CS156: The Calculus of Computation

Zohar Manna

Autumn 2008

Chapter 5: Program Correctness: Mechanics

Program A: LinearSearch with function specification

```
@pre $0 \le \ell \wedge u < |a|$
@post $rv \leftrightarrow \exists i.\ \ell \le i \le u \wedge a[i] = e$
bool LinearSearch(int[] $a$, int $\ell$, int $u$, int $e$) {
  for @ $\top$
    (int $i := \ell$; $i \le u$; $i := i+1$) {
    if ($a[i] = e$) return true;
  }
  return false;
}
```

Function LinearSearch searches subarray of array a of integers for specified value e.

Function specifications

- Function precondition (@*pre*)
  It behaves correctly only if $0 \le \ell$ and $u < |a|$
- Function postcondition (@*post*)
  It returns true iff a contains the value e in the range $[\ell, u]$

for loop: initially set i to be ℓ,
execute the body and increment i by 1
as long as $i \le u$

@ - program annotation

Program B: BinarySearch with function specification

```
@pre $0 \le \ell \wedge u < |a| \wedge \text{sorted}(a, \ell, u)$
@post $rv \leftrightarrow \exists i.\ \ell \le i \le u \wedge a[i] = e$
bool BinarySearch(int[] $a$, int $\ell$, int $u$, int $e$) {
  if ($\ell > u$) return false;
  else {
    int $m := (\ell + u)$ div 2;
    if ($a[m] = e$) return true;
    else if ($a[m] < e$) return BinarySearch($a, m+1, u, e$);
    else return BinarySearch($a, \ell, m-1, e$);
  }
}
```

The recursive function BinarySearch searches sorted subarray a of integers for specified value e.

sorted: weakly increasing order, i.e.

$$\text{sorted}(a, \ell, u) \;\Leftrightarrow\; \forall i, j.\ \ell \leq i \leq j \leq u \;\rightarrow\; a[i] \leq a[j]$$

Defined in the combined theory of integers and arrays, $T_{\mathbb{Z} \cup A}$

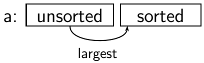
Function specifications

- Function precondition (@*pre*)
  It behaves correctly only if
  $0 \leq \ell$ and $u < |a|$ and
  $\text{sorted}(a, \ell, u)$.
- Function postcondition (@*post*)
  It returns true iff a contains the value e in the range $[\ell, u]$

Program C: BubbleSort with function specification

```
@pre ⊤
@post sorted(rv, 0, |rv| − 1)
int[] BubbleSort(int[] a₀) {
  int[] a := a₀;
  for @ ⊤
    (int i := |a| − 1; i > 0; i := i − 1) {
    for @ ⊤
      (int j := 0; j < i; j := j + 1) {
      if (a[j] > a[j + 1]) {
        int t := a[j];
        a[j] := a[j + 1];
        a[j + 1] := t;
      }
    }
  }
  return a;
}
```

Function BubbleSort sorts integer array a



by "bubbling" the largest element of the left unsorted region of a toward the sorted region on the right.

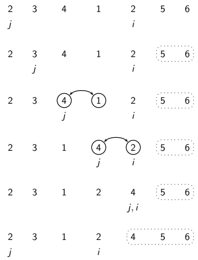
Each iteration of the outer loop expands the sorted region by one cell.[1]

Function specification

- Function postcondition (@post):
  BubbleSort returns array rv sorted on the range $[0, |rv| - 1]$.

[1] Except the last iteration, which expands the sorted region by two cells, so that an entire array of length n is sorted in $n - 1$ iterations.

Sample execution of BubbleSort



Program Annotation

- Function Specifications
  function precondition (@pre)
  function postcondition (@post)
- Runtime Assertions
  e.g., $@\ 0 \le j < |a| \ \wedge\ 0 \le j+1 < |a|$
  $a[j] := a[j+1]$
- Loop Invariants
  e.g., $@\ L : \ell \le i \ \wedge\ \forall j.\ \ell \le j < i \ \rightarrow\ a[j] \ne e$

  The L : gives a name to the formula, just like the F : we've used in other formulae.

