AAA528: Computational Logic

Lecture 7 — Program Verification (2)

Hakjoo Oh
2018 Fall

Total Correctness

- Total correctness = Partial correctness + Termination
- Total correctness of a function asserts that if the precondition holds on entry, then the function eventually halts and the postcondition holds.

Well-Founded Relations

- Termination proof is based on well-founded relations.
- A binary relation $\prec$ over a set S is well-founded iff there does not exist an infinite sequence $s_1, s_2, \ldots$ of elements of S such that

$$s_1 \succ s_2 \succ \cdots .$$

- For example, the relation $<$ is well-founded over the natural numbers, because any sequence of natural numbers decreasing according to $<$ is finite: e.g.,

$$1023 > 39 > 30 > 29 > 8 > 3 > 0.$$

However, the relation $<$ is not well-founded over the rationals or reals.

Lexicographic Relations

- A useful class of well-founded relations.
- From a set of pairs of sets and well-founded relations:

  $$(S_1, \prec_1), \ldots, (S_m, \prec_m)$$

  construct the set

  $$S = S_1 \times \cdots \times S_m$$

  and define the relation $\prec$:

  $$(s_1, \ldots, s_m) \prec (t_1, \ldots, t_m) \iff \bigvee_{i=1}^{m} \left(s_i \prec_i t_i \land \bigwedge_{j=1}^{i-1} s_j = t_j\right)$$

- For example, let $S = \mathbb{N}^3$ and $<_3$ be triples of natural numbers and the natural lexicographic extension of $<$ to such triples, respectively:

  $$(11, 9, 104) <_3 (11, 13, 3)$$

Proving Termination

- Define a set S with a well-founded relation $\prec$.
    - We usually choose as S the set of n-tuples of natural numbers and as $\prec_n$ the lexicographic extension $<_n$[1] of $<$, where n varies according to the application.
- Find a *ranking function* δ mapping program states to S such that δ decreases according to $\prec$ along every basic path.
- Then, since $\prec$ is well-founded, there cannot exist an infinite sequence of program states.

[1]When $n = 2$, $(a, b) <_2 (a', b') \iff a < a' \lor (a = a' \land b < b')$

Example: Bubble Sort

For each loop, annotate a ranking function:

```
@pre : ⊤
@post : ⊤
bool BubbleSort (int a[]) {
  int[] a := a_0
  @L_1 : i + 1 ≥ 0
  ↓ (i + 1, i + 1)
  for (int i := |a| − 1; i > 0; i := i − 1) {
    @L_2 : i + 1 ≥ 0 ∧ i − j ≥ 0
    ↓ (i + 1, i − j)
    for (int j := 0; j < i; j := j + 1) {
      if (a[j] > a[j + 1]) {
        int t := a[j];
        int a[j] := a[j + 1];
        int a[j + 1] := t;
      }
    }
  }
  return a;
}
```

Basic Paths

Prove that the ranking functions decrease along each basic paths.

$$
\begin{array}{ll}
(1) & @L_1 : i + 1 \geq 0 \\
 & \downarrow L_1 : (i+1, i+1) \\
 & \textbf{assume } i > 0; \\
 & j := 0; \\
 & \downarrow L_2 : (i+1, i-j) \\
(2) & L_2 : i + 1 \geq 0 \wedge i - j \geq 0 \\
 & \downarrow L_2 : (i+1, i-j) \\
 & \textbf{assume } j < i; \\
 & \textbf{assume } a[j] > a[j+1]; \\
 & t := a[j]; \\
 & a[j] := a[j+1]; \\
 & a[j+1] := t; \\
 & j := j + 1; \\
 & \downarrow L_2 : (i+1, i-j)
\end{array}
$$

Basic Paths

$$\begin{array}{ll}
(3) & L_2 : i+1 \geq 0 \land i-j \geq 0 \\
& \downarrow L_2 : (i+1, i-j) \\
& \textsf{assume } j < i; \\
& \textsf{assume } a[j] \leq a[j+1]; \\
& j := j+1; \\
& \downarrow L_2 : (i+1, i-j) \\
(4) & L_2 : i+1 \geq 0 \land i-j \geq 0 \\
& \downarrow L_2 : (i+1, i-j) \\
& \textsf{assume } j \geq i; \\
& i := i-1; \\
& \downarrow L_1 : (i+1, i+1)
\end{array}$$

Other basic paths are not relevant to proving termination.

Verification Conditions

The verification condition of basic path

$$\begin{array}{l} @F \\ \downarrow \delta[\bar{x}] \\ S_1; \\ \vdots \\ S_n; \\ \downarrow \kappa[\bar{x}] \end{array}$$

is

$$F \rightarrow \mathsf{wp}(\kappa \prec \delta[\bar{x}_0], S_1; \ldots; S_n)\{\bar{x}_0 \mapsto \bar{x}\}$$

The value of κ after executing the statements is less than the value of δ before executing the statements. The annotation F can provide extra invariant to prove the relation.

