Formal Methods and Specification (LS 2021)
Lecture 9: Termination, Total Program Correctness

Stefan Ratschan

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

Evropský sociální fond Praha & EU: Investujeme do vaší budoucnosti

Collatz Problem, Decidability

assume $x \geq 1$
while $x \neq 1$ **do**
    if $x \bmod 2 = 0$ **then**
        $x \leftarrow x/2$
    else
        $x \leftarrow 3x + 1$

Terminates for every input x?

Theorem [Turing, 1937]:
The problem of verifying termination (halting problem) is undecidable.

Today's Lecture

How to prove that a certain program terminates?

- Formal foundations
- Manual methods
- (Partially) automatic methods

At the same time we practice working with assertions

Basic Definitions

A partially correct program P *is terminating* iff
every regular execution of P
reaches a program line with the statement **return**.

We implicitely assume a
a **return** statement after the very last program line.

Attention: Termination does not imply that
there is an upper bound on the number of loop iterations

Example?

$i \leftarrow 0$
while $i < n$ **do**
$i \leftarrow i + 1$

Well-Founded Relations

A relation $\prec \subseteq \Omega \times \Omega$ is a *well-founded* on $S \subseteq \Omega$ iff
there is no infinite sequence $s_1 \in S, s_2 \in S, \ldots$ such that
$s_1 \succ s_2 \succ \ldots$

Example: $<$ is well founded on the natural numbers

but not on $\mathbb{R}^{\geq 0}$:

$$1 > \frac{1}{2} > \frac{1}{4} > \frac{1}{8} \ldots$$

Usually based on a strict partial order

Lexicographic Relation

Intuition: phone book

For given relations $\prec_1, \dots, \prec_n$,

$$(s_1, \dots s_n) \prec (t_1, \dots, t_n) \;:\Leftrightarrow\; \bigvee_{i=1}^{n} \left(\bigwedge_{j=1}^{i-1} s_j = t_j \wedge s_i \prec_i t_i \right)$$

Example: 8-bit representation of elements of $\{0, \dots, 255\}$,
$0 \prec_i 1$, $i \in \{1, \dots, 8\}$

$$(0,1,0,1,0,1,1,1) < (0,1,0,1,1,0,0,0)$$

If $\prec_1, \dots, \prec_n$ are well-founded relations on $S_1, \dots, S_n$, respectively
then $\prec$ is also a well-founded relation on $S_1 \times \dots \times S_n$.

Partial vs. Total Correctness

assume $x = 564$
while $\top$ **do**
  @ $x = 564$
@ $x = 564$
return x

The assertions hold on all executions, i.e., the program is *partially correct*.

We can prove that. The verification conditions are:

- $[x = 564 \land \top] \Rightarrow x = 564$ (twice)
- $[x = 564 \land \neg\top] \Rightarrow x = 564$ (twice)

But: The program is not terminating.

We want *total correctness* (=partial correctness + termination).

Example: Searching in a Sorted Array

For $a \in \mathcal{A}$, $n \in \mathcal{N}$, $\mathrm{sorted}(a, n) :\Leftrightarrow \forall i \in \{1, \ldots, n-1\} \,.\, a[i] \le a[i+1]$

assume $a \in \mathcal{A}$, $\mathrm{sorted}(a, n)$, $x \in \mathcal{N}$, $\exists i \in \{1, \ldots, n\} \,.\, a[i] = x$
$l \leftarrow 1$; $u \leftarrow n$
while $a[(l+u)/2] \ne x$ **do**
    @ $\exists i \in \{l, \ldots, u\} \,.\, a[i] = x$
    if $a[(l+u)/2] < x$ **then** $l \leftarrow (l+u)/2 + 1$
    else $u \leftarrow (l+u)/2 - 1$
@ $a[(l+u)/2] = x$
$r \leftarrow (l+u)/2$
@ $a[r] = x$
return r

Example: Searching in a Sorted Array

assume $a \in \mathcal{A}$, $\mathrm{sorted}(a, n)$, $x \in \mathcal{N}$, $\exists i \in \{1, \ldots, n\} \,.\, a[i] = x$
$l \leftarrow 1$; $u \leftarrow n$
while $a[(l+u)/2] \neq x$ **do**
    @ $\exists i \in \{l, \ldots, u\} \,.\, a[i] = x$
    if $a[(l+u)/2] < x$ **then** $l \leftarrow (l+u)/2 + 1$
    else $u \leftarrow (l+u)/2 - 1$
@ $a[(l+u)/2] = x$
$r \leftarrow (l+u)/2$
@ $a[r] = x$
return r

