Formal Methods and Specification (LS 2021) Lecture 5: Correctness of Programs with Control Structures

Stefan Ratschan

Katedra číslicového návrhu
Fakulta informačních technologií
České vysoké učení technické v Praze

Evropský sociální fond Praha & EU: Investujeme do vaší budoucnosti

Control Structures: Example:

if $x > y$ **then**
  $r \leftarrow x$
else
  $r \leftarrow y$
@ $r = \max\{x, y\}$

Two execution paths:

$r \leftarrow x$
@ $r = \max\{x, y\}$

$r \leftarrow y$
@ $r = \max\{x, y\}$

Executed only if certain conditions hold.

assume $x > y$
$r \leftarrow x$
@ $r = \max\{x, y\}$

assume $\neg[x > y]$
$r \leftarrow y$
@ $r = \max\{x, y\}$

Control Structures

$$
\begin{aligned}
&\textbf{if } x > y \textbf{ then}\\
&\quad r \leftarrow x\\
&\textbf{else}\\
&\quad r \leftarrow y\\
&\texttt{@}\ r = \max\{x, y\}
\end{aligned}
$$

assume $x > y$
$r \leftarrow x$
@ $r = \max\{x, y\}$

assume $\neg[x > y]$
$r \leftarrow y$
@ $r = \max\{x, y\}$

Corresponding verification conditions:

$$
\begin{aligned}
&\forall x, y, r\ .\\
&\qquad [x > y \wedge r = x] \Rightarrow\\
&\qquad\qquad r = \max\{x, y\}
\end{aligned}
$$

$$
\begin{aligned}
&\forall x, y, r\ .\\
&\qquad [\neg[x > y] \wedge r = y] \Rightarrow\\
&\qquad\qquad r = \max\{x, y\}
\end{aligned}
$$

Linearization of Control Structures

A basic path is a *basic path of a program* P iff the basic path

- corresponds to a sequence of program lines of P
  that can be executed in this order,
- either begins at the beginning of P,
  or at an @-assertion
  that is replaced by the corresponding **assume**-assertion, and
- contains **assume** ϕ instead of control structures, where ϕ is
  the condition under which program execution follows this path.

If for all basic paths π of a program P, $\models VC(\pi)$,
then the program is correct.

Linearization of **if-then-else**

if P **then**

$\ldots$

else

$\ldots$

Linearization:

- path following **if**: **assume** P
- path following **else**: **assume** $\neg P$

Basic Paths of **if-then-else** Cascades

if $x_1 > y_1$ **then**
    $r_1 \leftarrow x_1$
else
    $r_1 \leftarrow y_1$
if $x_2 > y_2$ **then**
    $r_2 \leftarrow x_2$
else
    $r_2 \leftarrow y_2$
if $r_1 > r_2$ **then**
    $r \leftarrow r_1$
else
    $r \leftarrow r_2$
@ $r = \max\{x_1, x_2, y_1, y_2\}$

Example of basic path

assume $x_1 > y_1$
$r_1 = x_1$
assume $x_2 > y_2$
$r_2 = x_2$
assume $r_1 > r_2$
$r = r_1$
@ $r = \max\{x_1, x_2, y_1, y_2\}$

$2^3 = 8$ basic paths with corresponding verification conditions

Basic Paths of **if-then-else** Cascades

if $x_1 > y_1$ **then**
  $r_1 \leftarrow x_1$
else
  $r_1 \leftarrow y_1$
@ $r_1 = \max\{x_1, y_1\}$
if $x_2 > y_2$ **then**
  $r_2 \leftarrow x_2$
else
  $r_2 \leftarrow y_2$
@ $r_1 = \max\{x_1, y_1\}$, $r_2 = \max\{x_2, y_2\}$
if $r_1 > r_2$ **then**
  $r \leftarrow r_1$
else
  $r \leftarrow r_2$
@ $r = \max\{x_1, x_2, y_1, y_2\}$

assume $x_1 > y_1$
$r_1 = x_1$
@ $r_1 = \max\{x_1, y_1\}$

assume $\neg[x_1 > y_1]$
$r_1 = y_1$
@ $r_1 = \max\{x_1, y_1\}$

$2 \times 3 = 6$ shorter basic paths with corresponding simpler verif. conditions