Example

To derive the VC for the path

$$\begin{array}{ll} (4) & L_2 : i+1 \geq 0 \wedge i-j \geq 0 \\ & \downarrow L_2 : (i+1, i-j) \\ & \textbf{assume } j \geq i; \\ & i := i-1; \\ & \downarrow L_1 : (i+1, i+1) \end{array}$$

compute

$$\begin{array}{l} \textbf{wp}((i+1, i+1) \prec_2 (i_0+1, i_0-j_0), \textbf{assume } j \geq i; i := i-1) \\ \iff \textbf{wp}(((i_0-1)+1, (i_0-1)+1) <_2 (i_0+1, i_0-j_0), \textbf{assume } j \geq i) \\ \iff j \geq i \rightarrow (i, i) <_2 (i_0+1, i_0-j_0) \end{array}$$

Then, replace the variables:

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

The VC:

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

Exercise

Compute the verification conditions for the basic paths (1)–(3).

Example: Binary Search

$@\text{pre} : u - l + 1 \geq 0$
$@\text{post} : \top$
$\downarrow u - l + 1$

```
bool BinarySearch (int a[], int l, int u, int e) {
  if (l > u) return false;
  else {
    int m := (l + 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, l, m − 1, e)
  }
}
```

Basic Paths

$$
\begin{array}{ll}
(1) & \textbf{pre} : u - l + 1 \geq 0 \\
& \downarrow u - l + 1 \\
& \textbf{assume } l \leq u; \\
& m := (l + u) \textbf{ div } 2; \\
& \textbf{assume } a[m] \neq e \\
& \textbf{assume } a[m] < e \\
& \downarrow u - (m + 1) + 1
\end{array}
$$

VC:

$$u - l + 1 \geq 0 \wedge l \leq u \wedge \cdots \rightarrow u - (((l + u) \textbf{ div } 2) + 1) + 1 < u - l + 1$$

Basic Paths

$$\begin{array}{ll} (2) & \textsf{pre}: u - l + 1 \geq 0 \\ & \textsf{assume}\ l \leq u; \\ & m := (l + u)\ \textsf{div}\ 2; \\ & \textsf{assume}\ a[m] \neq e \\ & \textsf{assume}\ a[m] \geq e \\ & \downarrow (m - 1) - l + 1 \end{array}$$

VC:

$$u - l + 1 \geq 0 \land l \leq u \land \cdots \to (((l + u)\ \textsf{div}\ 2) - 1) - l + 1 < u - l + 1$$

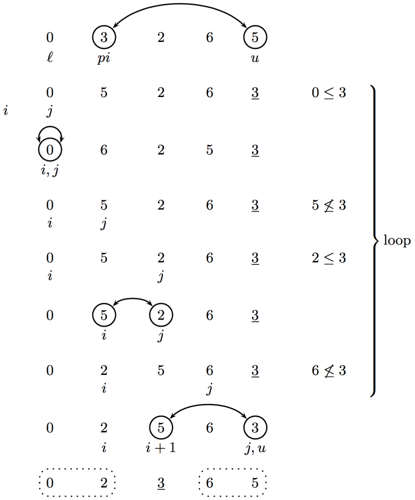
Example: QuickSort

Prove that QuickSort returns a sorted array and always halts.

```
typedef struct qs {
  int pivot;
  int[] array;
} qs;

@pre ⊤
@post sorted(rv, 0, |rv| − 1)
int[] QuickSort(int[] a) {
  return qsort(a, 0, |a| − 1);
}

@pre ⊤
@post ⊤
int[] qsort(int[] a₀, int ℓ, int u) {
  int[] a := a₀;
  if (ℓ ≥ u) return a;
  else {
    qs p := partition(a, ℓ, u);
    a := p.array;
    a := qsort(a, ℓ, p.pivot − 1);
    a := qsort(a, p.pivot + 1, u);
    return a;
  }
}
```

```
@pre ⊤
@post ⊤
qs partition(int[] a₀, int ℓ, int u) {
  int[] a := a₀;
  int pi := random(ℓ, u);
  int pv := a[pi];
  a[pi] := a[u];
  a[u] := pv;

  int i := ℓ − 1;
  for @ ⊤
    (int j := ℓ; j < u; j := j + 1) {
    if (a[j] ≤ pv) {
      i := i + 1;
      t := a[i];
      a[i] := a[j];
      a[j] := t;
    }
  }

  t := a[i + 1];
  a[i + 1] := a[u];
  a[u] := t;
  return
    { pivot = i + 1;
      a = a;
    };
}
```

QuickSort


0 3 2 6 5
ℓ pi u
0 5 2 6 3 0 ≤ 3
i j
0 6 2 5 3
i, j
0 5 2 6 3 5 ≰ 3
i j
0 5 2 6 3 2 ≤ 3
i j
0 5 2 6 3
i j
0 2 5 6 3 6 ≰ 3
i j
0 2 5 6 3
i i + 1 j, u
0 2 3 6 5
loop