Basic path and corresponding verification condition:

assume $\exists i \in \{l, \ldots, u\} \,.\, a[i] = x$
assume $a[(l+u)/2] < x$;
$l \leftarrow (l+u)/2 + 1$
assume $a[(l+u)/2] \neq x$
@ $\exists i \in \{l, \ldots, u\} \,.\, a[i] = x$

$$[\exists i \in \{l, \ldots, u\} \,.\, a[i] = x \wedge a[(l+u)/2] < x \wedge l_1 = (l+u)/2 + 1 \wedge \ldots] \Rightarrow \exists i \in \{l_1, \ldots, u\} \,.\, a[i] = x$$

Verification Condition

$$[\exists i \in \{l, \dots, u\} \,.\, a[i] = x \wedge a[(l+u)/2] < x \wedge l_1 = (l+u)/2 + 1 \wedge \dots] \Rightarrow \exists i \in \{l_1, \dots, u\} \,.\, a[i] = x$$

Holds?

No, only under assumption $\mathrm{sorted}(a, n)$

Program Termination

assume $a \in \mathcal{A}$, $n \in \mathcal{N}$, $\mathrm{sorted}(a, n)$, $x \in \mathcal{N}$, $\exists i \in \{1, \ldots, n\} \,.\, a[i] = x$
$l \leftarrow 1$; $u \leftarrow n$
@ $\exists i \in \{l, \ldots, u\} \,.\, a[i] = x$
while $a[(l+u)/2] \neq x$ **do**
    @ $\exists i \in \{l, \ldots, u\} \,.\, a[i] = x$
    if $a[(l+u)/2] < x$ **then** $l \leftarrow (l+u)/2 + 1$
    else $u \leftarrow (l+u)/2 - 1$
@ $a[(l+u)/2] = x$
$r \leftarrow (l+u)/2$
@ $a[r] = x$
return r

Observation:

For ensuring termination it suffices to ensure termination of each loop.

Loop Termination

while $a[(l+u)/2] \neq x$ **do**

$\mathring{@}$ $u - l < v$

@ $u - l \geq 1 \wedge \exists i \in \{l, \ldots, u\} \,.\, a[i] = x$

$v \leftarrow u - l$

if $a[(l+u)/2] < x$ **then** $l \leftarrow (l+u)/2 + 1$

else $u \leftarrow (l+u)/2 - 1$

How to ensure that the loop terminates?

$u - l$ decreases, but always $u - l \geq 1$,

$<$ is well founded on the natural numbers

Intuition of $\mathring{@}$: only required upon re-entrance of loop

Additional Basic Paths

while $a[(l+u)/2] \neq x$ **do**
  $\mathring{@}\ u - l < v$
  @ $u - l \geq 1 \wedge \exists i \in \{l, \ldots, u\} \,.\, a[i] = x$
  $v \leftarrow u - l$
  if $a[(l+u)/2] < x$ **then** $l \leftarrow (l+u)/2 + 1$
  else $u \leftarrow (l+u)/2 - 1$

For the first branch of **if-then-else**:
assume $u - l \geq 1 \wedge \exists i \in \{l, \ldots, u\} \,.\, a[i] = x$
$v \leftarrow u - l$
assume $a[(l+u)/2] < x$
$l \leftarrow (l+u)/2 + 1$
assume $a[(l+u)/2] \neq x$ **do**
$\mathring{@}\ u - l < v$

Verification condition (ignoring irrelevant assumes):

$$[u - l \geq 1 \wedge v = u - l \wedge l_1 = (l+u)/2 + 1] \Rightarrow u - l_1 < v$$

Proof of Verification Condition

$$[u - l \geq 1 \wedge v = u - l \wedge l_1 = \frac{l+u}{2} + 1] \Rightarrow u - l_1 < v$$

Attention: integer division, may have reminder!

Assume $u - l \geq 1$, $v = u - l$, $l_1 = \frac{l+u}{2} + 1$ and prove $u - l_1 < v$.

If suffices to prove $u - \frac{l+u}{2} + 1 < u - l$.

It suffices to prove that lower bound increases:

$$\frac{l+u}{2} + 1 > l$$

$\frac{l+u}{2} + 1$ as a function in u monotonically increasing.

Proof for $u = l + 1$ suffices: $\frac{l+u}{2} + 1 = \frac{2l+1}{2} + 1 = l + 1 > l$

The basic path and proof of the second branch is similar.

