Formal Methods and Specification (LS 2021) Lecture 4: Correctness of Programs Without 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

I/O Specifications and Program Correctness

Example:

- Input:
  - array a of length n s.t. $\forall i \in \{0, \dots, n-2\} \, . \, a[i] \leq a[i+1]$,
  - integer k
- Output: $p \in \{0, \dots, n-1\}$ s.t.
$$\begin{cases} a[p] = k, & \text{if such a } p \text{ exists, and} \\ -1, & \text{otherwise.} \end{cases}$$

As assertions:

assume $\forall i \in \{0, \dots, n-2\} \, . \, a[i] \leq a[i+1]$

...

@ $p \in \{0, \dots, n-1\}$ s.t. $a[p] = k$, if such a p exists, -1, otherwise

return p

Difference between assertions according to intended usage:

- **assume**: to be ensured by the user, from the outside
- **@**: to be ensured by the given program

Internal Program Correctness

assume I — // *input spec*

@ $i \neq 0$

$q \leftarrow 1000/i$

$\dots$

@ $l < 100$

$m \leftarrow a[l]$ — // *a has length* 100

$\dots$

@ O — // *output spec*

return

Program Correctness and Assertions

A program execution is *regular* iff
it satisfies all **assume**-assertions.

A program is *(partially) correct* iff
for every regular program execution
from the first program line to a program line with an @-assertion,
satisfies the final @-assertion.

Definitions not formal: "program execution" undefined

Tiny imperative programming language,
for which we all agree on its behavior

Program Correctness: Assignments

$y \leftarrow x$
$z \leftarrow x + y$
`@` $z \geq 0$

correct?

$$\forall x, y, z \;.\; [y = x \wedge z = x + y] \Rightarrow z \geq 0$$

demo (`assignments.cvc`)

Program Correctness: Assumptions

assume $x \geq 5$
$y \leftarrow x$
$z \leftarrow x + y$
@ $z \geq 0$

$$\forall x, y, z \,.\, [x \geq 5 \land y = x \land z = x + y] \Rightarrow z \geq 0$$

demo (`assignments_assume.cvc`)

Works, because assignments to y and z do not change x

What if an assignment changes x in between?

Program Correctness: Assumptions

assume $x \geq 5$
$y \leftarrow x$
$x \leftarrow -10$
$z \leftarrow x + y$
@ $z \geq 0$

x	y	z
5		
5	5	
-10	5	
-10	5	-5

$$\forall x, y, z \,.\, [x \geq 5 \wedge y = x \wedge x = -10 \wedge z = x + y] \Rightarrow z \geq 0$$

Shows program to be correct, but it is not! Where is the problem?

Program Correctness: Assumptions

assume $x \geq 5$
$y \leftarrow x$
$x_1 \leftarrow -10$
$z \leftarrow x_1 + y$
@ $z \geq 0$

x	y	z
5		
5	5	
-10	5	
-10	5	-5

$$\forall x, x_1, y, z \, . \, [x \geq 5 \wedge y = x \wedge x_1 = -10 \wedge z = x_1 + y] \Rightarrow z \geq 0$$

demo (`assignments_renamed.cvc`) counter-example

$\{x \mapsto 5, y \mapsto 5, x_1 \mapsto -10, z \mapsto -5\}$
(satisfies left-hand side, but not right-hand side of implication)

static single assignment form (SSA)

widely used in compiler construction, see Wikipedia

Program Correctness: Assignments

Original program:

$x \leftarrow 1$
$x \leftarrow x + 1$
`@` $x \geq 10$

SSA:

$x \leftarrow 1$
$x_1 \leftarrow x + 1$
`@` $x_1 \geq 10$

$$\forall x, x_1 \,.\, [x = 1 \wedge x_1 = x + 1] \Rightarrow x_1 \geq 10$$

Program Correctness: Array Assignments

$a[i] \leftarrow 7$
$a[i] \leftarrow 5$
$\ldots$

SSA???

