 \leq j + 1 \leq m + 1
$$

Compare $P'$ with $P \land B$ to find $S_{inv}$ such that $P \land B \Rightarrow wp("S_{inv}", P')$. $P$ and $P'$ differ in the value of $r$. This is fixed by adding $b[j]$ to $r$. Therefore, $S_{inv} = r := r + b[j]$ will do.

4. Find a statement $S_{init}$ such that $P$ holds initially. $r, j := 0, 0$

The program:

$$
r, j := 0, 0
\begin{array}{c}
do \quad j \neq m + 1 \rightarrow r := r + b[j]; j := j + 1 \\
d\end{array}
$$
Enlarge the range of a variable

Example 6 Linear Search

Given a non-empty array \( b[0 : m] \), find the first value \( x \) in \( b[0 : m] \), the value is known to exist.

Let \( ix \) be the lowest index such that \( b[ix] = x \).

Precondition \( Q : 0 \leq m \land x \in b[0 : m] \)

Postcondition \( R : i = ix \)

Find \( P \) and \( t \) and develop a program that satisfies the specification.

Solution: Enlarge the range of \( i \) in \( R \).

\[
\begin{align*}
P & : 0 \leq i \leq ix \\
t & : m - i
\end{align*}
\]

Develop the program using the method 1.

1. Find \( B \) such that \( P \land \neg B \Rightarrow R \). Take \( B = b[i] \neq x \).
2. Find a statement \( S_t \) that decreases \( t \). \( i := i + 1 \).
3. Find a statement \( S_{inv} \) such that \( P \) is preserved.

\[
\begin{align*}
P' & = wp("S_t", P) \\
& = wp("i := i + 1", P) \\
& = 0 \leq i + 1 \leq ix
\end{align*}
\]

Compare \( P' \) with \( P \land B \) to find \( S_{inv} \) such that \( P \land B \Rightarrow wp("S_{inv}", P') \).

Since, \( P \land B \Rightarrow P'(\text{Homework!}) \), \( S_{inv} = \text{skip} \) will do.

4. Find a statement \( S_{init} \) such that \( P \) holds initially. \( i := 0 \)

The program:

\[
i := 0 \\
do \\
b[i] \neq x \rightarrow i := i + 1 \\
\od
\]

Example 7 Dutch National Flag

Problem

We are given a row of buckets, each containing a pebble. The pebble is either red, white or blue. There are two robot arms, each supplied with a camera that can decide the color of the pebble. Using the robot arms we can swap the pebbles in buckets \( i \) and \( j \). Sort the pebbles according to the colours of the dutch flag. Formally

We represent the buckets as an array \( A[0 : n - 1] \), where \( n \) is the number of the buckets. Each array entry has one of the values
1. red
2. white
3. blue

\forall i : 0 \leq i < n : A[i] = red \lor A[i] = white \lor A[i] = blue

**Postcondition R**

\[
\begin{align*}
A[0 : r - 1] &= \text{red} \\
\land \\
A[r : w - 1] &= \text{white} \\
\land \\
A[w : n - 1] &= \text{blue} \\
\land \\
0 \leq r \leq w \leq n
\end{align*}
\]

**Solution:**
Problem can be solved by do-loops using Method 2. Find an invariant and a bound function.

**Invariant \( P \):**

\[
\begin{align*}
A[0 : r - 1] &= \text{red} \\
\land \\
A[m : w - 1] &= \text{white} \\
\land \\
A[w : n - 1] &= \text{blue} \\
\land \\
0 \leq r \leq m \leq w \leq n
\end{align*}
\]
Bound function $t$:
We want to decrease the "mixed" section. This gives us the bound function $t := m - r$.

We see that when $m = r$, $P \Rightarrow R$ by simple substitution. We can therefore use $m \neq r$ as a part of all guards.

Sketch:

First Iteration:
In the "mixed" region, $A[m-1]$ can be white, red or blue.

- $A[m-1] = \text{white}$ as a conjunct with $m \neq r$ in the guard. With guard $A[m-1] = \text{white} \land r \neq m$, the statement reducing $t$ is just $m := m - 1$.
  This preserves the invariant.

Therefore, $P \land BB \Rightarrow wp(\"m := m - 1\", P)$
But $P \land \neg BB \Rightarrow R$? NO!

Second Iteration:

- $A[m-1] = \text{red}$ as a conjunct with $m \neq r$ in the guard. With guard $A[m-1] = \text{red} \land r \neq m$, the statement reducing $t$ is just $r := r + 1$. Also, if $A[m-1] = \text{red}$, we need to interchange $A[r]$ with $A[m-1]$ to satisfy the invariant.

By the invariant, all pebbles in $A[0:r-1]$ are red. We can have the $S_{\text{inv}}$ as $A[r], A[m-1] = A[m-1], A[r]$.
But $P \land \neg BB \Rightarrow R$? NO!