Changed Basic Paths

while $a[(l+u)/2] \neq x$ **do**

$\quad$ @̊ $u - l < v$

$\quad$ @ $u - l \geq 1 \land \exists i \in \{l, \ldots, u\} \,.\, a[i] = x$

$\quad$ $v \leftarrow u - l$

$\quad$ **if** $a[(l+u)/2] < x$ **then** $l \leftarrow (l+u)/2 + 1$

$\quad$ **else** $u \leftarrow (l+u)/2 - 1$

For the first branch of **if-then-else** (second branch analogical):

assume $u - l \geq 1 \land \exists i \in \{l, \ldots, u\} \,.\, a[i] = x$

$v \leftarrow u - l$

assume $a[(l+u)/2] < x$

$l \leftarrow (l+u)/2 + 1$

assume $a[(l+u)/2] \neq x$

@ $u - l \geq 1 \land \exists i \in \{l, \ldots, u\} \,.\, a[i] = x$

Proof outline:

- If $u - l = 1$, then the last assume fails (i.e., the loop terminates).
- If $u - l = 2$, then, at the final assertion $u - l = 1$.
- If $u - l > 2$, then the distance will be larger than in the previous case.

Terminology and General Termination Proofs

For proving program termination, for each loop

- a term t over program variables denoting a function to a set S, and
- a well-founded relation $\prec$ on S s.t.

the value of t decreases for each execution of the loop wrt. $\prec$.

Such a term is called *loop variant* (also *ranking function*)

Corresponding assertions:

while ... **do**
    $\mathring{@}$ $t \prec v$
    @ $t \in S$
    $v \leftarrow t$
    ...

where v is a new, auxiliary variables (for each loop a different one).

Basic Paths of Termination Conditions: Summary

while $a[(l+u)/2] \neq x$ **do**
  $\mathring{@}\ u - l < v$
  $@\ u - l \geq 1 \wedge \exists i \in \{l, \ldots, u\} \,.\, a[i] = x$
  $v \leftarrow u - l$
  if $a[(l+u)/2] < x$ **then** $l \leftarrow (l+u)/2 + 1$
  else $u \leftarrow (l+u)/2 - 1$

Basic paths resulting from termination conditions:

- basic paths without $\mathring{@}$, constructed as usual, ignoring $\mathring{@}$
- additional basic paths leading from final assertion in loop to $\mathring{@}$

Relationship to Temporal Logic

Formulation of termination in temporal logic LTL:

$\mathbf{F}\ pc = r$,

where r is the program line with the command **return**.

Variant: similar notion for continuous systems: *Lyapunov function*

Nested Loops

$$\mathrm{substr}(a, s, p, l) :\Leftrightarrow \forall i \in \{1, \dots, l\} \,.\, a[p + i - 1] = s[i]$$

Input: $a \in \mathcal{A}, n \in \mathcal{N}, s \in \mathcal{A}, r \in \mathcal{N}$

Output: p, s.t. $\mathrm{substr}(a, s, p, r)$, if such a p, exists, and 0, otherwise.

Nested Loops

@ $a \in \mathcal{A}, n \in \mathcal{N}, s \in \mathcal{A}, r \in \mathcal{N}$
$i \leftarrow 0$; $l \leftarrow 0$
while $i + r \leq n \wedge \neg found$ **do**
    $\mathring{@}$ $i > w_1$
    @ $i \leq n - r \wedge \forall p \in \{1, \ldots, i\} \,.\, \neg \mathrm{substr}(a, s, p, r)$
    $w_1 \leftarrow i$; $i \leftarrow i + 1$; $l \leftarrow 1$
    while $l \leq r \wedge a[i + l - 1] = s[l]$ **do**
        $\mathring{@}$ $l > w_2$
        @ $l \leq r \wedge \mathrm{substr}(a, s, i, l)$
        $w_2 \leftarrow l$; $l \leftarrow l + 1$
if *found* **then**
    @ $\mathrm{substr}(a, s, i, r)$
    return i
else
    @ $\forall p \in \{1, \ldots, n - r + 1\} \,.\, \neg \mathrm{substr}(a, s, p, r)$
    return 0

Relationship to Program Complexity

If we have a variant for the relation $<$ on natural numbers
we cannot execute the corresponding loop more often than
the initial value of the variant.

So: upper bound on the run-time complexity of an algorithm

Completeness of the method:
Does there exist a variant for every terminating algorithm?

Yes, number of steps before termination.

Problem: How to write this as a simple term?

For algorithms whose termination is not known,
we do not know whether this number is finite.

