Principles of program termination

Notes for the Marktoberdorf summer school

Byron Cook
Microsoft Research Cambridge

1. Introduction

The *program termination problem*, also known as the *uniform halting problem*, can be defined (using today's terms) as follows:

> Using a finite amount of time: determine whether a given program will always finish running or could potentially execute forever.

This problem rose to prominence before the invention of programs or computers, in the era of Hilbert's *Entscheidungsproblem*[1]: the challenge to formalize all of mathematics into logic and use mechanical means to determine the validity of mathematical statements. After Hilbert's challenge, a number of logicians and mathematicians began finding instances of undecidable problems, thus showing the ideal of the Entscheidungsproblem to be impossible. Turing [18][2], for example, proved the halting problem undecidable.

Turing's result is now perhaps the most frequently covered topic in introductory courses in the areas of logic and theoretical computer science. This popularization of Turing's result has, unfortunately, had the side-effect of giving birth to a frequently held misconception that we are *always unable to prove termination*: many believe that it is impossible to prove program termination of *any* program; many others believe that program termination is too hard a problem to tackle. Thus, little effort was expended to automate termination proving—those who suggest solutions to termination or related problems were usually derided by their peers.

The true consequence of Turing's proof, in contrast, is the much more benign fact that we are *unable to always* prove termination—meaning that we can potentially prove termination in most cases, but no matter how sophisticated a termination prover we build, there will always be at least one terminating program that cannot be proved terminating. The program termination problem has recently found new life [16] as our society increasingly depends more on computers, the need for practical tools that automatically *prove the correctness of software* (as

[1] In English: "decision problem"

[2] There is some controversy as to whether or not Turing proved the undecidability in [18]. Technically he did not, but termination's undecidability is an easy consequence of the result that is proved.

opposed to simply finding bugs) is becoming clear [1]. Practical and powerful industrial tools are now emerging that allow us to express and automatically prove properties of computer programs. It turns out that the program termination problem is at the foundation of many of the properties that we might want to prove of software: a *liveness property* such as "Every call to AcquireLock is eventually followed by a call to ReleaseLock" amounts proving the termination of the code occurring between calls to AcquireLock and ReleaseLock. Several advanced prototype tools have recently emerged that attempt to automatically prove program termination. Since termination is formally undecidable, the modern challenge is to show the problem to be "effectively decidable"—*i.e.* to build robust termination provers that work in all practical cases, or maybe even all known cases.

As of the writing of these notes, these new termination tools can automatically prove or disprove termination of many famous complex examples (*e.g.* Ackermann's function, McCarthy's 91 function), as well as moderately-sized industrial examples (*e.g.* Windows OS device drivers). Perhaps, as the tools improve, we will one day be able prove liveness properties of most industrial programs.

Currently the techniques for automating program termination proofs are known perhaps to only a handful of people. Furthermore, there is no single good source for those interested in learning more: knowledge on how to automate termination proofs must today be synthesized from obscure research papers in tandem with voluminous foundational texts (each of which used distinct notations and required different levels of sophistication).

These notes are designed to accompany lectures I will present at the Marktoberdorf summer school. My goal in giving these lectures is to provide advanced students with an understandable and uniform introduction to the foundations of the program termination problem and the modern approaches for automation. The lectures will begin with information drawn from this paper. The material in the remaining lectures will be drawn primarily from the research papers written during the development of the TERMINATOR termination prover: [2,3,5,6,7,8,9,13].

2. Defining termination

For the purpose of these notes it is convenient to think of a computer program as its possible initial configurations paired together with a relation that specifies the possible transitions that the program can make between configurations during execution. Program executions can be thought of as traversals starting from one of the program's initial configurations through the various changes of configuration allowed by the program's transition relation. We say a program is *terminating* if all of its executions are finite. A program is called *non-terminating* if there exists at least one infinite execution.

When trying to prove termination, formally we are trying to prove that the program's transition relation is *well-founded.* In a sense, termination is the user's experience, whereas well-foundedness is a mathematical property the holds of transition relations of terminating programs. Despite their differences, we shall use the two terms interchangeably throughout these notes and lectures.

Before attempting to automate the search for proofs of termination (as happens in the advanced literature) we first must ground ourselves with some basic concepts and notation—*e.g. states*, *programs*, *ranking functions* and *well-founded relations*.

3. States, sets and relations

Throughout these notes we will be concentrating on program configurations. We call these configurations *states*. States are encoded as partial finite mappings from variables to values. Let $\mathcal{S}$ be the set of all such mappings. We will assume that the set of variables, $\textsc{Var}$, is infinite, and formed of strings expressed in sans-serif font (*e.g.* $\mathsf{x} \in \textsc{Var}$). The set of values, $\textsc{Val}$, will be an under-specified set of values which can include $\mathbb{Z}$ or other arithmetic constants. Later we will point to papers which describe termination methods supporting programs with dynamically allocated heap-based data structures.