Function symbols of arrays:

- $\cdot[\cdot] : \mathcal{A}[\mathcal{T}] \times \mathcal{N} \rightarrow \mathcal{T}$
- write : $\mathcal{A}[\mathcal{T}] \times \mathcal{N} \times \mathcal{T} \rightarrow \mathcal{A}[\mathcal{T}]$

Using those function symbols:

$a \leftarrow \text{write}(a, i, 7)$
$a \leftarrow \text{write}(a, i, 5)$
$\ldots$

SSA

$a_1 \leftarrow \text{write}(a, i, 7)$
$a_2 \leftarrow \text{write}(a_1, i, 5)$
$\ldots$

Program Correctness: User Input

assume $x \geq 5$
input x
$z \leftarrow x + y^2$
@ $z \geq 0$

$$\forall x, y, z \,.\, [x \geq 5 \wedge z = x + y^2] \Rightarrow z \geq 0$$

???

Program Correctness: User Input

Original program:

assume $x \geq 5$
input x
$z \leftarrow x + y^2$
@ $z \geq 0$

SSA:

assume $x \geq 5$
input x_1
$z \leftarrow x_1 + y^2$
@ $z \geq 0$

$$\forall x, x_1, y, z \,.\, [x \geq 5 \wedge z = x_1 + y^2] \Rightarrow z \geq 0$$

Correctness of Programs without Control Structures

Basic path: program of the form

c_1
$\dots$
c_n
@ α

where

- $c_1, \dots, c_n$ neither contains control structures nor @
- α is a logical formula,

Here: assignments, **assume**, **input**

Static Single Assignment Form

Whenever a variable is assigned a new value (e.g., **input**), rename variable to new one:

Example:

assume $x \geq 0$	**assume** $x_1 \geq 0$
input x	**input** x_2
assume $2x - 1 \geq 3$	**assume** $2x_2 - 1 \geq 3$
$x \leftarrow x - 2$	$x_3 \leftarrow x_2 - 2$
assume $x \geq 0$	**assume** $x_3 \geq 0$
input x	**input** x_4
assume $\neg 2x - 1 \geq 3$	**assume** $\neg 2x_4 - 1 \geq 3$
$x \leftarrow x - 1$	$x_5 \leftarrow x_4 - 1$
@ $x \neq 0$	@ $x_5 \neq 0$

Notation: SSA(P)

First-Order Formula from SSA

Example:
assume $x_1 \geq 0$; **input** x_2; **assume** $2x_2 - 1 \geq 3$; $x_3 \leftarrow x_2 - 2$;
assume $x_3 \geq 0$; **input** x_4; **assume** $\neg 2x_4 - 1 \geq 3$; $x_5 \leftarrow x_4 - 1$;

$$x_1 \geq 0 \wedge 2x_2 - 1 \geq 3 \wedge x_3 = x_2 - 2 \wedge x_3 \geq 0 \wedge 2x_4 - 1 < 3$$

Formally:
$F(\textbf{assume } \phi) := \phi$
$F(\textbf{input } v) := \top$
$F(v \leftarrow t) := v = t$

$$F(c_1; \ldots; c_n; @\ \alpha) := [\bigwedge_{i \in \{1,\ldots,n\}, F(c_i) \neq \top} F(c_i)] \Rightarrow \alpha$$

Correctness of Basic Paths

Resulting formula (*verification condition*) for a basic path P

$$VC(P) := F(SSA(P))$$

Then:

A basic path P is *correct* (as a program)

iff

$\models VC(P)$

Programs with Several Assertions @

$x \leftarrow 2$
$@\ x \geq 2$
$x \leftarrow 2x$
$@\ x \geq 4$

is correct if

$x \leftarrow 2$
$@\ x \geq 2$

and

assume $x \geq 2$
$x \leftarrow 2x$
$@\ x \geq 4$

are correct.

BUT:

$x \leftarrow 2$
$@\ 0 = 0$
$x \leftarrow 2x$
$@\ x \geq 4$

is correct and

assume $0 = 0$
$x \leftarrow 2x$
$@\ x \geq 4$

is not.