Program A: LinearSearch with runtime assertions

```
@pre $0 \le \ell \ \wedge\ u < |a|$
@post $rv \ \leftrightarrow\ \exists i.\ \ell \le i \le u \wedge a[i] = e$
bool LinearSearch(int[] $a$, int $\ell$, int $u$, int $e$) {
  for
    @ $L : \top$
    (int $i := \ell$; $i \le u$; $i := i+1$) {
    @ $0 \le i < |a|$;
    if ($a[i] = e$) return true;
  }
  return false;
}
```

Program B: BinarySearch with runtime assertions

```
@pre $0 \le \ell \ \wedge\ u < |a| \ \wedge\ \mathrm{sorted}(a, \ell, u)$
@post $rv \ \leftrightarrow\ \exists i.\ \ell \le i \le u \wedge a[i] = e$
bool BinarySearch(int[] $a$, int $\ell$, int $u$, int $e$) {
  if ($\ell > u$) return false;
  else {
    @ $2 \ne 0$;
    int $m := (\ell + u)$ div 2;
    @ $0 \le m < |a|$;
    if ($a[m] = e$) return true;
    else {
      @ $0 \le m < |a|$;
      if ($a[m] < e$) return BinarySearch($a, m+1, u, e$);
      else return BinarySearch($a, \ell, m-1, e$);
    }
  }
}
```

Program C: BubbleSort with runtime assertions

```
@pre $\top$
@post $\mathrm{sorted}(rv, 0, |rv| - 1)$
int[] BubbleSort(int[] $a_0$) {
  int[] $a := a_0$;
  for
    @ $L_1 : \top$
    (int $i := |a| - 1$; $i > 0$; $i := i - 1$) {
    for
      @ $L_2 : \top$
      (int $j := 0$; $j < i$; $j := j + 1$) {
      @ $0 \le j < |a| \wedge 0 \le j+1 < |a|$;
      if ($a[j] > a[j+1]$)  {
        int $t := a[j]$;
        $a[j] := a[j+1]$;
        $a[j+1] := t$;
      }
    }
  }
  return $a$;
}
```

Loop Invariants

```
while
  @ $F$
  $\langle cond \rangle$ { $\langle body \rangle$ }
```

- apply $\langle body \rangle$ as long as $\langle cond \rangle$ holds
- assertion F holds at the beginning of every iteration evaluated before $\langle cond \rangle$ is checked

for @ F $(\langle init \rangle; \langle cond \rangle; \langle incr \rangle)$ $\{\langle body \rangle\}$	$\langle init \rangle$; while @ F $\langle cond \rangle$ { $\langle body \rangle$; $\langle incr \rangle$ }

Program A: LinearSearch with loop invariants

```
@pre $0 \leq \ell \wedge u < |a|$
@post $rv \;\leftrightarrow\; \exists j.\; \ell \leq j \leq u \wedge a[j] = e$
bool LinearSearch(int[] $a$, int $\ell$, int $u$, int $e$) {
  for
    @$L$ : $\ell \leq i \wedge (\forall j.\; \ell \leq j < i \;\rightarrow\; a[j] \neq e)$
    (int $i := \ell$; $i \leq u$; $i := i + 1$) {
    if $(a[i] = e)$ return true;
  }
  return false;
}
```

Proving Partial Correctness

A function is partially correct if
when the program's precondition is satisfied on entry,
its postcondition is satisfied when the program halts/exits.

- A program + annotation is reduced to finite set of verification conditions (VCs), FOL formulae
- If all VCs are T-valid, then the program obeys its specification (partially correct)

Basic Paths: Loops

To handle loops, we break the program into basic paths

@ $\leftarrow$ precondition or loop invariant

sequence of instructions
(with no loop invariants)

@ $\leftarrow$ loop invariant, runtime assertion, or postcondition

Program A: LinearSearch I

Basic Paths of LinearSearch

(1)

```
@pre $0 \leq \ell \wedge u < |a|$
$i := \ell$;
@$L$ : $\ell \leq i \wedge \forall j.\ \ell \leq j < i \ \rightarrow\ a[j] \neq e$
```

(2)

```
@$L$ : $\ell \leq i \wedge \forall j.\ \ell \leq j < i \ \rightarrow\ a[j] \neq e$
assume $i \leq u$;
assume $a[i] = e$;
$rv$ := true;
@post $rv \ \leftrightarrow\ \exists j.\ \ell \leq j \leq u \wedge a[j] = e$
```

Program A: LinearSearch II

(3)

```
@$L$ : $\ell \leq i \wedge \forall j.\ \ell \leq j < i \ \rightarrow\ a[j] \neq e$
assume $i \leq u$;
assume $a[i] \neq e$;
$i := i + 1$;
@$L$ : $\ell \leq i \wedge \forall j.\ \ell \leq j < i \ \rightarrow\ a[j] \neq e$
```

(4)

```
@$L$ : $\ell \leq i \wedge \forall j.\ \ell \leq j < i \ \rightarrow\ a[j] \neq e$
assume $i > u$;
$rv$ := false;
@post $rv \ \leftrightarrow\ \exists j.\ \ell \leq j \leq u \wedge a[j] = e$
```