Basic Paths of **if-then-else** Cascades

@
if P_1 **then**
    ...
else
    ...
...
if P_n **then**
    ...
else
    ...
@

Result: 2^n basic paths

Basic Paths of **if-then-else** Cascades

@

if P_1 **then**

$\quad\ldots$

else

$\quad\ldots$

$\ldots$

if P_n **then**

$\quad\ldots$

else

$\quad\ldots$

@

Introduce intermediate assertions

$2n$ basic paths

Loops

$i \leftarrow 0$
while $a[i] \neq 0$ **do**
    $i \leftarrow i + 1$
$@\ a[i] = 0 \wedge \forall k \in \{0, \ldots, i-1\} \,.\, a[k] \neq 0$

Basic paths:

$i \leftarrow 0$
assume $a[i] \neq 0$
$i \leftarrow i + 1$
assume $a[i] \neq 0$
$i \leftarrow i + 1$
assume $a[i] \neq 0$
$i \leftarrow i + 1$
...
assume $a[i] = 0$
$@\ a[i] = 0 \wedge \forall k \in \{0, \ldots, i-1\} \,.\, a[k] \neq 0$

Problem: infinitely many basic paths!

Again: intermediate assertions

Loops

$i \leftarrow 0$
while $a[i] \neq 0$ **do**
    @
    $i \leftarrow i + 1$
@ $a[i] = 0 \wedge \forall k \in \{0, \ldots, i-1\} \,.\, a[k] \neq 0$

Intermediate assertions:

$i \leftarrow 0$
assume $a[i] \neq 0$
@
$i \leftarrow i + 1$
assume $a[i] \neq 0$
@
$i \leftarrow i + 1$
assume $a[i] \neq 0$
@
$i \leftarrow i + 1$
...
assume $a[i] = 0$
@ $a[i] = 0 \wedge \forall k \in \{0, \ldots, i-1\} \,.\, a[k] \neq 0$

Loops

$i \leftarrow 0$
while $a[i] \neq 0$ **do**
  @ ???$\forall k \in \{0, \dots, i\} \,.\, a[k] \neq 0$
  $i \leftarrow i + 1$
@ $a[i] = 0 \wedge \forall k \in \{0, \dots, i-1\} \,.\, a[k] \neq 0$

Basic paths:

$i \leftarrow 0$
assume $a[i] = 0$
@ $a[i] = 0 \wedge \forall k \in \{0, \dots, i-1\} \,.\, a[k] \neq 0$

$i \leftarrow 0$
assume $a[i] \neq 0$
@ ???$\forall k \in \{0, \dots, i\} \,.\, a[k] \neq 0$

assume ???$\forall k \in \{0, \dots, i\} \,.\, a[k] \neq 0$
$i \leftarrow i + 1$
assume $a[i] \neq 0$
@ ???$\forall k \in \{0, \dots, i\} \,.\, a[k] \neq 0$

assume ???$\forall k \in \{0, \dots, i\} \,.\, a[k] \neq 0$
$i \leftarrow i + 1$
assume $a[i] = 0$
@ $a[i] = 0 \wedge \forall k \in \{0, \dots, i-1\} \,.\, a[k] \neq 0$

Linearization of **while**

Assumption: loop contains at least one assertion `@`

while P **do**

   ...
   `@`
   ...

