Verification

Lecture 27

Bernd Finkbeiner

UNIVERSITÄT
DES
SAARLANDES

Plan for today

- Deductive verification: basic mechanics
  - Partial correctness
  - Total correctness

REVIEW: Partial Correctness

A function is partially correct if

- when the function's precondition is satisfied on entry,
- its postcondition is satisfied when the function returns (if it ever does).

Inductive assertion method

- Each function and its annotion are reduced to a finite set of verification conditions (VCs)
- VCs are formulas of first-order logic
- If all VCs are valid, then the function is partially correct.

REVIEW: Verification condition

- $wp(F, \texttt{assume}\ c) \Leftrightarrow c \rightarrow F$
- $wp(F[v], v := e) \Leftrightarrow F[e]$
- $wp(F, S_1; S_2; \ldots; S_{n-1}; S_n) \Leftrightarrow wp(wp(F, S_n), S_1; S_2; \ldots S_{n-1})$

The verification condition of basic path

```
@ F
S_1;
...
S_n;
@ G
```

is

$$F \rightarrow wp(G, S_1; \ldots; S_n).$$

Traditionally, this verification condition is denoted by the Hoare triple $\{F\}\ S_1; \ldots; S_n\ \{G\}$.

Total correctness

- Total correctness: If the input satisfies the precondition, the function eventually halts and produces output that satisfies the postcondition.
- Termination: The function halts on every input satisfying the precondition.
- Total correctness = partial correctness + termination

Termination proofs: Find a ranking function δ, mapping program states to a set with a well-founded relation $\prec$, such that δ decreases along every basic path.

Well-founded relations

A binary predicate $\prec$ over a set S is a well-founded relation iff there does not exist an infinite decreasing sequence

$$s_1 \succ s_2 \succ s_3 \succ \ldots$$

where $s_i \in S$. (Notation: $s \succ t$ iff $t \prec s$.)

Examples:

- $<$ is well-founded over the natural numbers.
- $<$ is not well-founded over the rationals in $[0, 1]$.

  $$1 > \frac{1}{2} > \frac{1}{3} > \frac{1}{4} > \ldots$$

  is an infinite decreasing sequence
- $<$ is not well-founded over the integers.
- The strict sublist relation is well-founded over the set of all lists.

Lexicographic relations

Given pairs $(S_i, \prec_i)$ of sets S_i and well-founded relations $\prec_i$

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

construct

$$S = S_1 \times \ldots \times S_m,$$

i.e., the set of m-tuples $(s_1, \ldots, s_m)$ where each $s_i \in S_i$.
Define lexicographic relation $\prec$ over S as

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

for $s_i, t_i \in S_i$.

If $(S_1, \prec_1), \ldots, (S_m, \prec_m)$ are well-founded, so is $(S, \prec)$.

Proving termination

- Choose set W with well-founded relation $\prec$.

  Usually the set of n-tuples of natural numbers with the lexicographic relation.

- Find ranking function δ mapping program states to W such that δ decreases according to $\prec$ along every basic path.

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

Verification conditions

For every basic path

$$
\begin{array}{l}
@\,L_1 : F \\
\downarrow \delta[\vec{x}] \\
S_1 \\
\vdots \\
S_n \\
\downarrow \kappa[\vec{x}] \\
@\,L_j : G
\end{array}
$$

we prove the verification condition

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

Example: Bubble sort

```
@pre ⊤
@post ⊤
int[] BubbleSort(int[] a₀) {
int[] a := a₀;
for @ L₁ : i + 1 ≥ 0
  ↓ (i + 1, i + 1)
  (int i := |a| − 1; i > 0; i := i − 1) {
  for @ L₂ : i + 1 ≥ 0 ∧ i − j ≥ 0
    ↓ (i + 1, i − j)
    (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;
```

Example: Ackermann function

```
@pre x ≥ 0 ∧ y ≥ 0
@post rv ≥ 0
↓ (x, y)
int Ack(int x, int y) {
  if (x = 0) {
    return y + 1;
  }
  else if (y = 0) {
    return Ack(x − 1, 1);
  }
  else {
    int z := Ack(x, y − 1);
    return Ack(x − 1, z);
  }
}
```

Ackermann function

Verification conditions for the three basic paths

1. $x \geq 0 \ \wedge \ y \geq 0 \ \wedge \ x \neq 0 \ \wedge \ y = 0 \ \Rightarrow \ (x-1, 1) <_2 (x, y)$
2. $x \geq 0 \ \wedge \ y \geq 0 \ \wedge \ x \neq 0 \ \wedge \ y \neq 0 \ \Rightarrow \ (x, y-1) <_2 (x, y)$
3. $x \geq 0 \ \wedge \ y \geq 0 \ \wedge \ x \neq 0 \ \wedge \ y \neq 0 \ \wedge \ v_1 \geq 0 \ \Rightarrow \ (x-1, v_1) <_2 (x, y)$