Visualization of basic paths of LinearSearch

$$\begin{tikzcd}
\text{@pre} \arrow[d, "\textbf{(1)}"] \\
L \arrow[d, "\textbf{(2)},\textbf{(4)}"] \arrow[loop left, "\textbf{(3)}"] \\
\text{@post}
\end{tikzcd}$$

Program C: BubbleSort with loop invariants

```
@pre $|a_0| > 0$
@post sorted$(rv, 0, |rv| - 1)$
int[] BubbleSort(int[] $a_0$) {
  int[] $a := a_0$;
  for
```

$$@L_1 : \begin{bmatrix} 0 \leq i < |a| \\ \wedge \text{partitioned}(a, 0, i, i+1, |a|-1) \\ \wedge \text{sorted}(a, i, |a|-1) \end{bmatrix}$$

```
    (int $i := |a| - 1$; $i > 0$; $i := i - 1$) {
```

```
    for
        @$L_2$ : $\begin{bmatrix} 1 \le i < |a| \land 0 \le j \le i \\ \land \text{partitioned}(a, 0, i, i+1, |a|-1) \\ \land \text{partitioned}(a, 0, j-1, j, j) \\ \land \text{sorted}(a, i, |a|-1) \end{bmatrix}$
      (int $j := 0;\ j < i;\ j := j+1$) {
      if ($a[j] > a[j+1]$) {
        int $t := a[j]$;
        $a[j] := a[j+1]$;
        $a[j+1] := t$;
      }
    }
  }
  return $a$;
}
```

Partition

$$\text{partitioned}(a, \ell_1, u_1, \ell_2, u_2) \;\Leftrightarrow\; \forall i, j.\ \ell_1 \le i \le u_1 < \ell_2 \le j \le u_2 \;\rightarrow\; a[i] \le a[j]$$

in $T_{\mathbb{Z}} \cup T_{\mathsf{A}}$.

That is, each element of a in the range $[\ell_1, u_1]$ is $\le$ each element in the range $[\ell_2, u_2]$.

Basic Paths of BubbleSort

(1)

```
@pre $|a_0| > 0$
$a := a_0$;
$i := |a| - 1$;
@$L_1$ : $\begin{bmatrix} 0 \le i < |a| \land \text{partitioned}(a, 0, i, i+1, |a|-1) \\ \land\ \text{sorted}(a, i, |a|-1) \end{bmatrix}$
```

(2)

```
@$L_1$ : $\begin{bmatrix} 0 \le i < |a| \land \text{partitioned}(a, 0, i, i+1, |a|-1) \\ \land\ \text{sorted}(a, i, |a|-1) \end{bmatrix}$
assume $i > 0$;
$j := 0$;
@$L_2$ : $\begin{bmatrix} 1 \le i < |a| \land 0 \le j \le i \land \text{partitioned}(a, 0, i, i+1, |a|-1) \\ \land\ \text{partitioned}(a, 0, j-1, j, j) \land \text{sorted}(a, i, |a|-1) \end{bmatrix}$
```

(3)

```
@$L_2$ : $\begin{bmatrix} 1 \le i < |a| \land 0 \le j \le i \land \text{partitioned}(a, 0, i, i+1, |a|-1) \\ \land\ \text{partitioned}(a, 0, j-1, j, j) \land \text{sorted}(a, i, |a|-1) \end{bmatrix}$
assume $j < i$;
assume $a[j] > a[j+1]$;
$t := a[j]$;
$a[j] := a[j+1]$;
$a[j+1] := t$;
$j := j+1$;
@$L_2$ : $\begin{bmatrix} 1 \le i < |a| \land 0 \le j \le i \land \text{partitioned}(a, 0, i, i+1, |a|-1) \\ \land\ \text{partitioned}(a, 0, j-1, j, j) \land \text{sorted}(a, i, |a|-1) \end{bmatrix}$
```

(4)

$$@L_2 : \begin{bmatrix} 1 \leq i < |a| \land 0 \leq j \leq i \land \text{partitioned}(a, 0, i, i+1, |a|-1) \\ \land\ \text{partitioned}(a, 0, j-1, j, j) \land \text{sorted}(a, i, |a|-1) \end{bmatrix}$$

assume $j < i$;

assume $a[j] \leq a[j+1]$;

$j := j + 1$;

$$@L_2 : \begin{bmatrix} 1 \leq i < |a| \land 0 \leq j \leq i \land \text{partitioned}(a, 0, i, i+1, |a|-1) \\ \land\ \text{partitioned}(a, 0, j-1, j, j) \land \text{sorted}(a, i, |a|-1) \end{bmatrix}$$

(5)

$$@L_2 : \begin{bmatrix} 1 \leq i < |a| \land 0 \leq j \leq i \land \text{partitioned}(a, 0, i, i+1, |a|-1) \\ \land\ \text{partitioned}(a, 0, j-1, j, j) \land \text{sorted}(a, i, |a|-1) \end{bmatrix}$$

assume $j \geq i$;