Depending on the context we will use two forms of notation interchangeably when describing relations over states: sets of pairs of states, and formulae drawn from quantifier-free first-order logic with pre- and post-variables. When using formulae, unprimed variables will represent the pre-variables and primed variables will represent the post-variables. We define the usual semantic mapping from formulae to the underlying sets of pairs of states that they represent, *e.g.* $[\![\mathsf{x} < \mathsf{x}']\!] = \{(s,t) \mid s(\mathsf{x}) < t(\mathsf{x})\}$. If Q is a set of states using unprimed variables, we use the notation Q' to mean a set expressed using primed variables such that $Q \cong Q'$. Note that many of the definitions and results described in these notes hold over all sets and relations, not just those over states.

Definition 1 (Relational application, composition, closure) Assume that $R \subseteq S \times T$ and $I \subseteq S$, we define the image of R on I (notationally, $R(I)$) as:

$$R(I) \triangleq \{b \mid a \in I \wedge (a,b) \in R\}$$

Note that $R(I) \subseteq T$. If $a \in S$ and $R \subseteq S \times T$ then we define $R(a) \subseteq T$ as

$$R(a) \triangleq \{b \mid (a,b) \in R\}$$

Let ; be relational composition where

$$R;Q \triangleq \{(a,b) \mid \exists c.\ (a,c) \in R \wedge (c,b) \in Q\}$$

We define $R^0 \triangleq \{(a,b) \mid a = b\}$. When $k > 0$, $R^k \triangleq R;R^{k-1}$. The non-reflexive and reflexive transitive closure of R are defined respectively:

$$\begin{array}{rcl} R^+ & \triangleq & \{(a,b) \mid \exists n > 0.\ (a,b) \in R^n\} \\ R^* & \triangleq & \{(a,b) \mid \exists n \geq 0.\ (a,b) \in R^n\} \end{array}$$

We define relational inverse and projection as follows:

$$\begin{array}{rcl} R^{-1} & \triangleq & \{(a,b) \mid (b,a) \in R\} \\ \Pi_1(R) & \triangleq & \{a \mid \exists b.\ (a,b) \in R\} \\ \Pi_2(R) & \triangleq & \{b \mid \exists a.\ (a,b) \in R\} \end{array}$$

If Q is a set of states (*i.e.* $Q \subseteq \mathcal{S}$) then $\neg Q \triangleq \mathcal{S} - Q$

4. Well-ordered sets and well-founded relations

In this section we describe what it means for a set to be *well ordered*, and a relation to be *well founded*.

Definition 2 (Total-order) The structure $(S, \geq)$ forms a *total order* iff for all $a, b, c \in S$

- $a \geq a$ (reflexive),
- $a \leq b$ and $a \geq b$ then $a = b$ (antisymmetry),
- If $a \geq b$ and $b \geq c$ then $a \geq c$ (transitivity),
- $a \leq b$ or $a \geq b$ (totality),

Definition 3 (Well order) $(S, \geq)$ forms a *well order* iff it is a total order and every nonempty subset of S has a least element.

Example 1 The natural numbers, $\mathbb{N}$, are a well-ordered set, as in the worst case 0 is the least element of any subset. The integers, $\mathbb{Z}$, are not well ordered because there is no least element. However, for any integer constant $b \in \mathbb{Z}$, the set $\{x \mid x \in \mathbb{Z} \wedge x \geq b\}$ is a well-ordered set. $\star$

Example 2 The non-negative real numbers with relation $\geq$ are not a well-ordered set because there there is no least element in the open interval (0,1). The non-negative real numbers can be made into a well-ordered set when paired with the alternative comparison relation $\geq_w$, which we define $x \geq_w y \triangleq x \geq y + 1 \vee x = y$. $\star$

Definition 4 (Sequences) We say that s is an S-sequence if $s = s_1, s_2, \ldots$ and each $s_i \in S$. A finite sequence will have a last index $\mathsf{last}(s)$. Let $R \subseteq S \times S$. A finite sequence is said to be *permitted by R* iff $\forall i \in \{1, \ldots \mathsf{last}(s) - 1\}.\ R(s_i, s_{i+1})$. An infinite sequence is permitted by R iff $\forall i.\ i > 0 \Rightarrow R(s_i, s_{i+1})$. Let $I \subseteq S$. We say that s is permitted by (I, R) iff s is permitted by R and $s_1 \in I$.

Definition 5 (Well-founded relations) A binary relation $R \subseteq S \times S$ is well-founded iff it does not permit infinite sequences.

Example 3 The relation $\mathsf{x} > \mathsf{x}' \wedge \mathsf{x}' > 0$ is a well-founded relation if the variables range over the integers or natural numbers, but not if the variables range over the reals. The reason is that, if we apply the relation point-wise to any sequence of naturals or integers, we'll see that the values along the sequence are forced to go down towards (and eventually pass) a bound. Thus no permitted sequence can be infinite. In the reals the constraint $\mathsf{x} > \mathsf{x}'$ does not require the value to go down enough to guarantee eventual progress to 0. The relation $\mathsf{x} \geq \mathsf{x}' + 1 \wedge \mathsf{x}' > 0$, on the other hand, is well founded in all three interpretations. $\star$

Theorem 1 *Assume that $(S, \geq)$ is a total order. $(S, \geq)$ is a well order iff the relation $x > y$ (defined as $x > y \triangleq x \geq y \wedge x \neq y$) is well founded on S-sequences.*

Proof.