Compute

$$\begin{aligned}
& \mathrm{wp}((x-1, z) <_2 (x_0, y_0) \\
& \quad , \texttt{assume}\ x \neq 0;\ \texttt{assume}\ y \neq 0;\ \texttt{assume}\ v_1 \geq 0;\ z := v_1) \\
\Leftrightarrow\ & \mathrm{wp}((x-1, v_1) <_2 (x_0, y_0) \\
& \quad , \texttt{assume}\ x \neq 0;\ \texttt{assume}\ y \neq 0;\ \texttt{assume}\ v_1 \geq 0) \\
\Leftrightarrow\ & x \neq 0 \ \wedge \ y \neq 0 \ \wedge \ v_1 \geq 0 \ \rightarrow \ (x-1, v_1) <_2 (x_0, y_0)
\end{aligned}$$

Renaming x_0 and y_0 to x and y, respectively, gives

$$x \neq 0 \ \wedge \ y \neq 0 \ \wedge \ v_1 \geq 0 \ \rightarrow \ (x-1, v_1) <_2 (x, y) \ .$$

Noting that path **(3)** begins by asserting $x \geq 0 \ \wedge \ y \geq 0$, we finally have

$$x \geq 0 \ \wedge \ y \geq 0 \ \wedge \ x \neq 0 \ \wedge \ y \neq 0 \ \wedge \ v_1 \geq 0 \ \Rightarrow \ (x-1, v_1) <_2 (x, y) \ .$$

Simple heuristics for developing annotations

Basic facts in loop invariants

Loop of LinearSearch:

```
for @ L : ⊤
  (int i := l; i ≤ u; i := i + 1) {
  if (a[i] = e) return true;
}
```

Because of the initialization of i, the loop guard, and because i is only modified in the loop update, we know that at L, $l \leq i \leq u + 1$.

```
for @ L : l ≤ i ≤ u + 1
  (int i := l; i ≤ u; i := i + 1) {
  if (a[i] = e) return true;
}
```

Note that on the final iteration, the loop guard is not true.

Basic facts in loop invariants

Loops of BubbleSort:

```
for @ L_1 : −1 ≤ i < |a|
  (int i := |a| − 1; i > 0; i := i − 1) {
  for @ L_2 : 0 ≤ i < |a| ∧ 0 ≤ j ≤ i
    (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;
    }
  }
}
```

The precondition method

1. Identify a fact F that is known at a location L in the function but that is not supported by annotations earlier in the function.

$$@L : F$$

2. Repeat:
   - Compute the weakest precondition of F backward through the function, ending at loop invariants or at the beginning of the function.
   - At each new annotation location L', generalize the new facts to new formula F'.

$$@L' : F'$$

Example: Linear search

```
@post rv ↔ ∃i. l ≤ i ≤ u ∧ a[i] = e
  for @ L : l ≤ i ≤ u + 1
    (int i := l; i ≤ u; i := i + 1) {
    if (a[i] = e) return true;
  }
  return false;
```

(4)	@ $L : F_1 : l \leq i \leq u + 1$
	S_1 : assume $i > u$
	$S_2 : rv :=$ false
	@post $F_2 : rv \leftrightarrow \exists i.\ l \leq i \leq u \wedge a[i] = e$

The VC $\{F_1\}\ S_1; S_2\ \{F_2\}$ is not valid!

Example: Linear search

(4)	@ $L : F_1 : l \leq i \leq u + 1$
	S_1 : assume $i > u$
	$S_2 : rv :=$ false
	@post $F_2 : rv \leftrightarrow \exists i.\ \ l \leq j \leq u \wedge a[j] = e$

We propagate F_2 back to the loop invariant:

$$
\begin{array}{ll}
 & wp(F_2, S_1; S_2) \\
\Leftrightarrow & wp(wp(F_2, rv := \texttt{false}), \texttt{assume}\ i > u) \\
\Leftrightarrow & i > u \rightarrow \forall j.\ l \leq j \leq u \rightarrow a[j] \neq e
\end{array}
$$

With some intuition...

$$G' : \forall j.\ l \leq j < i \rightarrow a[j] \neq e$$

Summary

- Specification of sequential programs via function preconditions and function postconditions. Other annotations: loop invariants, assertions.
- Partial correctness is proven with an inductive argument. Additional annotations strengthen the inductive argument. Key notions: basic paths, program state, verification conditions, inductive invariants.
- Termination is proven by mapping the program states to a domain with a well-founded relation via a ranking function. Typically, additional annotations are needed.

→ basic mechanics of deductive verification.