$i := i - 1$;

$$@L_1 : \begin{bmatrix} 0 \leq i < |a| \land \text{partitioned}(a, 0, i, i+1, |a|-1) \\ \land\ \text{sorted}(a, i, |a|-1) \end{bmatrix}$$

(6)

$$@L_1 : \begin{bmatrix} 0 \leq i < |a| \land \text{partitioned}(a, 0, i, i+1, |a|-1) \\ \land\ \text{sorted}(a, i, |a|-1) \end{bmatrix}$$

assume $i \leq 0$;

$rv := a$;

@post $\text{sorted}(rv, 0, |rv|-1)$

Visualization of basic paths of BubbleSort

$$\begin{tikzcd}
 & @\text{pre} \arrow[d, "\textbf{(1)}"] & \\
 & L_1 \arrow[dl, bend right, "\textbf{(6)}"'] \arrow[dr, bend left, "\textbf{(2)}"] & \\
@\text{post} & & L_2 \arrow[ul, bend left, "\textbf{(5)}"] \arrow[loop below, "\textbf{(3)},\textbf{(4)}"]
\end{tikzcd}$$

Basic Paths: Function Calls

- Loops produce unbounded number of paths
  loop invariants cut loops to produce
  finite number of basic paths
- Reursive calls produce unbounded number of paths
  function specifications cut function calls

In BinarySearch

@pre $0 \leq \ell \land u < |a| \land \text{sorted}(a, \ell, u)$ $\quad \ldots F[a, \ell, u, e]$

$\vdots$

$@R_1 :\ 0 \leq m+1 \land u < |a| \land \text{sorted}(a, m+1, u)$ $\quad \ldots F[a, m+1, u, e]$

return BinarySearch$(a, m+1, u, e)$

$\vdots$

$@R_2 :\ 0 \leq \ell \land m-1 < |a| \land \text{sorted}(a, \ell, m-1)$ $\quad \ldots F[a, \ell, m-1, e]$

return BinarySearch$(a, \ell, m-1, e)$

Program B: BinarySearch with function call assertions

```
@pre $0 \le \ell \wedge u < |a| \wedge \text{sorted}(a, \ell, u)$
@post $rv \leftrightarrow \exists i.\ \ell \le i \le u \wedge a[i] = e$
bool BinarySearch(int[] $a$, int $\ell$, int $u$, int $e$) {
  if ($\ell > u$) return false;
  else {
    int $m := (\ell + u)$ div 2;
    if ($a[m] = e$) return true;
    else if ($a[m] < e$) {
      @$R_1$ : $0 \le m + 1 \wedge u < |a| \wedge \text{sorted}(a, m + 1, u)$;
      return BinarySearch($a, m + 1, u, e$);
    } else {
      @$R_2$ : $0 \le \ell \wedge m - 1 < |a| \wedge \text{sorted}(a, \ell, m - 1)$;
      return BinarySearch($a, \ell, m - 1, e$);
    }
  }
}
```

(1)

```
@pre $0 \le \ell \wedge u < |a| \wedge \text{sorted}(a, \ell, u)$
assume $\ell > u$;
$rv :=$ false;
@post $rv \leftrightarrow \exists i.\ \ell \le i \le u \wedge a[i] = e$
```

(2)

```
@pre $0 \le \ell \wedge u < |a| \wedge \text{sorted}(a, \ell, u)$
assume $\ell \le u$;
$m := (\ell + u)$ div 2;
assume $a[m] = e$;
$rv :=$ true;
@post $rv \leftrightarrow \exists i.\ \ell \le i \le u \wedge a[i] = e$
```

(3)

```
@pre $0 \le \ell \wedge u < |a| \wedge \text{sorted}(a, \ell, u)$
assume $\ell \le u$;
$m := (\ell + u)$ div 2;
assume $a[m] \ne e$;
assume $a[m] < e$;
@$R_1 : 0 \le m + 1 \wedge u < |a| \wedge \text{sorted}(a, m + 1, u)$
```

(5)

```
@pre $0 \le \ell \wedge u < |a| \wedge \text{sorted}(a, \ell, u)$
assume $\ell \le u$;
$m := (\ell + u)$ div 2;
assume $a[m] \ne e$;
assume $a[m] \ge e$;
@$R_2 : 0 \le \ell \wedge m - 1 < |a| \wedge \text{sorted}(a, \ell, m - 1)$
```