Well-ordered set ⇒ Well-founded relation: By a contrapositive argument, assume that $>$ is not well founded, meaning in this case that there is an infinitely descending chain of S-elements. In this case there can be no least element. ✓

Well-ordered set ⇐ Well-founded relation: Again, by a contrapositive argument. Assume that the infinite S-subset S' has no least element (the fact that every finite set has a least element can be established using the fact that S is a total order). Let $s_1 \in S'$. Since s_1 cannot be minimal we know that there exists an $s_2 \in S'$ such that $s_1 > s_2$, and an $s_3 \in S'$ such that $s_2 > s_3$, etc. Therefore, using the somewhat controversial axiom of dependent choice we can show that there exists an infinite sequence of S'-elements that is permitted by $>$ ✓

□

Observation 1 *If Q is well founded and $R \subseteq Q$, then R is well founded.*

Proof. Assume that R is not well founded. Therefore there exists an infinite sequence s such that $\forall i.(s_i, s_{i+1}) \in R$. Because $R \subseteq Q$, we know that $\forall i.(s_i, s_{i+1}) \in Q$, thus contradicting the claim that Q is well founded. □

Corollary 1 *If R is not well founded and $R \subseteq Q$, then Q is not well founded.*

Remark on Cantor's ordinal numbers. We often see Cantor's ordinal numbers [4] used in the literature discussing well-ordered sets. Cantor's ordinals are a canonical representation for sets of well-ordered sets who are all related in size. (*e.g.* the natural numbers and any isomorphic set can be represented by the ordinal number ω). In these notes we avoid the ordinals for the reason that, although they can make a fundamental discussion more concise, they come at a great initial cost. Many distracting ideas and notation would need to be introduced.

5. Ranking functions and ranking relations

The most popular method of proving a relation $R \subseteq S \times S$ well founded is to follow Turing's suggestion [19] and find a map from the structure (R, S) to some known well-ordered set $(\geq, T)$ and then prove that the map is structure-preserving (*i.e.* that it is a homomorphism). Since we know that the $>$ relation (where $x > y \triangleq x \geq y \wedge x \neq y$) on T is well founded, by the properties of homomorphisms and Observation 1, we know that R too is well founded. Turing's maps are typically called *ranking functions.*

Definition 6 (Ranking function) A mapping f with a range to a well-ordered set is called a ranking function. In cases where f ranges over a set that would be a well-order if a bounder were given, we may chose to make the bound explicit. In this case we say that (f, b) is a ranking function. In cases where the set would be a well-order if an explicit delta d were given, then we say (f, b, d) is the ranking function.

Definition 7 (Ranking relation) Let $f : X \to Y$ be a ranking function. We define f's ranking relation, $\trianglerighteq_f$, to be

$$\trianglerighteq_f = \{(s,t) \mid f(s) > f(t)\}$$

We also introduce variants of $\trianglerighteq$ for the case where an explicit bound is needed

$$\trianglerighteq_{f,b} = \{(s,t) \mid f(s) > f(t) \wedge f(s) \geq b\}$$

and where an explicit delta is needed

$$\trianglerighteq_{f,b,d} = \{(s,t) \mid f(s) \geq f(t) + d \wedge f(s) \geq b\}$$

Observation 2 *For any ranking function f, $\trianglerighteq_f$ is well founded. Analogously, for any ranking function (f, b), $\trianglerighteq_{f,b}$ is well founded.*

Proof. We know that there exists some Y such that $f : X \to Y$ such that $(\geq, Y)$ is a well-ordered set. Thus, due to Theorem 1, we know that $>$ is a well-founded relation on sequences drawn from Y. By way of contradiction, assume that $s_1, s_2, s_3, \ldots$ is an infinite sequence permitted by $\trianglerighteq_f$. This gives rise to the infinite sequence of Y-elements $f(s_1) > f(s_2) > f(s_3) > \ldots$. But this infinite sequence is not permitted, as $(\geq, Y)$ is well ordered. □

Example 4 Consider the example relation

$$R \triangleq \mathsf{x} > 0 \wedge \mathsf{y} > 0 \wedge \mathsf{x}' = \mathsf{x} - 1 \wedge \mathsf{y}' = \mathsf{y} + 1$$

Assume that x and y range over the integers. To prove R well-founded we can use the ranking function $f(s) = s(\mathsf{x})$ and bound 0 to construct $\trianglerighteq_{f,0}$:

$$\begin{array}{lcl} \trianglerighteq_{f,0} & = & \{(s,t) \mid f(s) > f(t) \wedge f(s) \geq 0\} \\ & = & \{(s,t) \mid s(\mathsf{x}) > t(\mathsf{x}) \wedge s(\mathsf{x}) \geq 0\} \\ & = & [\![\mathsf{x} > \mathsf{x}' \wedge \mathsf{x} \geq 0]\!] \end{array}$$

To prove that the inclusion $R \subseteq \trianglerighteq_{f,0}$ holds we can construct a query for a decision procedure. See Figure 1 for an implementation expressed in F# using an interface to the Z3 decision procedure tool. When executed this program prints the result true.