Instead of loop:

- basic paths entering or staying in loop: **assume** P
- leaving or jumping over loop : **assume** $\neg P$

If loop contains one assertion and no further control structures:
   4 resulting basic paths

Linearization of **for** Loop

assume $k \geq 5$
for $i \leftarrow 1$ **to** 10 **do**
    @ $k \geq 3$
    $k \leftarrow k + i$
@ $k \geq 0$

assume $k \geq 5$
assume $\neg 1 \leq 10$
@ $k \geq 0$

assume $1 \leq i \leq 10 \wedge k \geq 3$
$k \leftarrow k + i$
$i \leftarrow i + 1$
assume $i \leq 10$
@ $k \geq 3$

assume $k \geq 5$
assume $i = 1$
@ $k \geq 3$

assume $1 \leq i \leq 10 \wedge k \geq 3$
$k \leftarrow k + i$
$i \leftarrow i + 1$
assume $\neg i \leq 10$
@ $k \geq 0$

Corresponding Verification Conditions

assume $1 \leq i \leq 10 \wedge k \geq 3$
$k \leftarrow k + i$
$i \leftarrow i + 1$
assume $i \leq 10$
@ $k \geq 3$

assume $1 \leq i \leq 10 \wedge k \geq 3$
$k_1 \leftarrow k + i$
$i_1 \leftarrow i + 1$
assume $i_1 \leq 10$
@ $k_1 \geq 3$

Verification condition:

$$[i \leq i \wedge i \leq 10 \wedge k \geq 3 \wedge k_1 = k + i \wedge i_1 = i + 1 \wedge i_1 \leq 10] \Rightarrow k_1 \geq 3$$

Linearization of **for**

Assumptions:

- loop contains at least one assertion `@`
- loop does not modify i

for $i \leftarrow l$ **to** u **do**

  ...

  `@`

  ...

Instead of loop:

- path into loop from outside: **assume** $i = l \wedge l \leq u$
- path jumping over loop: **assume** $\neg l \leq u$
- path staying in loop: $i \leftarrow i + 1$; **assume** $i \leq u$
- path leaving loop, depending on programming language
  - $i \leftarrow i + 1$; **assume** $\neg i \leq u$ (e.g., C)
  - in some programming languages: **assume** $i = u$ (e.g., Pascal)

Moreover: path starting inside of loop: add $l \leq i \leq u$ to initial **assume**

Program Correctness and Loops

A program with loops may have
infinitely many basic paths and corresponding verification conditions

How to ensure a finite number of basic paths?

Every loop must contain at least one assertion
(*loop invariant*)

The loop invariant can be at any position in the loop

Nested loops: every loop needs an assertion, not just the innermost one!

How to come up with loop invariants?

General Program Linearization

Example:

if C_1 **then**
  $P_1 \ldots$
else
  $P_2 \ldots$
while C_2 **do**
  @ A_1
  $P_3 \ldots$
  while C_3 **do**
    @ A_2
    $P_4 \ldots$
  $P_5 \ldots$
$P_6 \ldots$
@ A_3

basic paths:

assume C_1; P_1; **assume** C_2; @ A_1
assume $\neg C_1$; P_2; **assume** C_2; @ A_1
assume A_1; P_3; **assume** C_3; @ A_2
assume A_1; P_3; **assume** $\neg C_3$; P_5; **assume** C_2; @ A_1
assume A_1; P_3; **assume** $\neg C_3$; P_5; **assume** $\neg C_2$; P_6; @ A_3
assume A_2; P_4; **assume** C_3; @ A_2
assume A_2; P_4; P_5; **assume** C_2; @ A_1
assume A_2; P_4; **assume** $\neg C_3$; P_5; **assume** $\neg C_2$; P_6; @ A_3

Verification Conditions vs. Assertion Failures

$i \leftarrow 1$
while $\top$ **do**
    @ $i \geq -1$
    $i \leftarrow 2i$

Does the assertion hold during program execution? Yes
Do all verification conditions hold? No!!!