(4)

```
@pre $0 \le \ell \wedge u < |a| \wedge \text{sorted}(a, \ell, u)$
assume $\ell \le u$;
$m := (\ell + u)$ div 2;
assume $a[m] \ne e$;
assume $a[m] < e$;
assume $v_1 \leftrightarrow \exists i.\ m + 1 \le i \le u \wedge a[i] = e$;
$rv := v_1$;
@post $rv \leftrightarrow \exists i.\ \ell \le i \le u \wedge a[i] = e$
```

(6)

```
@pre $0 \le \ell \wedge u < |a| \wedge \text{sorted}(a, \ell, u)$
assume $\ell \le u$;
$m := (\ell + u)$ div 2;
assume $a[m] \ne e$;
assume $a[m] \ge e$;
assume $v_2 \leftrightarrow \exists i.\ \ell \le i \le m - 1 \wedge a[i] = e$;
$rv := v_2$;
@post $rv \leftrightarrow \exists i.\ \ell \le i \le u \wedge a[i] = e$
```

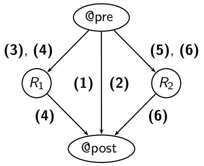


Figure: Visualization of basic paths of BinarySearch

Program States

Program counter pc holds current location of control

State s of P assignment of values to all variables (proper types) of P

Example:

$$s : \left\{ \begin{array}{l} pc \mapsto L_2,\ a \mapsto [0;1;2], \\ i \mapsto 3,\ j \mapsto 0 \end{array} \right\}$$

is a state of BubbleSort.

Reachable state s of P a state that can be reached during some computation of P

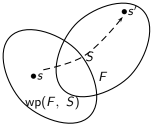
Example:

$$s : \left\{ \begin{array}{l} pc \mapsto L_2,\ a \mapsto [0;1;2], \\ i \mapsto 2,\ j \mapsto 0 \end{array} \right\}$$

is a reachable state of BubbleSort.

Weakest Precondition $\text{wp}(F, S)$

For FOL formula F, program statement S,
$s \models \text{wp}(F,\ S)$ iff
statement S is executed on state s to produce state s',
and $s' \models F$:



- $\text{wp}(F,\ \texttt{assume}\ c) \Leftrightarrow c \rightarrow F$
- $\text{wp}(F[v],\ v := e) \Leftrightarrow F[e]$
- For $S_1; \dots; S_n$,
  $\text{wp}(F,\ S_1; \dots; S_n) \Leftrightarrow \text{wp}(\text{wp}(F,\ S_n),\ S_1; \dots; S_{n-1})$

Verification Conditions

Verification Condition of basic path

$@\ F$
$S_1;$
$\dots$
$S_n;$
$@\ G$

is

$F \rightarrow \text{wp}(G,\ S_1; \dots; S_n)$

Also denoted by

$\{F\} S_1; \dots; S_n \{G\}$

That is, for every state s,

if $s \models F$
then $s' \models G$ (after the path $S_1; S_2; \dots; S_n$ is executed)

Example: Basic path

(1)

$@\ F:\ x \geq 0$
$S_1:\ x := x + 1;$
$@\ G:\ x \geq 1$

The VC is $F \rightarrow \text{wp}(G,\ S_1)$. That is,

$$\begin{aligned} & \text{wp}(G,\ S_1) \\ \Leftrightarrow\ & \text{wp}(x \geq 1,\ x := x+1) \\ \Leftrightarrow\ & (x \geq 1)\{x \mapsto x+1\} \\ \Leftrightarrow\ & x + 1 \geq 1 \\ \Leftrightarrow\ & x \geq 0 \end{aligned}$$

Therefore the VC of path (1) is

$$x \geq 0 \rightarrow x \geq 0\ ,$$

which is $T_{\mathbb{Z}}$-valid.

Example 1: Shortcut (backward substitution)

VC: $\boxed{\underbrace{x \geq 0}_{F} \rightarrow \underbrace{x \geq 0}_{\text{wp}(G, S_1)}}$

$@F:\ x \geq 0$
$x + 1 \geq 1$ *i.e.* $x \geq 0$

$S_1:\ x := x + 1;$
$x \geq 1$

$@G:\ x \geq 1$

$\Uparrow$

Example: Basic path (2) of LinearSearch

(2)

$$
\begin{array}{l}
@L:\ F:\ \ell \le i \land \forall j.\ \ell \le j < i \ \rightarrow\ a[j] \ne e \\
S_1:\ \texttt{assume}\ i \le u; \\
S_2:\ \texttt{assume}\ a[i] = e; \\
S_3:\ rv := \texttt{true}; \\
@\text{post}\ G:\ rv\ \leftrightarrow\ \exists j.\ \ell \le j \le u \land a[j] = e
\end{array}
$$