⋆

6. Supporting invariants

The common wisdom when proving a relation well founded is that one must find both a ranking function *and a supporting invariant.* The difficulty that this strategy is solving is the fact that, in practice, relations are often only well founded when restricted to the states reachable by the relation from some set of initial states.

```
let n0 = Dp.constant 0
let n1 = Dp.constant 1
let x = Dp.var "x"
let y = Dp.var "y"
let x' = Dp.var "x'"
let y' = Dp.var "y'"
let R = Dp.conj [ Dp.gt x n0 ; Dp.gt y n0
                ; Dp.eq x' (Dp.sub x n1)
                ; Dp.eq y' (Dp.add y n1)
                ]
let f = x
let f' = x'
let RR = Dp.conj [ Dp.gt f f' ; Dp.ge f n0 ]
let query = Dp.implies R RR
Dp.valid query |> print_bool
```

Figure 1. F# code which proves the condition from Example 4

Definition 8 (Transition systems) We say that P is a *transition system* if $P = (I, R, S)$, where S is the (possibly infinite) set of program states represented as finite partial functions from $\textsc{Vars}$ to $\textsc{Vals}$, $I \subseteq S$, and $R \subseteq S \times S$. We call I the *initial states*, and R the *update relation.*

Definition 9 (Reachable states) We call $R^*(I)$ the *reachable states* of the transition system $P = (I, R, S)$.

Definition 10 (Transition relation) Let $P = (I, R, S)$. We use the notation $R\underset{*}{\downharpoonright}_I$ to denote P's *transition relation*:

$$R\underset{*}{\downharpoonright}_I \triangleq R \cap (R^*(I) \times R^*(I))$$

In contrast to transition relations, in practice update relations are usually simple disjunctions representing simple commands—usually update relations are much larger than $R\underset{*}{\downharpoonright}_I$, though of course it is possible to define a R such that $R = R\underset{*}{\downharpoonright}_I$.

Definition 11 (Invariant) A set of states Q is an invariant of a relation $R \subseteq S \times S$ and initial set $I \subseteq S$ iff $Q \supseteq R^*(I)$.

Note that, because $R\underset{*}{\downharpoonright}_I \subseteq R$, if R is well founded then $R\underset{*}{\downharpoonright}_I$ is also well founded. Clearly $R^*(I)$ is the strongest possible invariant, but it is not computable in theory and very difficult to compute in practice. Instead we usually look for a weaker (but easier to find) invariant Q that is strong enough to prove relations well founded. Because, by definition, $Q \supseteq R^*(I)$, if $Q \times Q \cap R$ is well founded, then we know that $R\underset{*}{\downharpoonright}_I$ is well founded. Note also that $Q \times Q \cap R = Q \times T \cap R$ whenever $Q \subseteq T$. Thus it suffices to find an invariant and prove $Q \times \mathcal{S} \cap R$ well founded, assuming that $R \subseteq \mathcal{S} \times \mathcal{S}$.

Example 5 Consider the relation $R \triangleq \mathsf{x} > 0 \wedge \mathsf{x}' = \mathsf{x} + \mathsf{y} \wedge \mathsf{y}' = \mathsf{y}$, where the variables range over the integers. R is not well founded if $\mathsf{y} \geq 0$. However, if we

```
let n0 = Dp.constant 0
let x = Dp.var "x"
let x’ = Dp.var "x’"
let y = Dp.var "y"
let y’ = Dp.var "y’"
let R = Dp.conj [ Dp.gt x n0
                ; Dp.eq x’ (Dp.add x y)
                ; Dp.eq y’ y
                ]
in
let I = Dp.lt y n0

// The relation Q * Q is expressed via Q && Q’, where Q’ is like
// Q but expressed over primed variables
let Q = Dp.lt y n0
let Q’ = Dp.lt y’ n0

// Base check
Dp.implies I Q |> Dp.valid |> print_bool

// Inductive check
Dp.implies (Dp.conj [Q;R]) Q’ |> Dp.valid |> print_bool

// WF-check
let RR = Dp.conj [ Dp.gt x x’ ; Dp.ge x n0 ]
Dp.implies (Dp.conj [R;Q;Q’]) RR |> Dp.valid |> print_bool
```

Figure 2. Implementation of the check described in Example 5.

let the initial set of states be $I \triangleq \mathsf{y} \leq -1$, then $R{\underset{*}{\rfloor}}_I$ is well founded. To prove this we can let $Q = \mathsf{y} \leq -1$. Luckily, in this case, Q is an *inductive invariant*, meaning that we can show Q invariant simply via induction (*i.e.* $I \Rightarrow Q$ and $Q \wedge R \Rightarrow Q'$) To prove that $Q \times Q \cap R$ is well founded we can show that $Q \times Q \cap R \subseteq \trianglerighteq_{\mathsf{x},0}$. This query is encoded in Figure 2.

$\star$

7. Proving non-termination

Until now we have considered only proving termination, but not disproving—*i.e.* proving non-termination.