Termination of Recursion

GCD(x, y)
@ $x, y \in \mathcal{N}, x \geq y$
$v \leftarrow y$ // variant y
if $y = 0$ **then**
    $r \leftarrow x$
else
    @ $y[x \leftarrow y, y \leftarrow x \bmod y] < v \wedge y[x \leftarrow y, y \leftarrow x \bmod y] \geq 0$
    $r \leftarrow \mathtt{GCD}(y, x \bmod y)$
return r

well-founded relation? variant? $<$ on $\{0, 1, \ldots\}$, y

new verification condition (i.e., extension of old condition):
$[x \geq y \wedge v = y \wedge y \neq 0] \Rightarrow [x \bmod y < v \wedge x \bmod y \geq 0]$

Automatization

Theorem: [Turing, 1937]:
The problem of verifying termination (halting problem) is undecidable.

If we have a variant,
  the corresponding verification conditions
    can often be proved automatically (e.g., using CVC4)

Can we find variants automatically?

T2 Temporal Prover [Cook et al., 2011]

Rest of lecture: an underlying technique

Automatic Synthesis of Variants

while $10 \leq x \land x \leq 20 \land 10 \leq y \land y \leq 20$ **do**

  @̊ $ax + by \leq v - 1$

  @ $ax + by \geq 0$

  $v \leftarrow ax + by$

  $x \leftarrow x - y$

  $y \leftarrow x + 2y$

Variant?

Assumption: a linear function

But we do not know which one,
  that is, we do not know the coefficients of the function.

Variant template: $ax + by$

Well-founded relation $\prec$: $\mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0}$,
  $u \prec v$ iff $u \leq v - 1$.

Automatic Synthesis of Variants: Verification Conditions

while $10 \leq x \wedge x \leq 20 \wedge 10 \leq y \wedge y \leq 20$ **do**

$\mathring{@}\ ax + by \leq v - 1$

$@\ ax + by \geq 0$

$v \leftarrow ax + by$

$x \leftarrow x - y$

$y \leftarrow x + 2y$

Verification Conditions:

- $[10 \leq x \wedge x \leq 20 \wedge 10 \leq y \wedge y \leq 20] \Rightarrow ax + by \geq 0$
- $\left[\begin{array}{l} ax + by \geq 0 \wedge \\ 10 \leq x - y \wedge x - y \leq 20 \wedge \\ 10 \leq x + 2y \wedge x + 2y \leq 20 \wedge \\ v = ax + by \wedge x_1 = x - y \wedge y_1 = x + 2y \end{array}\right] \Rightarrow ax_1 + by_1 \geq 0$
- $\left[\begin{array}{l} ax + by \geq 0 \wedge \\ 10 \leq x - y \wedge x - y \leq 20 \wedge \\ 10 \leq x + 2y \wedge x + 2y \leq 20 \wedge \\ v = ax + by \wedge x_1 = x - y \wedge y_1 = x + 2y \end{array}\right] \Rightarrow ax_1 + by_1 \leq v - 1$

Solving the Verification Conditions

- $[10 \leq x \wedge x \leq 20 \wedge 10 \leq y \wedge y \leq 20] \Rightarrow ax + by \geq 0$
- $\left[\begin{array}{l} ax + by \geq 0 \wedge \\ 10 \leq x - y \wedge x - y \leq 20 \wedge \\ 10 \leq x + 2y \wedge x + 2y \leq 20 \wedge \\ v = ax + by \wedge x_1 = x - y \wedge y_1 = x + 2y \end{array}\right] \Rightarrow ax_1 + by_1 \geq 0$
- $\left[\begin{array}{l} ax + by \geq 0 \wedge \\ 10 \leq x - y \wedge x - y \leq 20 \wedge \\ 10 \leq x + 2y \wedge x + 2y \leq 20 \wedge \\ v = ax + by \wedge x_1 = x - y \wedge y_1 = x + 2y \end{array}\right] \Rightarrow ax_1 + by_1 \leq v - 1$

We have to find a, b, such that for all x, y, v, x_1, y_1 the conditions hold.

Purely algebraic problem, Redlog demo

If the conditions do not have a solution, what does this imply?

This does not imply non-termination.
(we only know that there is not variant of the given form)

We may use a different one, for example,
by increasing the degree of the polynomial: $ax^2 + bcx + cy^2 + ex + fy$

Conclusion

- Program termination can be proved manually.
- Termination proofs can be partially automatized.
- However, in general, undecidable

Literature I

Byron Cook, Andreas Podelski, and Andrey Rybalchenko. Proving program termination. *Commun. ACM*, 54(5):88–98, May 2011. doi: 10.1145/1941487.1941509.

A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937.