The VC is $F\ \rightarrow\ \text{wp}(G,\ S_1; S_2; S_3)$. That is,

$$
\begin{array}{ll}
 & \text{wp}(G,\ S_1; S_2; S_3) \\
\Leftrightarrow & \text{wp}(\text{wp}(rv\ \leftrightarrow\ \exists j.\ \ell \le j \le u \land a[j] = e,\ rv := \texttt{true}),\ S_1; S_2) \\
\Leftrightarrow & \text{wp}(\texttt{true}\ \leftrightarrow\ \exists j.\ \ell \le j \le u \land a[j] = e,\ S_1; S_2) \\
\Leftrightarrow & \text{wp}(\exists j.\ \ell \le j \le u \land a[j] = e,\ S_1; S_2) \\
\Leftrightarrow & \text{wp}(\text{wp}(\exists j.\ \ell \le j \le u \land a[j] = e,\ \texttt{assume}\ a[i] = e),\ S_1) \\
\Leftrightarrow & \text{wp}(a[i] = e\ \rightarrow\ \exists j.\ \ell \le j \le u \land a[j] = e,\ S_1) \\
\Leftrightarrow & \text{wp}(a[i] = e\ \rightarrow\ \exists j.\ \ell \le j \le u \land a[j] = e,\ \texttt{assume}\ i \le u) \\
\Leftrightarrow & i \le u\ \rightarrow\ (a[i] = e\ \rightarrow\ \exists j.\ \ell \le j \le u \land a[j] = e)
\end{array}
$$

Therefore the VC of path (2) is

$$
\begin{array}{l}
\ell \le i\ \land\ (\forall j.\ \ell \le j < i\ \rightarrow\ a[j] \ne e) \\
\quad \rightarrow\ (i \le u\ \rightarrow\ (a[i] = e\ \rightarrow\ \exists j.\ \ell \le j \le u \land a[j] = e))
\end{array} \qquad (1)
$$

or, equivalently,

$$
\begin{array}{l}
\ell \le i\ \land\ (\forall j.\ \ell \le j < i\ \rightarrow\ a[j] \ne e)\ \land\ i \le u\ \land\ a[i] = e \\
\quad \rightarrow\ \exists j.\ \ell \le j \le u\ \land\ a[j] = e
\end{array} \qquad (2)
$$

according to the equivalence

$$
\begin{array}{ll}
 & F_1 \land F_2\ \rightarrow\ (F_3\ \rightarrow\ (F_4\ \rightarrow\ F_5)) \\
\Leftrightarrow & (F_1 \land F_2 \land F_3 \land F_4)\ \rightarrow\ F_5\ .
\end{array}
$$

This formula (2) is $(T_{\mathbb{Z}} \cup T_{\mathsf{A}})$-valid.

Example 2: Shortcut (backward substitution)

VC: $\boxed{\underbrace{1 \le i \land (\forall j.A[j])}_{F}\ \land i \le u\ \land\ a[i] = e \rightarrow (\exists j.B[j])}$

$@L:\ F:\ 1 \le i\ \land\ \forall j.\underbrace{1 \le j < i \rightarrow a[j] \ne e}_{A[j]}$

$i \le u\ \land\ a[i] = e \rightarrow (\exists j.B[j])$

$S_1:$ assume $i \le u$;

$a[i] = e \rightarrow (\exists j.B[j])$

⇑

Example 2: Shortcut (backward substitution), cont.

$S_1:$ assume $i \le u$;

$a[i] = e \rightarrow (\exists j.B[j])$

$S_2:$ assume $a[i] = e$;

true $\leftrightarrow (\exists j.B[j])$ $\quad$ *i.e.* $\quad (\exists j.B[j]))$

$S_3:$ $rv :=$ true;

$rv \leftrightarrow (\exists j.B[j])$

$@\text{post}\ G:\ rv \leftrightarrow \exists j.\underbrace{1 \le j \le u\ \land\ a[j] = e}_{B[j]}$

⇑

P-invariant and P-inductive I

Consider program P with function f s.t.

function precondition F_{pre} and

initial location L_0.

A P-computation is a sequence of states

$s_0, s_1, s_2, \ldots$

such that

- $s_0[pc] = L_0$ and $s_0 \models F_{\text{pre}}$, and
- for each i, s_{i+1} is the result of executing the instruction at $s_i[pc]$ on state s_i.

where $s_i[pc]$ = value of pc given by state s_i.

P-invariant and P-inductive II

A formula F annotating location L of program P is P-invariant if for all P-computations $s_0, s_1, s_2, \ldots$ and for each index i,

$$s_i[pc] = L \quad \Rightarrow \quad s_i \models F$$

Annotations of P are P-invariant iff each annotation of P is P-invariant at its location.

Not Implementable: checking if F is P-invariant requires an infinite number of P-computations in general.