Definition 12 (Recurrence sets) Assume $P = (I, R, S)$. $Q \subseteq \mathcal{S}$ is a *recurrence set* of P if:

1. $Q \subseteq \Pi_1(R)$
2. $Q \cap I \neq \emptyset$
3. $\forall x \in Q.\ \exists x'.\ (x, x') \in R \wedge x' \in Q$

Theorem 2 *$R{\underset{*}{\rfloor}}_I$ is not well founded iff there exists a recurrence set Q for $R{\underset{*}{\rfloor}}_I$*

Example 6 Consider the relation over $\mathcal{S}$:

$$R \triangleq \mathsf{x} > 0 \wedge (\mathsf{x}' = \mathsf{x} - 1 \vee \mathsf{x}' = \mathsf{x})$$

Let $I \triangleq \mathcal{S}$. The relation $R\underset{*}{|}_I$ is not well founded. To *prove* it not well founded (as opposed to simply failing to prove it well founded) we define $Q = \{s \mid s(\mathsf{x}) = 1 \wedge s \in \mathcal{S}\}$. $\Pi_1(R) = \{s \mid s(\mathsf{x}) > 0 \wedge s \in \mathcal{S}\}$, thus $Q \subseteq \Pi_1(R)$. Because $Q \subseteq I$ and $Q \neq \emptyset$, $Q \cap I \neq \emptyset$. Finally, $R(Q) = Q$, thus $\forall x \in Q. \exists x'.(x, x') \in R \wedge x' \in Q$.

$\star$

8. Composing termination arguments

In many cases, constructing a ranking function for a complex relation can be a subtle art. As we have seen: *once we know* a ranking function, proving the necessary subset inclusion is usually not difficult—finding the ranking function argument is the hard part. This section describes a method for constructing termination arguments via the composition of small sub-arguments. As we will see later, the method makes the search for and construction of termination arguments easier, but makes checking the argument more difficult. Modern approaches to termination are based on this result.

Theorem 3 (Podelski & Rybalchenko) *Let be a binary relation $R \subseteq S \times S$. Let $Q_1, Q_2, \ldots Q_n$ be a finite set of binary relations $Q_i \subseteq S \times S$ such that each Q_i is well founded. R is well founded iff $R^+ \subseteq Q_1 \cup Q_2 \cup \ldots \cup Q_n$.*

The proof of this theorem can be found in [14].

It is important to note that the union of well-founded relations is not necessarily well founded, thus making Theorem 3 a little surprising. Transitive closure is the key to Theorem 3's soundness. To see why this is true consider the relations $P \triangleq 0 < \mathsf{x}' \wedge \mathsf{x}' < \mathsf{x}$ and $Q \triangleq 100 > \mathsf{x}' \wedge \mathsf{x}' > \mathsf{x}$. Both P and Q are well founded, but $P \cup Q$ is not. To see that $P \cup Q$ is not well founded consider the case where $s(\mathsf{x}) = 5$. In this case $(s, s) \in (P \cup Q)^2$, thus making $\{s \mid s(\mathsf{x}) = 5 \wedge s \in \mathcal{S}\}$ a valid recurrence set for $(P \cup Q)^2$.

Example 7 Consider the relation

$$\begin{array}{rcl} R & \triangleq & (\mathsf{x} > 0 \wedge \mathsf{y} > 0 \wedge \mathsf{x}' = \mathsf{x} - 1 \wedge \mathsf{y}' = \mathsf{y}) \\ & \vee & (\mathsf{x} > 0 \wedge \mathsf{y} > 0 \wedge \mathsf{x}' = \mathsf{x} \wedge \mathsf{y}' = \mathsf{y} - 1) \end{array}$$

We can prove R well founded by showing $R \subseteq \trianglerighteq_{\mathsf{x+y},0}$. Alternatively we use Theorem 3 and establish termination via proof that $R^+ \subseteq \trianglerighteq_{\mathsf{x},0} \cup \trianglerighteq_{\mathsf{y},0}$. Note that we cannot prove the inclusion $R^+ \subseteq \trianglerighteq_{\mathsf{x},0} \cup \trianglerighteq_{\mathsf{y},0}$ directly with any known decision procedure, as they do not support transitive closure (transitive closure for infinite-state systems is undecidable in theory, and difficult in practice). In the advanced research literature we see the use of techniques from program analysis being adapted to address this class of question. For now, define R^+_α to be

```
let n0 = Dp.constant 0
let x = Dp.var "x"
let x' = Dp.var "x'"
let y = Dp.var "y"
let y' = Dp.var "y'"
let R_star_abs =
    Dp.conj [ Dp.gt x n0
            ; Dp.gt y n0
            ; Dp.disj [ Dp.conj [Dp.gt x x'; Dp.ge y y' ]
                      ; Dp.conj [Dp.ge x x'; Dp.gt y y' ]
                      ]
            ]
let RR_x = Dp.conj [ Dp.gt x x' ; Dp.ge x n0 ]
let RR_y = Dp.conj [ Dp.gt y y' ; Dp.ge y n0 ]
let arg = Dp.disj [ RR_x ; RR_y]
Dp.implies R_star_abs arg |> Dp.valid |> print_bool
```

Figure 3. Implementation of the check described in Example 7.

$$R^+_\alpha \triangleq (\mathsf{x} > 0 \wedge \mathsf{y} > 0 \wedge \mathsf{x}' \leq \mathsf{x} \wedge \mathsf{y}' < \mathsf{y}) \vee (\mathsf{x} > 0 \wedge \mathsf{y} > 0 \wedge \mathsf{x}' < \mathsf{x} \wedge \mathsf{y}' \leq \mathsf{y})$$

It can be proved (via methods described later) that $R^+ \subseteq R^+_\alpha$. Thus we can use R^+_α to check the condition from Theorem 3. An encoding of this check can be found in Figure 3.