Programs with Several Assertions @

$\quad\ldots$
@ α_1
$\quad\ldots$
@ α_2
$\quad\ldots$
$\quad\ldots$
@ α_n

is correct if the following n basic paths are correct:

$\ldots$ @ α_1	**assume** α_1 $\ldots$ @ α_2	$\ldots$	**assume** α_{n-1} $\ldots$ @ α_n

To make them correct, strenghtening of $\alpha_1, \ldots, \alpha_{n-1}$ might be necessary.

Verification Conditions: Manual Proof

assume $x \leq -2$
$y \leftarrow x$
$x \leftarrow -2y + 1$
$z \leftarrow x + y$
`@` $z \geq 0$

$$\forall x, x_1, y, z \,.\, [x \leq -2 \wedge y = x \wedge x_1 = -2y + 1 \wedge z = x_1 + y] \Rightarrow z \geq 0$$

Assumptions: $x \leq -2$, $y = x$, $x_1 = -2y + 1$, $z = x_1 + y$
Prove $z \geq 0$
Using the assumption $z = x_1 + y$, this means to prove $x_1 + y \geq 0$
Using the assumption $x_1 = -2y + 1$, this means to prove $-2y + 1 + y \geq 0$
This means to prove $-y + 1 \geq 0$.
Using the assumption $y = x$, this means to prove $-x + 1 \geq 0$ that is $x \leq 1$.
This certainly holds under the assumption $x \leq -2$.

Counter-Examples

assume $x \leq 5$
$y \leftarrow x$
$x \leftarrow -2y + 1$
$z \leftarrow x + y$
@ $z \geq 0$

$$\forall x, x_1, y, z \,.\, [x \leq 5 \wedge y = x \wedge x_1 = -2y + 1 \wedge z = x_1 + y] \Rightarrow z \geq 0$$

From previous proof:
$x \leq 1$ implies that the final assertion holds (*weakest precondition*)

But the weaker assumption $x \leq 5$ does not imply this!

We get a counter-example by finding a solution to
$x \leq 5 \wedge \neg x \leq 1$, that is $x \leq 5 \wedge x > 1$.

A counter-example is, for example, 2.

Verification Conditions: Automated Proof

Solvers sometimes do not prove, but check satisfiability.

$$\forall x, x_1, y, z \,.\, [x \leq -2 \wedge y = x \wedge x_1 = -2y + 1 \wedge z = x_1 + y] \Rightarrow z \geq 0$$

is equivalent to

$$\neg \exists x, x_1, y, z \,.\, x \leq -2 \wedge y = x \wedge x_1 = -2y + 1 \wedge z = x_1 + y \wedge z < 0$$

Intuition: there is no bug

To prove this we show that the assumption

$$x \leq -2 \wedge y = x \wedge x_1 = -2y + 1 \wedge z = x_1 + y \wedge z < 0$$

leads to a contradiction.

This amounts to showing that this formula is not satisfiable.

If it is satisfiable, the satisfying assignment represents a bug.

demo (`sat.cvc`)

Verification Conditions: Alternative Form

$$x \leq -2 \wedge y = x \wedge x_1 = -2y + 1 \wedge z = x_1 + y \wedge z < 0$$

Original program

assume $x \leq 0$
$y \leftarrow x$
$x \leftarrow -2y + 1$
$z \leftarrow x + y$
@ $z \geq 0$

is correct

iff

assume $x \leq 0$
$y \leftarrow x$
$x \leftarrow -2y + 1$
$z \leftarrow x + y$
assume $z < 0$

cannot be executed.

Alternative definition of verification condition:
conjunction resulting from right-hand side program

two steps, hence less intuitive but simpler

Summary

For proving correctness of a program without control structure we need

1. view the program as being in SSA, and
2. check the corresponding verification condition.

Literature: many variants (not all of them use SSA)

Closest approach: [Kroening and Strichman, 2016, Section 12.2]

Literature I

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

Daniel Kroening and Ofer Strichman. *Decision Procedures*. Springer, 2nd edition, 2016.