Function Specification

$$\text{@pre}\begin{bmatrix}0 \le \ell \ \wedge \ u < |a_0| \\ \wedge \ \mathsf{partitioned}(a_0, 0, \ell-1, \ell, u) \\ \wedge \ \mathsf{partitioned}(a_0, \ell, u, u+1, |a_0|-1)\end{bmatrix}$$

$$\text{@post}\begin{bmatrix}|rv| = |a_0| \ \wedge \ \mathsf{beq}(rv, a_0, 0, \ell-1) \ \wedge \ \mathsf{beq}(rv, a_0, u+1, |a_0|-1) \\ \wedge \ \mathsf{partitioned}(rv, 0, \ell-1, \ell, u) \\ \wedge \ \mathsf{partitioned}(rv, \ell, u, u+1, |rv|-1) \\ \wedge \ \mathsf{sorted}(rv, \ell, u)\end{bmatrix}$$

$\texttt{int}[]$ qsort($\texttt{int}[]\ a_0,\ \texttt{int}\ \ell,\ \texttt{int}\ u$)

$$\text{@pre}\begin{bmatrix}0 \le \ell \ \wedge \ u < |a_0| \\ \wedge \ \mathsf{partitioned}(a_0, 0, \ell-1, \ell, u) \\ \wedge \ \mathsf{partitioned}(a_0, \ell, u, u+1, |a_0|-1)\end{bmatrix}$$

$$\text{@post}\begin{bmatrix}|rv.array| = |a_0| \ \wedge \ \mathsf{beq}(rv.array, a_0, 0, \ell-1) \\ \wedge \ \mathsf{beq}(rv.array, a_0, u+1, |a_0|-1) \\ \wedge \ \mathsf{partitioned}(rv.array, 0, \ell-1, \ell, u) \\ \wedge \ \mathsf{partitioned}(rv.array, \ell, u, u+1, |rv.array|-1) \\ \wedge \ \ell \le rv.pivot \le u \\ \wedge \ \mathsf{partitioned}(rv.array, \ell, rv.pivot-1, rv.pivot, rv.pivot) \\ \wedge \ \mathsf{partitioned}(rv.array, rv.pivot, rv.pivot, rv.pivot+1, u)\end{bmatrix}$$

qs partition($\texttt{int}[]\ a_0,\ \texttt{int}\ \ell,\ \texttt{int}\ u$)

Termination Argument

```
@pre u − ℓ + 1 ≥ 0
@post ⊤
↓ δ₂ : u − ℓ + 1
int[] qsort(int[] a₀, int ℓ, int u) {
  int[] a := a₀;
  if (ℓ ≥ u) return a;
  else {
    qs p := partition(a, ℓ, u);
    a := p.array;
    a := qsort(a, ℓ, p.pivot − 1);
    a := qsort(a, p.pivot + 1, u);
    return a;
  }
}

@pre ℓ ≤ u
@post ℓ ≤ rv.pivot ∧ rv.pivot ≤ u
qs partition(int[] a₀, int ℓ, int u) {
  ⋮
  int i := ℓ − 1;
  for
    @L₁ : ℓ ≤ j ∧ j ≤ u ∧ ℓ − 1 ≤ i ∧ i < j
    ↓ δ₁ : u − j
    (int j := ℓ; j < u; j := j + 1) {
    ⋮
  }
  ⋮
  return
    { pivot = i + 1;
      a = a;
    };
}
```

Exercise 1: Absolute Value

Prove the partial correctness of the function:

$$
\begin{array}{l}
@\mathsf{pre}\ \top \\
@\mathsf{post}\ \forall i.\ 0 \le i < |rv|\ \rightarrow\ rv[i] \ge 0 \\
\texttt{int}[]\ \mathsf{abs}(\texttt{int}[]\ a_0)\ \{ \\
\quad \texttt{int}[]\ a := a_0; \\
\quad \texttt{for}\ @\ \top \\
\qquad (\texttt{int}\ i := 0;\ i < |a|;\ i := i + 1)\ \{ \\
\qquad \texttt{if}\ (a[i] < 0)\ \ \{ \\
\qquad\quad a[i]\ :=\ -a[i]; \\
\qquad \} \\
\quad \} \\
\quad \texttt{return}\ a; \\
\}
\end{array}
$$

That is, annotate the function; list basic paths and verification conditions; and argue that the VC's are valid.

Exercise 2: Insertion Sort

Prove the partial correctness of the function:

```
@pre ⊤
@post sorted(rv, 0, |rv| − 1)
int[] InsertionSort(int[] a_0) {
  int[] a := a_0;
  for @ ⊤
    (int i := 1; i < |a|; i := i + 1) {
    int t := a[i];
    for @ ⊤
      (int j := i − 1; j ≥ 0; j := j − 1) {
      if (a[j] ≤ t) break;
      a[j + 1] := a[j];
    }
    a[j + 1] := t;
  }
  return a;
}
```

That is, annotate the function; list basic paths and verification conditions; and argue that the VC's are valid.