Note that finding $\trianglerighteq_{\mathsf{x},0}$ and $\trianglerighteq_{\mathsf{y},0}$ is, in a sense, easier than $\trianglerighteq_{\mathsf{x}+\mathsf{y},0}$. The reason is that if we can often find the former argument by looking individually at R's disjuncts: $\trianglerighteq_{\mathsf{x},0}$ is motivated by looking at the first disjunct in R (*i.e.* $\mathsf{x} > 0 \wedge \mathsf{y} > 0 \wedge \mathsf{x}' = \mathsf{x} - 1 \wedge \mathsf{y}' = \mathsf{y}$), and $\trianglerighteq_{\mathsf{y},0}$ is motivated by the second.

$\star$

As mentioned above, we find that Theorem 3 makes the construction of termination arguments easier but—because of the use of transitive closure—the checking of the inclusion harder. For more discussion on this topic, see [8]

Definition 13 (Termination arguments, validity) We say that M is a *valid termination argument* of R's iff $R^+ \subseteq M$ and M is disjunctively well founded or $R \subseteq M$ and M is well founded. We say that a valid recurrence set is a *valid argument for non-termination.*

9. Decomposition using program structure

When we consider programs with locations and nested loops the ranking function to prove termination for even the simplest program can be surprisingly complex if we attempt to directly find a ranking function using the relation that the code denotes. The difficulty is that the program's notation of location complicates matters.

A program's set of locations can actually work to our advantage if we use them appropriately. Following Floyd's suggestion [10], we define a technique that allows us decompose a single termination check into a fixed number of easier checks. The decomposition makes use of our assumption that the range of pc is finite in any state in reachable state. The decomposition technique is, in fact, general and can be used with any program variable of finite range.

Definition 14 Assume that Q is a set of states. We define the set $Q\underset{\circ}{\downarrow}_p$ as

$$Q\underset{\circ}{\downarrow}_p \triangleq Q \cap \{s \mid p(s)\}$$

We also define a similar restriction on relations:

$$R{\downarrow}_p \triangleq R \cap \{(s,t) \mid p(s) \wedge p(t)\}$$

Theorem 4 *Assume that $v \in \textsc{Var}$ and that the set L is finite, where*

$$L = \{x \mid s \in R^*(I) \wedge s(v) = x\}$$

$R\underset{*}{\downarrow}_I$ is well founded if for all $l \in L$, $(R\underset{*}{\downarrow}_I^+){\downarrow}_{v=l}$ is well founded.

Proof. By contradiction and the pigeon-hole principle. Assume that $s = s_1, s_2, s_3, \ldots$ is an infinite sequence such that $(s_1, s_2) \in R\underset{*}{\downarrow}_I$, $(s_2, s_3) \in R\underset{*}{\downarrow}_I$, etc. Because L is finite and $R^*(I)(v) = L$, we know that there exists a $c \in L$ such that $s_i(v) = c$ infinitely-often in s. Let s' be the infinite sequence of these states. We know that s' is in the sequences allowed by $R\underset{*}{\downarrow}_I^+ \cap \{(s,t) \mid s(v) = t(v) = c\}$ (*i.e.* $(R\underset{*}{\downarrow}_I^+){\downarrow}_{v=c}$. But $(R\underset{*}{\downarrow}_I^+){\downarrow}_{v=c}$ is well founded. □

Example 8 Consider the relation

$$\begin{array}{lll} R & \triangleq & (\mathsf{b}' = 1 \wedge \mathsf{b} = 0) \vee (\mathsf{b}' = 0 \wedge \mathsf{b} = 1) \\ & \wedge & (\mathsf{b} = 1 \wedge \mathsf{x}' = \mathsf{x} - 1 \wedge \mathsf{x} > 0) \vee (\mathsf{b} = 0 \wedge \mathsf{x}' = \mathsf{x}) \end{array}$$

In this case we could invent a fairly complex ranking function involving both x and b, or alternatively we can simply prove $R^+{\downarrow}_{\mathsf{b}=0} \subseteq \trianglerighteq_{\mathsf{x},0}$ and $R^+{\downarrow}_{\mathsf{b},1} \subseteq \trianglerighteq_{\mathsf{x},0}$. Note that we can do slightly better—the following lemmas will allow us to eliminate one of these conjuncts. ⋆

Lemma 1 *Assume that $v \in \textsc{Var}$ and that the set $L = R^*(I)(v)$ is finite. Let k_1 and k_2 be constants from $\textsc{Val}$. Assume that, if $(s,t) \in R$ and $t(v) = k_2$ then $s(v) = k_1$. $(R\underset{*}{\downarrow}_I^+){\downarrow}_{v=l}$ is well founded for each $l \in L$ iff $(R\underset{*}{\downarrow}_I^+){\downarrow}_{v=l}$ is well founded for each $l \in L - \{k_2\}$.*