Annotations of P are P-inductive iff all VCs generated from the basic paths of program P are T-valid

$$P\text{-inductive} \Rightarrow P\text{-invariant}$$

In Practice: we check if the annotations are P-inductive.

Theorem (Verification Conditions)

If for every basic path

```
@$L_1$ : $F$
$S_1$;
⋮
$S_n$;
@$L_j$ : $G$
```

of program P, the verification condition

$$\{F\}S_1; \ldots ; S_n\{G\}$$

is T-valid, then the annotations are P-inductive, and therefore P-invariant.

Partial Correctness: For program P, if there is a P-invariant annotation, then P is partially correct.

Total Correctness

Total Correctness = Partial Correctness + Termination

For every input that satisfies F_{pre}, the program eventually halts and produces output that satisfies F_{post}.

Proving function termination:

- Choose set W with well-founded relation $\prec$
  Usually set of n-tuples of natural numbers with the lexicographic relation $<_n$
- Find function δ (ranking function) mapping
  program states $\rightarrow$ W
  such that δ decreases according to $\prec$ along every basic path.

Since $\prec$ is well-founded, there cannot exist an infinite sequence of program states. The program must terminate.

Showing decrease of ranking function

For basic path with ranking function

```
@ $F$
↓ $\delta[\overline{x}]$      ... ranking function
$S_1$;
⋮
$S_k$;
↓ $\kappa[\overline{x}]$      ... ranking function
```

We must prove that
the value of $\kappa \in W$ after executing $S_1; \cdots ; S_n$
is less than
the value of $\delta \in W$ before executing the statements
Thus, we show the verification condition

$$F \rightarrow \text{wp}(\kappa \prec \delta[\overline{x}_0],\ S_1; \cdots ; S_k)\{\overline{x}_0 \mapsto \overline{x}\}\ .$$

Example: BubbleSort — loops

Choose $(\mathbb{N}^2, <_2)$ as well-founded set

```
@pre $\top$
@post $\top$
int[] BubbleSort(int[] $a_0$) {
  int[] $a := a_0$;
  for
    @$L_1 :\ i + 1 \geq 0$
    ↓ $(i + 1,\ i + 1)$              ... ranking function $\delta_1$
    (int $i := |a| - 1;\ i > 0;\ i := i - 1$) {
```

```
      for
        @$L_2 :\ i + 1 \geq 0 \wedge i - j \geq 0$
        ↓ $(i + 1,\ i - j)$              ... ranking function $\delta_2$
        (int $j := 0;\ j < i;\ j := j + 1$) {
        if ($a[j] > a[j + 1]$) {
          int $t := a[j]$;
          $a[j] := a[j + 1]$;
          $a[j + 1] := t$;
        }
      }
    }
  return $a$;
}
```

We have to prove

- loop invariants are inductive (we don't show here)
- function decreases along each basic path.

The relevant basic paths:

(1)

```
@$L_1 :\ i + 1 \geq 0$
↓$L_1 :\ (i + 1, i + 1)$
assume $i > 0$;
$j := 0$;
↓$L_2 :\ (i + 1, i - j)$
```

Path **(1)**:

$$i + 1 \geq 0 \wedge i > 0 \rightarrow (i + 1, i - 0) <_2 (i + 1, i + 1)$$

(2, 3)

```
@$L_2$ :  $i+1 \geq 0 \wedge i-j \geq 0$
↓$L_2$ :  $(i+1, i-j)$
assume $j < i$;
...
$j := j+1$;
↓$L_2$ :  $(i+1, i-j)$
```

Paths **(2)** and **(3)**:

$$i+1 \geq 0 \wedge i-j \geq 0 \wedge j < i \rightarrow (i+1, i-(j+1)) <_2 (i+1, i-j)$$

(4)

```
@$L_2$ :  $i+1 \geq 0 \wedge i-j \geq 0$
↓$L_2$ :  $(i+1, i-j)$
assume $j \geq i$;
$i := i-1$;
↓$L_1$ :  $(i+1, i+1)$
```

Path **(4)**:

$$i+1 \geq 0 \wedge i-j \geq 0 \wedge j \geq i \rightarrow ((i-1)+1, (i-1)+1) <_2 (i+1, i-j)$$

All VCs are valid. Hence, BubbleSort always halts.