Third Iteration:

- $A[m-1] = \text{blue}$ as a conjunct with $m \neq r$ in the guard. With guard $A[m-1] = \text{blue} \land r \neq m$, the statement reducing $t$ is $m := m - 1; w := w - 1$.
  Also, if $A[m-1] = \text{blue}$, we need to interchange $A[r]$ with $A[m-1]$ to satisfy the invariant.

But $P \land \neg BB \Rightarrow R$? YES!

Find a statement $S_{\text{init}}$ such that $P$ holds initially. $m := n, r := 0, w := n$.

Program:

\[
m, r, w := n, 0, n
\]
\[
\begin{align*}
\text{do} & \\
& r \neq m \land A[m-1] = \text{white} \rightarrow m := m - 1 \\
& r \neq m \land A[m-1] = \text{red} \rightarrow A[r], A[m-1], r := A[m-1], A[r], r + 1 \\
& r \neq m \land A[m-1] = \text{blue} \rightarrow A[m-1], A[w-1], m, w := A[w-1], A[m-1], m - 1, w - 1
\end{align*}
\]
\[
\text{od}
\]
Example 8 Input: $x,y$ : integers s.t $0 \leq x$ and $0 < y$

**Output:** Quotient $q$ and remainder $r$ from dividing $x$ by $y$.

$q = x/y$

$r = x \% y$

Precondition $Q : 0 \leq x \land 0 < y$

Postcondition $R : 0 \leq r \land r < y \land q \cdot y + r = x$

Develop invariant : Method 1 - Delete a conjunct

**Invariant is**

$P : 0 \leq r \land q \cdot y + r = x$

**Bound function $t$:** $r - q \cdot y$

1. Find $B$ such that $P \land \neg B \Rightarrow R$. We note that $P \land \neg B \Rightarrow R$ is true if $P \land \neg B = R$, i.e $\neg BB$ is the removed conjunct. Use $BB$, the negation of the removed conjunct as guard. Therefore, $\neg B = (r < y)$, i.e, $B = r \geq y$

2. Find a statement $S_t$ that decreases $t$. $q := q + 1$

3. Find a statement $S_{inv}$ such that $P$ is preserved.

$$P' = wp("S_t", P)$$

$$= wp("q := q + 1", 0 \leq r \land q \cdot y + r = x)$$

$$= 0 \leq r \land (q + 1) \cdot y + r = x$$

Compare $P'$ with $P \land B$ to find $S_{inv}$ such that $P \land B \Rightarrow wp("S_{inv}", P')$.

To get $q \cdot y + r = x$, we need $r := r - y$.

4. Find a statement $S_{init}$ such that $P$ holds initially. $r, q := x, 0$

The program:

$$i := 0$$

$$\text{do}$$

$$\quad r \geq y \rightarrow r := r - y; q := q + 1$$

$$\text{od}$$

Redo using $t : r$

Precondition $Q : 0 \leq x \land 0 < y$

Postcondition $R : 0 \leq r \land r < y \land q \cdot y + r = x$

Develop invariant : Method 1 - Delete a conjunct

**Invariant is**

$P : 0 \leq r \land q \cdot y + r = x$

**Bound function $t$:** $r$

1. Find $B$ such that $P \land \neg B \Rightarrow R$. We note that $P \land \neg B \Rightarrow R$ is true if $P \land \neg B = R$, i.e $\neg BB$ is the removed conjunct. Use $BB$, the negation of the removed conjunct as guard. Therefore, $\neg B = (r < y)$, i.e, $B = r \geq y$
2. Find a statement \( S_t \) that decreases \( t. \ r := r - y \)

3. Find a statement \( S_{inv} \) such that \( P \) is preserved.
   \[
   P' = \text{wp}("S_t", P) \\
   = \text{wp}("r := r - y", 0 \leq r \land q \cdot y + r = x) \\
   = 0 \leq r \land q \cdot y + r - y = x
   \]

   Compare \( P' \) with \( P \land B \) to find \( S_{inv} \) such that \( P \land B \Rightarrow \text{wp}("S_{inv}", P'). \)

   To get \( q \cdot y + r = x \), we need \( q := q - 1 \).

4. Find a statement \( S_{init} \) such that \( P \) holds initially. \( r, q := x, 0 \)

The program:

\[
\begin{align*}
i & := 0 \\
\text{do} \\
\quad r & \geq y \Rightarrow q := q - 1; r := r - y \\
\text{od}
\end{align*}
\]

**Example 9 Sorting**

Precondition \( Q : n \geq 0 \)

Postcondition \( R : \text{sorted}(a[0 : n]) \)

Develop invariant : Method 2 - Replace a constant with a variable

**Invariant** is

Two possibilities : \( a)\text{sorted}(A[0 : i]) \land 0 \leq i \leq n \quad b)\text{sorted}(A[i : n]) \land 0 \leq i \leq n \)

Bound function \( t: a)n - i \quad b)i \)

1. Find \( B \) such that \( P \land \neg B \Rightarrow R. \ a)\neg B = (i = n), b)\neg B = (i = 0) \)
   
   Therefore, \( a)B = (i \neq n) \quad b)B = (i \neq 0) \)