Proof. By contradiction. Assume that there is an infinite sequence $s = s_1, s_2, s_3, \ldots$ allowed by R such that $s_i(v) = k_2$ infinitely often. By assumption, if $s_{i+1}(v) = k_2$ then $s_i(v) = k_1$. Thus $s_i(v) = k_1$ occurs infinitely often in s. But, by assumption, $(R\underset{*}{\downarrow}_I^+){\downarrow}_{v=k_1}$ is well founded, meaning that s cannot be infinite and thus contradicting the starting assumption. □

Lemma 2 *Assume that $v \in \textsc{Var}$ and that the set $L = R^*(I)(v)$ is finite. Let k_1 and k_2 be constants from $\textsc{Val}$. Assume that, if $(s,t) \in R$ and $s(v) = k_2$ then $t(v) = k_1$. $(R{\downarrow}_{I}^{+}){\downarrow}_{v=l}$ is well founded for each $l \in L$ iff $(R{\downarrow}_{I}^{+}){\downarrow}_{v=l}$ is well founded for each $l \in L - \{k_2\}$.*

Proof. By the same argument as Lemma 1 □

Lemmas 1 and 2 allow us to remove one of the termination checks from Example 8. We can now simply prove R well founded by proving $R^{+}{\downarrow}_{\mathsf{b},0)} \subseteq \trianglerighteq_{\mathsf{x},0}$

10. Further reading

The reader interested in examining the original papers from which these notes are drawn should begin with the proof of termination's undecidability [18,17], and the seminal papers on proving program correctness (*e.g.* Turing's paper on proving programs correct [19], Gupta *et al.* [11] for more information on methods of proving non-termination, and Floyd's paper on program semantics [10]. Readers interested in well-ordered sets, well-founded relations, and the ordinals should refer to a text like [4]. Readers interested in disjunctive termination proofs (*i.e.* Theorem 3) should read Podelski & Rybalchenko's paper [14] together with Ramsey's original paper [15]. The *size-change principle* [12] is very similar to Podelski & Rybalchenko's result, but specialized to functional programs. For methods of automating termination based on Podelski & Rybalchenko's result, see [2,3,5,8]. For methods of proving programs terminating that use dynamically allocated and deallocated data structures, see [3]. Note also that [13] produces arithmetic abstractions of programs that are sound for termination proving when treating programs with heap. For information on how to prove concurrent programs terminating, see [9]. To see methods on synthesizing preconditions which guarentee termination of non-terminating programs, see [7]. Finally, in the beginning of these notes we alluded to the fact that proving liveness properties can be converted into a problem of termination—for this reduction see [6]

11. Exercises

1. Assume that the variables in the following relations range over the integers. Which of the following relations are well founded? Which are not? Prove your answer by either finding (and proving the validity) of a ranking relation, or finding (and proving the validity) of a recurrence set.

   (a) $1 < 0$
   (b) $0 < 1$
   (c) $\mathsf{x}' > \mathsf{x} \wedge \mathsf{x}' < 1000$
   (d) $\mathsf{x}' > \mathsf{x} \wedge \mathsf{x}' > 1000$
   (e) $\mathsf{x}' \geq \mathsf{x} + 1 \wedge \mathsf{x}' < 1000$
   (f) $\mathsf{x}' \geq \mathsf{x} - 1 \wedge \mathsf{x}' < 1000$
   (g) $\mathsf{y}' \geq \mathsf{y} + 1 \wedge \mathsf{z}' = \mathsf{z} \wedge \mathsf{z} < 1000$
   (h) $\mathsf{y}' + 1 \geq \mathsf{y} \wedge \mathsf{z}' = \mathsf{z} \wedge \mathsf{z} < 1000$

(i) $(x' = x - 1 \vee x' = x + 1) \wedge x < 1000$
(j) $x' = x - z \wedge x > 0$
(k) $x' = x - z \wedge x' > 0$
(l) $x' = x - 1 \wedge (x > 0 \vee x < 200)$
(m) $x > 0 \wedge y > 0 \wedge [(x' = x - 1 \wedge y' = y) \vee (y' = y - 1 \wedge x' = x)]$
(n) $x > 0 \wedge y > 0 \wedge [(x' = x - 1 \wedge y' = y) \wedge (y' = y - 1 \wedge x' = x)]$
(o) $x > 0 \wedge y > 0 \wedge [(x' = x - 1 \wedge y' = y + 1) \vee (y' = y - 1 \wedge x' = x)]$
(p) $x > 0 \wedge y > 0 \wedge [(x' = x - 1 \wedge y' = y + 1) \vee (y' = y - 1 \wedge x' = x + 1)]$
(q) $(x > 0 \vee y > 0) \wedge x' = x - 1 \wedge y' = y - 1$
(r) $x > 0 \wedge x' = x - y \wedge y' = y + 1$

2. Reconsider the relations above in the case where the variables range over the real numbers? Which of the following relations are well founded? Which are not? Again, prove your answers.
3. Prove or disprove the following assertions:

   (a) If R^2 is well founded, R is well founded.
   (b) If R is well founded, R^2 is well founded.
   (c) If R is well founded, R^+ is well founded.
   (d) If R^+ is well founded, R is well founded.
   (e) If R is well founded, $R \cap Q$ is well founded.
   (f) If R is well founded, $R \cup Q$ is well founded.
   (g) If R is well founded, R^{-1} is well founded.