The verification condition of the basic path

assume $i \geq -1$
$i \leftarrow 2i$
assume $\top$
@ $i \geq -1$

is

$$[i \geq -1 \wedge i_1 = 2i] \Rightarrow i_1 \geq -1$$

which does not hold. Counter-example: $\{i \leftarrow -1; i_1 \leftarrow -2\}$

Verification Conditions vs. Assertion Failures

Difference:

- assertions hold: global condition
  (for checking, one needs to understand the whole program)
- verification conditions hold: locally checkable condition

In practice:

- manually checking all VCs unrealistic
- make assertion as locally checkable as possible

Finding Loop Invariants: Example

Specification:

- Input: array a
- Output: r s.t. $r \Leftrightarrow \left[\exists k \,.\, 1 \leq k \leq n \wedge a[k] = 7\right]$

Program:

$r \leftarrow \bot$
for $i \leftarrow 1$ **to** n **do**
    if $a[i] = 7$ **then** $r \leftarrow \top$
@ $r \Leftrightarrow [\exists k \,.\, 1 \leq k \leq n \wedge a[k] = 7]$
return r

Loop Invariant for Example

$r \leftarrow \bot$
for $i \leftarrow 1$ **to** n **do**
  @ $\top$
  if $a[i] = 7$ **then** $r \leftarrow \top$
@ $r \Leftrightarrow [\exists k \,.\, 1 \leq k \leq n \wedge a[k] = 7]$
return r

1			4
4	3	7	6

i	r
1	$\bot$
2	$\bot$
3	$\bot$
4	$\top$

In general: finding invariants is an art, no technique can replace ideas!

Invariant involved in three verification conditions:

- holds in first loop iteration
- if it holds, and the loop is re-entered, then it must hold again
- if it holds, and the loop is left, then assertion after the loop must hold

Guess $\top$: must be strengthened

Guess $r \Leftrightarrow [\exists k \,.\, 1 \leq k \leq i-1 \wedge a[k] = 7]$

Checking the Invariant

$r \leftarrow \bot$
for $i \leftarrow 1$ **to** n **do**
  @ $r \Leftrightarrow [\exists k \,.\, 1 \leq k \leq i-1 \land a[k] = 7]$
  if $a[i] = 7$ **then** $r \leftarrow \top$
@ $r \Leftrightarrow [\exists k \,.\, 1 \leq k \leq n \land a[k] = 7]$
return r

1			4
4	3	7	6

i	r
1	$\bot$
2	$\bot$
3	$\bot$
4	$\top$

Paths through loop:

assume $r \Leftrightarrow [\exists k \,.\, 1 \leq k \leq i-1 \land a[k] = 7]$
assume $a[i] = 7$
$r \leftarrow \top$
$i \leftarrow i+1$
assume $i \leq n$
@ $r \Leftrightarrow [\exists k \,.\, 1 \leq k \leq i-1 \land a[k] = 7]$

assume $r \Leftrightarrow [\exists k \,.\, 1 \leq k \leq i-1 \land a[k] = 7]$
assume $a[i] \neq 7$
$i \leftarrow i+1$
assume $i \leq n$
@ $r \Leftrightarrow [\exists k \,.\, 1 \leq k \leq i-1 \land a[k] = 7]$

Checking the Invariant

$r \leftarrow \bot$
for $i \leftarrow 1$ **to** n **do**
  @ $r \Leftrightarrow [\exists k \,.\, 1 \leq k \leq i-1 \wedge a[k] = 7]$
  if $a[i] = 7$ **then** $r \leftarrow \top$
@ $r \Leftrightarrow [\exists k \,.\, 1 \leq k \leq n \wedge a[k] = 7]$
return r

Final basic paths:

assume $r \Leftrightarrow [\exists k \,.\, 1 \leq k \leq i-1 \wedge a[k] = 7]$
assume $a[i] = 7$
$r \leftarrow \top$
assume $i = n$
@ $r \Leftrightarrow [\exists k \,.\, 1 \leq k \leq n \wedge a[k] = 7]$

assume $r \Leftrightarrow [\exists k \,.\, 1 \leq k \leq i-1 \wedge a[k] = 7]$
assume $a[i] \neq 7$
assume $i = n$
@ $r \Leftrightarrow [\exists k \,.\, 1 \leq k \leq n \wedge a[k] = 7]$

Programs with Bugs

Example:

