Simple Termination is Complicated

Brenton Bostick

Abstract

In this survey we discuss the concept of simple termination of term rewriting systems. Simple termination captures the idea of using syntax to prove termination of term rewriting systems. We cover many ideas concerning simple termination and related topics, including Kruskal's Tree Theorem, and well-quasi-orderings. Also convered are alternative definitions of terminology, and the position of simple termination as an active area of research.

1 Introduction

Proving termination of computer programs is an important and active area of research. It is by knowing that programs eventually terminate that you know search engines will return results, that rendering graphics in video games will be fast, and in general that it is safe to use computers without waiting indefinitely.

Proving termination is necessary for showing total correctness of mathematical functions and algorithms, and is part of the study of liveness properties of programs.

In this survey, we discuss simple termination of term rewriting systems. Term rewriting is a general system for representing programs as expressions or "terms", and defining transformations between states as rules transforming one term to another.

Termination proofs usually proceed by giving meaning and interpreting successive states of programs in some well-order, thus ensuring any descending sequence is finite. But interpretation of states is not necessary; proof of termination can follow from the syntax of the states themselves.

Simple termination of term rewriting systems employs just such a concept. Proofs of simple termination follow from the syntax of the terms themselves, thus making it easier to reason about than other forms of termination. This is possible by an application of Kruskal's Tree Theorem, a generalization of Ramsey's Theorem.

We assume basic knowledge of computer science and mathematics. We address many potentially confusing ideas concerning simple termination, motivate the employment of the imposing Kruskal's Tree Theorem, explain alternative definitions of terminology, clarify contradictory definitions, illustrate these ideas with several examples, and attempt to communicate that simple termination is indeed not simple, but rather complicated, but offers a very rich and interesting theory.

In section 2, we discuss general approaches to proving termination of programs. In sections 3, and 4, we cover term rewriting systems, how terms can be compared with each other, and termination ideas specific to term rewriting. With the following sections, 5, 6, 7, and 8, the bulk of the theory is developed, discussing the landmark Tree Theorem of Kruskal, its usefulness to term rewriting, the definition of simple termination, technical details about its application, and examples of usage. Finally, in sections 9 and 10, termination techniques that are strictly weaker and stronger than simple termination are covered.

2 Proving Termination of Programs

In this section, we cover some preliminaries concerning the theory of termination proofs, examples of programs illustrating the concept of termination, the difficulty of termination proofs, concepts of computation where termination is not assumed, a brief overview of some proof methods, and the history of the subject.

2.1 Preliminaries

Proving termination of programs is a very important endeavour in computer science. It is necessary for showing total correctness of mathematical functions and algorithms, and is part of the study of liveness properties of programs.

We will assume a basic knowledge of computer programs and forgo any axiomatic treatment of the subject.

2.2 Examples

We start by illustrating the difficulty of proving termination of programs with a few examples. In these examples, all values are assumed to be non-negative integers, and the *input*() function represents input from the user.

These examples are from [5].

Example 2.2.1:

```
$x := input();$
$y := input();$
while $x > 0$ && $y > 0$ do
  if $input() = 1$ then
    $x := x - 1;$
    $y := y + 1;$
  else
    $y := y - 1;$
  end if
end while
```

In Example 2.2.1, we see two variables initialized to arbitrary non-negative integers. Control loops while both are greater than 0. Inside the loop, a randomly chosen integer is tested to see if it is 1. If so, x decrements and y increments; otherwise, y decrements.

Example 2.2.2:

```
$x := input();$
$y := input();$
**while** $x > 0$ && $y > 0$ **do**
  **if** $input() = 1$ **then**
    $x := x - 1;$
    $y := input();$
  **else**
    $y := y - 1;$
  **end if**
**end while**
```

Example 2.2.2 is slightly different, in that y is set to an arbitrary integer instead of being decremented.

Example 2.2.3:

```
$x := input();$
$y := input();$
**while** $x > 0$ && $y > 0$ **do**
  **if** $input() = 1$ **then**
    $x := x - 1;$
    $y := y + 1;$
  **else**
    $x := x + 1;$
    $y := y - 1;$
  **end if**
**end while**
```

And in the final Example 2.2.3, the variables are either incremented or decremented depending on the branch.

Do these programs terminate? In this section we will show how to answer that question with mathematical proofs.

2.3 Difficulty

The problem of program termination, otherwise known as the Uniform Halting problem, could perhaps be considered the first problem posed in computer science.

The problem is that in general, programs can have an infinite number of states and we must reason about infinite sequences of states. So showing termination requires mathematical reasoning about states of programs, rather than plain observation.

In general, a program cannot be shown to be terminating or non-terminating. This is the famous result of Turing [32].

Nevertheless, there has been tremendous research in this area and it is possible to prove a great number of non-trivial computer programs to be terminating.

