Softwaretechnik

Program verification

Albert-Ludwigs-Universität Freiburg

June 20, 2013

Road Map

- Program verification
- Automatic program verification
  - Programs with loops
  - Programs with recursive function calls

Proving Program Correctness: General Approach

UNI FREIBURG

Program annotation

- Annotation @F at program location L asserts that formula F is true whenever program control reaches L
- Special annotation: function specification
  - Precondition = specifies what should be true upon entering
  - Postcondition = specifies what must hold after executing

Proving Program Correctness

- Input: Program with annotations
- Translate input to first order formula f
- Validity of f implies program correctness

Outline

- Proving partial correctness
  - Programs with loops

Proving Partial Correctness

Recall

A function f is partially correct if
when f's precondition is satisfied on entry and f terminates,
then f's postcondition is satisfied.

Proving Partial Correctness

Recall

A function f is partially correct if
when f's precondition is satisfied on entry and f terminates,
then f's postcondition is satisfied.

Automatic Verification

- Function + annotation is transformed to finite set of FOL formulae, the verification conditions (VCs)
- If all VCs are valid, then the function obeys its specification (partially correct)

Programs with Loops

Loop invariants

- Each loop must be annotated with a loop invariant, @L
- `while` loop: L must hold
  - at the beginning of each iteration before the loop condition is evaluated
- `for` loop: L must hold
  - after the loop initialization, and
  - before the loop condition is evaluated

Basic Paths: Loops

To handle loops, we break the function into basic paths.

Basic Path

@ $\leftarrow$ precondition or loop invariant

finite sequence of instructions
(no loop invariants)

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

Basic Paths: Conditionals

Basic paths split at conditionals

Replace each path $BP[\texttt{if } B \texttt{ then } S_1 \texttt{ else } S_2]$ by two paths

- $BP[\texttt{assume } B; S_1]$
- $BP[\texttt{assume } \neg B; S_2]$

Semantics of "`assume` B"

Execution ends unless B holds

Example: LinearSearch

```
@pre 0 ≤ ℓ ∧ u < a.length
@post rv ↔ ∃i. ℓ ≤ i ≤ u ∧ a[i] = e
bool LinearSearch(int[] a, int ℓ, int u, int e) {
  for
    @L : ℓ ≤ i ∧ (∀j. ℓ ≤ j < i → a[j] ≠ e)
    (int i := ℓ; i ≤ u; i := i + 1) {
    if (a[i] = e) return true;
  }
  return false;
}
```

Example: Basic Paths of LinearSearch

(1)

@pre $0 \leq \ell \ \wedge \ u < a.length$
$i := \ell;$
@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 $rv \ \leftrightarrow \ \exists j.\ \ell \leq j \leq u \ \wedge \ a[j] = e$

Example: Basic Paths of LinearSearch

(3)

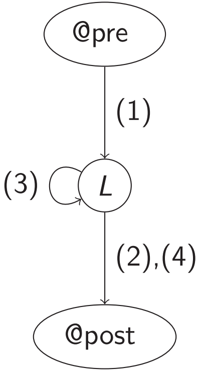
$@L:\ \ell \leq i\ \wedge\ \forall j.\ \ell \leq j < i\ \rightarrow\ a[j] \neq e$
`assume` $i \leq u$;
`assume` $a[i] \neq e$;
$i := i + 1$;
$@L:\ \ell \leq i\ \wedge\ \forall j.\ \ell \leq j < i\ \rightarrow\ a[j] \neq e$

(4)

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

Example: Basic Paths of LinearSearch

Visualization of basic paths of LinearSearch



Proving Partial Correctness

Goal

- Prove that annotated function f agrees with annotations
- Transform f to finite set of verification conditions VC
- Validity of VC implies that function behaviour agrees with annotations

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

- Informally: What must hold before executing statement S to ensure that formula F holds afterwards?
- $\text{wp}(F, S)$ = weakest formula such that executing S results in formula that satisfies F
- For all states σ such that $\sigma \in \text{wp}(F, S)$: successor state $\mathcal{S}[\![S]\!]\sigma \in F$.

Proving Partial Correctness

Weakest preconditions for each statement

- Assumption: What must hold before statement `assume` B is executed to ensure that F holds afterward?

$$\mathrm{wp}(F,\ \texttt{assume}\ B)\ \Leftrightarrow\ B\ \rightarrow\ F$$

- Assignment: What must hold before statement $x := e$ is executed to ensure that $F[x]$ holds afterward?

$$\mathrm{wp}(F[x],\ x := e)\ \Leftrightarrow\ F[e]$$

  ("substitute x with e")

- Sequence of statements $S_1; \ldots ; S_n$ $(n > 1)$,
  $\mathrm{wp}(F,\ S_1; \ldots ; S_n)\ \Leftrightarrow\ \mathrm{wp}(\mathrm{wp}(F,\ S_n),\ S_1; \ldots ; S_{n-1})$

Proving Partial Correctness

Verification condition of basic path

$$
\begin{array}{l}
@\ F \\
S_1; \\
\dots \\
S_n; \\
@\ G
\end{array}
$$

is defined as

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

This verification condition is often denoted by the Hoare triple

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

Proving Partial Correctness

Approach

- Input: Annotated program
- Compute the set P of all basic paths (finite)
- For all $p \in P$: generate verification condition $VC(p)$
- Check validity of $\bigwedge_{p \in P} VC(p)$

Theorem

If $\bigwedge_{p \in P} VC(p)$ is valid, then each function agrees with its annotation.

Example 1: VC of 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)

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

which is valid.

Example 2: VC of basic path (2) of LinearSearch

(2)

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

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

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

Outline

- Proving partial correctness
  - Programs with recursive function calls

Basic Paths: Recursive Function Calls

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

Function specification

- Add function summary for each function call
- Instantiate pre- and postcondition with parameters of recursive call

Example: BinarySearch

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

sorted: weakly increasing order, i.e.

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

Function specifications

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

Example: BinarySearch

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

Example: Binary Search with Function Call Assertions

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

Summary

Automatic verification of sequential programs

- Goal: Proof of partial correctness
- Program specification
  - Pre- and postconditions
  - Loop invariants
- Tools
  - Basic paths
  - Weakest precondition
  - Verification conditions
  - Function summaries