Construction of last VC

The verification condition for Path (4) is generated as follows:

$$\begin{aligned} & \mathsf{wp}((i+1, i+1) <_2 (i_0+1, i_0-j_0),\ \texttt{assume}\ j \geq i; i := i-1) \\ \Leftrightarrow\quad & \mathsf{wp}(((i-1)+1, (i-1)+1) <_2 (i_0+1, i_0-j_0),\ \texttt{assume}\ j \geq i) \\ \Leftrightarrow\quad & j \geq i\ \rightarrow\ (i, i) <_2 (i_0+1, i_0-j_0) \end{aligned}$$

Replace back $(i_0, j_0)\ \rightarrow\ (i, j)$:

$$j \geq i\ \rightarrow\ (i, i) <_2 (i+1, i-j),$$

producing the VC

$$i+1 \geq 0\ \wedge\ i-j \geq 0\ \wedge\ j \geq i\ \rightarrow\ (i, i) <_2 (i+1, i-j).$$

Example 3: Shortcut (backward substitution)

VC: $\boxed{i+1 \geq 0 \wedge i-j \geq 0 \wedge j \geq i \rightarrow (i, i) <_2 (i+1, i-j)}$

$$i+1 \geq 0 \wedge i-j \geq 0 \wedge j \geq i \rightarrow (i, i) <_2 (i_0+1, i_0-j_0)$$

@L_2 : $i+1 \geq 0 \wedge i-j \geq 0$

$j \geq i \rightarrow (i, i) <_2 (i_0+1, i_0-j_0)$

↓ L_2 : $(i+1, i-j)$

$j \geq i \rightarrow (i, i) <_2 (i_0+1, i_0-j_0)$

assume $j \geq i$;

$(i, i) <_2 (i_0+1, i_0-j_0)$

$i := i-1$;

$(i+1, i+1) <_2 (i_0+1, i_0-j_0)$

↓ L_1 : $(i+1, i+1)$ $\Uparrow$

Example 3: Shortcut (backward substitution)

VC: $\boxed{i+1 \geq 0 \wedge i-j \geq 0 \wedge j \geq i \rightarrow (i,i) <_2 (i+1, i-j)}$

$@L_2: \; i+1 \geq 0 \wedge i-j \geq 0$

$\qquad\qquad j \geq i \rightarrow (i,i) <_2 (i+1, i-j)$

$\downarrow L_2: \; (i+1, i-j)$

$\qquad\qquad j \geq i \rightarrow (i,i) <_2 \;?$

assume $j \geq i$;

$\qquad\qquad (i,i) <_2 \;?$

$i := i-1$;

$\qquad\qquad (i+1, i+1) <_2 \;?$

$\downarrow L_1: \; (i+1, i+1)$ $\qquad\qquad \Uparrow$

Example: Binary Search — recursive calls

Choose $(\mathbb{N}, <)$ as well-founded set and ranking function $\delta : u - \ell + 1$

```
@pre $u - \ell + 1 \geq 0$
@post $\top$
$\downarrow u - \ell + 1$      ... ranking function $\delta$
bool BinarySearch(int[] $a$, int $\ell$, int $u$, int $e$) {
  if ($\ell > u$) return false;
  else {
    int $m := (\ell + u)$ div 2;
    if ($a[m] = e$) return true;
    else if ($a[m] < e$) return
      @$R_1 : u - (m+1) + 1 \geq 0$
      BinarySearch($a, m+1, u, e$);
    else return
      @$R_2 : (m-1) - \ell + 1 \geq 0$
      BinarySearch($a, \ell, m-1, e$);
  }
}
```

Show $@R_1$ and $@R_2$ are P-invariant

Show decrease in $u - \ell + 1$:

(1)

```
@pre $u - \ell + 1 \geq 0$
$\downarrow u - \ell + 1$
assume $\ell \leq u$;
$m := (\ell + u)$ div 2;
assume $a[m] \neq e$;
assume $a[m] < e$;
$\downarrow u - (m+1) + 1$
```

Verification condition:

$$\begin{aligned} & u - \ell + 1 \geq 0 \;\wedge\; \ell \leq u \;\wedge\; \cdots \\ \rightarrow\quad & u - (((\ell + u) \;\mathtt{div}\; 2) + 1) + 1 < u - \ell + 1 \end{aligned}$$

Show decrease in $u - \ell + 1$:

(2)

```
@pre $u - \ell + 1 \geq 0$
$\downarrow u - \ell + 1$
assume $\ell \leq u$;
$m := (\ell + u)$ div 2;
assume $a[m] \neq e$;
assume $a[m] \geq e$;
$\downarrow (m-1) - \ell + 1$
```

Verification condition:

$$\begin{aligned} & u - \ell + 1 \geq 0 \;\wedge\; \ell \leq u \;\wedge\; \cdots \\ \rightarrow\quad & (((\ell + u) \;\mathtt{div}\; 2) - 1) - \ell + 1 < u - \ell + 1 \end{aligned}$$

Note: two other basic paths (... `return false` and ... `return true`) are irrelevant to the termination argument (recursion ends at each).

Both VCs are $T_{\mathbb{Z}}$-valid. Thus BinarySearch halts on all input in which ℓ is initially at most $u + 1$.