$r \leftarrow \bot$
for $i \leftarrow 1$ **to** $n - 1$ **do**
  @ $r \Leftrightarrow [\exists k \,.\, 1 \leq k \leq i - 1 \wedge a[k] = 7]$
  if $a[i] = 7$ **then** $r \leftarrow \top$
@ $r \Leftrightarrow [\exists k \,.\, 1 \leq k \leq n \wedge a[k] = 7]$
return r

Final basic paths:

assume $r \Leftrightarrow [\exists k \,.\, 1 \leq k \leq i - 1 \wedge a[k] = 7]$
assume $a[i] = 7$
$r \leftarrow \top$
assume $i = n - 1$
@ $r \Leftrightarrow [\exists k \,.\, 1 \leq k \leq n \wedge a[k] = 7]$

assume $r \Leftrightarrow [\exists k \,.\, 1 \leq k \leq i - 1 \wedge a[k] = 7]$
assume $a[i] \neq 7$
assume $i = n - 1$
@ $r \Leftrightarrow [\exists k \,.\, 1 \leq k \leq n \wedge a[k] = 7]$

Verification Condition: **if**

$i \leftarrow i + 1$
assume $i \leq n$
assume $a[i] = 7$
$r \leftarrow \top$
@ $i = n \Rightarrow [r \Leftrightarrow [\exists k \,.\, 1 \leq k \leq n \wedge a[k] = 7]]$

$$[i_1 = i + 1 \wedge i_1 \leq n \wedge a[i_1] = 7 \wedge r \wedge i_1 = n] \Rightarrow \\ [r \Leftrightarrow [\exists k \,.\, 1 \leq k \leq n \wedge a[k] = 7]]$$

So: verification condition holds

Intuition: after execution of the **if** branch,
the result is correct (independently of the initial state)

Verification Condition: **else**

$i \leftarrow i + 1$
assume $i \leq n$
assume $a[i] \neq 7$
@ $i = n \Rightarrow [r \Leftrightarrow [\exists k \, . \, 1 \leq k \leq n \wedge a[k] = 7]]$

$$[i_1 = i + 1 \wedge i_1 \leq n \wedge a[i_1] \neq 7 \wedge i_1 = n] \Rightarrow$$
$$[r \Leftrightarrow [\exists k \, . \, 1 \leq k \leq n \wedge a[k] = 7]$$

$$[a[n] \neq 7 \wedge i + 1 = n] \Rightarrow [r \Leftrightarrow [\exists k \, . \, 1 \leq k \leq n \wedge a[k] = 7]$$

So: does not hold, in general, we have to ensure this.

Automatization

Explicit support for writing and checking assertions is growing, for example

- Eiffel
- Microsoft Code Contracts
- ANSI/ISO C Specification Language
- Java Modeling Language (JML), OpenJML (http://www.openjml.org)
- KeY (http://www.key-project.org)
- TLA+ (Microsoft, also used by Amazon)
- ...

The resulting verification conditions can often be proved automatically.

Remaining problem: we need loop invariants

Loop Invariants and Program Development

develop before loop body

$r \leftarrow \bot$
for $i \leftarrow 1$ **to** n **do**
    @ $r \Leftrightarrow [\exists k \,.\, 1 \leq k \leq i \wedge a[k] = 7]]$
    ???
@ $r \Leftrightarrow [\exists k \,.\, 1 \leq k \leq n \wedge a[k] = 7]$
return r

Summary: Assertions for Program Correctness

For proving correctness of a program we need

1. at least one assertion in every loop, that is loop invariants
2. a proof of the verification condition of
   of every basic path of the program

Conclusion

In practice, usually no correctness proof needed

Still, writing assertions has many advantages

Localize understanding of program:
instead of thinking about correctness of whole program
just think about correctness from one assertion to the next

... even if incomplete, and unproved

Assertions represent the essence of correctness of a program:

- important documentation
- check correctness during program execution
- more and more: (semi)-automatic correctness proofs based on assertions

Literature I

Aaron Bradley and Zohar Manna. *The calculus of computation*. Springer, 2007.