4. Is the following relation well founded?

$$x > 0 \wedge y > 0 \wedge [(x' = x + 1 \wedge y' = y - 1) \vee (x' = x - 1 \wedge y' = y)]$$

   If so: Use Theorem 3 to prove the following relation well founded (*i.e.* figure out the transitive closure, find two ranking relations, etc). If not, find and prove the validity of a recurrence set.
5. Translate the following program into the representation using **goto** and **assume** statements:

```
while $x > 0$ do
    $x := x - 1;$
    $y := x;$
    while $y > 0$ do
        $y := y - 1;$
    od
od
exit;
```

   What is this program's semantic meaning (in the form of a relation)? Give a valid set of cutpoints for this program. Find and prove the validity of a ranking function that proves the relation well founded.
6. Use the techniques from Section 9 to prove the following relation well founded:

$$\begin{array}{rl} R \triangleq & \mathsf{x} = 0 \Rightarrow (\mathsf{x}' = 1 \wedge \mathsf{y} > 0 \wedge \mathsf{y}' = \mathsf{y}) \\ \wedge & \mathsf{x} = 1 \Rightarrow (\mathsf{x}' = 2 \wedge \mathsf{y}' = \mathsf{y}) \\ \wedge & \mathsf{x} = 2 \Rightarrow (\mathsf{x}' = 3 \wedge \mathsf{y}' = \mathsf{y} - 1) \\ \wedge & \mathsf{x} = 3 \Rightarrow (\mathsf{x}' = 0 \wedge \mathsf{y}' = \mathsf{y}) \\ \wedge & (\mathsf{x} = 3 \vee \mathsf{x} = 2 \vee \mathsf{x} = 1 \vee \mathsf{x} = 1 \vee \mathsf{x} = 0) \end{array}$$

What is the value of $R^+\downharpoonright_{\mathsf{x}=2}$?

References

[1] Building a better bug-trap. *Economist Magazine*, June 2003.

[2] J. Berdine, A. Chawdhary, B. Cook, D. Distefano, and P. O'Hearn. Variance analyses from invariance analyses. In *POPL: Programming Language Design and Implementation*, 2007.

[3] J. Berdine, B. Cook, D. Distefano, and P. O'Hearn. Automatic termination proofs for programs with shape-shifting heaps. In *CAV: Computer Aided Verification*, 2006.

[4] G. Cantor. *Contributions to the Founding of the Theory of Transfinite Numbers.* Dover, 1955.

[5] A. Chawdhary, B. Cook, S. Gulwani, M. Sagiv, and H. Yang. Ranking abstractions. In *ESOP: European Symposium on Programming*, 2008.

[6] B. Cook, A. Gotsman, A. Podelski, A. Rybalchenko, and M. Vardi. Proving that programs eventually do something good. In *POPL: Programming Language Design and Implementation*, 2007.

[7] B. Cook, S. Gulwani, T. Lev-Ami, A. Rybalchenko, and M. Sagiv. Proving conditional termination. In *CAV: Computer Aided Verification*, 2008.

[8] B. Cook, A. Podelski, and A. Rybalchenko. Termination proofs for systems code. In *PLDI: Programming Language Design and Implementation*, 2006.

[9] B. Cook, A. Podelski, and A. Rybalchenko. Proving thread termination. In *PLDI: Programming Language Design and Implementation*, 2007.

[10] R. Floyd. Symposia in applied mathematics. In *Mathematical Aspects of Computer Science*, 1967.

[11] A. Gupta, T. Henzinger, R. Majumdar, A. Rybalchenko, and R. Xu. Proving non-termination. In *POPL: Principles of Programming Languages*, 2008.

[12] C. S. Lee, N. D. Jones, and A. M. Ben-Amram. The size-change principle for program termination. In *POPL: Principles of Programming Languages*, 2001.

[13] S. Magill, J. Berdine, E. Clarke, and B. Cook. Arithmetic strengthening for shape analysis. In *SAS: Static Analysis Symposium*, 2007.

[14] A. Podelski and A. Rybalchenko. Transition invariants. In *LICS: Logic in Computer Science*, 2004.

[15] F. Ramsey. On a problem of formal logic. *London Math. Soc.*, 30:264–286, 1930.

[16] G. Stix. Send in the Terminator. *Scientific American Magazine*, November 2006.

[17] C. Strachey. An impossible program. *Computer Journal*, 7(4):313, 1965.

[18] A. Turing. On computable numbers, with an application to the Entscheidungsproblem. *London Mathematical Society*, 42(2):230–265, 1936.

[19] A. Turing. Checking a large routine. In *Report of a Conference on High Speed Automatic Calculating Machines*, 1949.