2. Find a statement \( S_t \) that decreases \( t. \ a)i := i + 1 \quad b)i := i - 1 \)

3. Find a statement \( S_{inv} \) such that \( P \) is preserved using \( S_t = a)\).
   \[
   \begin{align*}
P' &= \text{wp}("S_t", P) \\
   &= \text{wp}("i := i + 1", \text{sorted}(A[0 : i]) \land 0 \leq i \leq n) \\
   &= \text{sorted}(A[0 : i + 1]) \land 0 \leq i + 1 \leq n
   \end{align*}
   \]

   Compare \( P' \) with \( P \land B \) to find \( S_{inv} \) such that \( P \land B \Rightarrow \text{wp}("S_{inv}", P'). \)

   sorted \( A[0 : i + 1] \) is not established by \( P \land B \). It would be established if \( A[0 : i] \leq A[i + 1] \). Since we want to repeat this, we want \( A[i + 1] \) to be the minimum of \( A[i + 1 : n] \), i.e \( A[i + 1] \leq A[i + 1 : n] \).

   Add \( A[0 : i] \leq A[i + 1 : n] \) to the invariant.

   We want \( S_{inv} \) to find the minimum of \( A[i + 1 : n] \) and swap it into position \( A[i + 1] \). We need a loop to do that.

Find minimum :
(a) PostCondition \( R_{min} = P' = 0 \leq i + 1 \leq n \wedge \text{sorted}(A[0:i+1]) \wedge A[0 : i + 1] \leq A[i + 2 : n] \)

Since \( A[0 : i] \) is out of range for this min search, predicated dealing with this part can be ignored. \( R_{min} = 0 \leq i + 1 \leq n \wedge A[i + 1 : n] \leq A[i + 2 : n] \)

(b) Invariant : replace constant with a variable
\[ P_{min} : 0 \leq i + 1 \leq j \leq n \wedge A[i + 1] \leq A[i + 1 : j] \]
\[ t_{min} : n - j \]

(c) Using method 2,

i. Find statement that decreases the bound here. \( j := j + 1 \)

ii. Guard that preserves \( P \)
\[ A[i + 1] \]

iii. \( \neg BB = (j = n) \). Does \( P \wedge \neg BB \Rightarrow R \) ? NO!

iv. Go to step 1. Another statement which decreases the bound can be: \( j, A[i + 1], A[j + 1] := j + 1, A[j + 1], A[i + 1] \)

v. Guard for this: \( j \neq n \wedge A[i + 1] \geq A[j + 1] \)

vi. Does \( P \wedge \neg BB \Rightarrow R \) ? YES!

(d) \( S_{init} = (j := i + 1) \)

The program \( FIND_{MIN} \):

\[
j := i + 1
\]

\[
do
\]

\[
\begin{align*}
& j \neq n \wedge A[i + 1] \leq A[j + 1] \rightarrow j := j + 1 \\
& \text{od}
\end{align*}
\]

4. Find a statement \( S_{inv} \) such that \( P \) is preserved using \( S_t = b \).

\[
P' = \text{wp}("S_{inv}", P) = \text{wp}("i := i - 1", \text{sorted}(A[0 : i]) \wedge 0 \leq i \leq n) \wedge \text{sorted}(A[0 : i - 1]) \wedge 0 \leq i - 1 \leq n)
\]

Compare \( P' \) with \( P \wedge B \) to find \( S_{inv} \) such that \( P \wedge B \Rightarrow \text{wp}("S_{inv}", P') \).

sorted \( A[0 : i - 1] \) is not established by \( P \wedge B \). We need to know \( A[i - 1] \leq A[i + 1 : n] \) to be able to repeat \( A[0 : i - 1] \leq A[i + 1 : i - 1] \).

Add \( A[0 : i - 1] \leq A[i - 1 : n] \leq A[i : n] \) to the invariant.

We want \( S_{inv} \) to find the maximum of \( A[0 : i - 1] \) and swap it into position \( A[i - 1] \). We need a loop to do that: We can do it similarly as we found minimum (Home work!)

5. Find a statement \( S_{init} \) such that \( P \) holds initially. \( i := -1 \)

The program:

\[
\{ Q : n \geq 0 \} i := -1 \\
\{ P : \text{sorted}(A[0 : i]) \wedge -1 \leq i \leq n \wedge A[0 : i] \leq A[i + 1 : n] \} \\
\{ t : n - i \} 
\]
\[
\text{do}
\]
\[
i \neq n \rightarrow j := i + 1
\]
\[
\text{do}
\]
\[
\begin{align*}
&j \neq n \land A[i + 1] \leq A[j + 1] \rightarrow j := j + 1 \\
&j \neq n \land A[i + 1] \geq A[j + 1] \rightarrow \\
\end{align*}
\]
\[
\text{do}
\]
\[
i := i + 1
\]
\[
\{ \text{R : sorted}(A[0 : n]) \}\]