2.3.1 Las Vegas Algorithms

A class of computations where termination is not required is randomized algorithms. A Las Vegas algorithm is guaranteed to return a correct answer, but is not guaranteed to be complete. It has a running time that is a random variable. [15] characterizes the running times of Las Vegas algorithms.

Complete Las Vegas algorithms can be guaranteed to solve each solvable problem instance in time t_{max}, where t_{max} is a instance-dependent constant. Let $P(RT_{A,\pi} \leq t)$ denote the probability that A finds a solution for a solvable instance π in time $\leq t$, then A is complete exactly if for each π there exists some t_{max} such that $P(RT_{A,\pi} \leq t_{max}) = 1$

Approximately complete Las Vegas algorithms solve each solvable problem instance with a probability converging to 1 as the run-time approaches ∞. Thus, A is approximately complete, if for each soluble instance π, $\lim_{t\to\infty} P(RT_{A,\pi} \leq t) = 1$.

Essentially incomplete Las Vegas algorithms are Las Vegas algorithms which are not approximately complete (and therefore also not complete).

In terms of this paper, we are interested in determining when an approximately complete Las Vegas algorithm is actually complete.

2.4 Proving Termination of Programs

Proving termination of programs involves showing that all possible sequences of valid state transitions are finite.

Termination proofs for programs are sometimes ad hoc, and are sometimes systematic, but virtually all proofs follow these few steps built from the first principles developed by Turing in [33]:

1. Find a termination argument, i.e., a map into a well-founded order.
2. Show that every program state transition leads to a strict decrease in the map.

Because it is often difficult to find a single well-founded order to map program transitions, the termination argument is many times *modular*. Modularity means that a property is taken from many sets, or that a property holds over many sets.

Well-founded orders have a nice property that allows modularity:

Proposition 2.4.1: A termination argument can be a union of finitely many well-founded orders. If a transitive relation R is covered by finitely many well-founded relation $U_1, \ldots, U_n$, then R is well-founded. This is lemma 1 in [3].

Proof. Assume that there is an infinitely descending R-chain. Because there is only a finite number of relations $U_1, \ldots, U_n$, then some U_i must occur in the chain infinitely often. However, U_i is well-founded, there can be no infinitely descending U_i-chain. Thus, our initial assumption is incorrect and a termination argument can be a union of finitely many well-founded orders. □

An example of using modularity for termination is the size-change principle, introduced in [23]. The size-change principle applies when each variable of a program has a well-founded value (e.g. an integer) and every infinite chain of state transitions would cause an infinite descent of some variable. The size-change principle is modular because every infinite chain of state transitions has to consider some variable and Proposition 2.4.1 guarantees that the entire termination argument is well-founded.

Proposition 2.4.1 is a specific example of an area of infinite combinatorics called Ramsey Theory. Ramsey Theory is the study of what internal order is necessarily imposed on objects as their size increases. A familiar example of Ramsey Theory is the Party Problem:

In a group of n people, how many people are guaranteed to all know each other or all not know each other?

In a group of 6 people, it is guaranteed that some 3 people will all know each or not know each other. This guarantee of 3 people remains until the group grows in size to 18 people, where it is now guaranteed that 4 people either know or don't know each other. There are bounds on the guaranteed number for each party size, but for each party size $n > 4$, the exact guaranteed number is not known.

Ramsey Theory relates to termination proofs because it can show that some sequences of objects are necessarily finite; eventually the objects in the sequence will have a property that contradicts some pre-defined condition on the sequence. Ramsey Theory's relation to simple termination will be explored in section 5. The interested reader is directed to [12] for a good reference on Ramsey Theory.

2.5 Answers to Examples

Proposition: Example 2.2.1 is terminating.

Proof. The termination argument is the natural order of the integers on the function $2x + y$. Each iteration of the loop causes the value of the function to decrease. Because the order is on the non-negative integers and is well-founded, there can be no infinitely descending chain of state transitions. So the program is terminating. □

Proposition: Example 2.2.2 is terminating.

Proof. The termination argument is the lexicographic order on the tuple $\langle x, y \rangle$. Each iteration of the loop causes the value of the tuple to decrease. Because the order is on the tuples of non-negative integers and is well-founded, there can be no infinitely descending chain of state transitions. So the program is terminating. □

Proposition: Example 2.2.3 is not terminating.

Proof. There is a sequence of inputs that could result in an infinite loop. For example, after intializing x and y to arbitrary values, the infinite sequence of inputs $1, 0, 1, 0, \ldots$ causes the value of x and y to simply fluctuate and never reach 0. □

2.6 History

In [32], Turing showed that the Uniform Halting Problem was undecidable. Turing proposed in [33] the first principles approach for termination proofs used in this survey. Floyd [11] pioneered the idea of interpreting programs into ordinals to show termination. [5] and [3] provide a good overview of program termination proofs.

3 Orders

In this section we develop general mathematical theory used in later sections. It can safely be skipped by the advanced reader.

3.1 Preliminaries

We begin with some preliminary definitions for orders.

Definition: A partial order is a binary relation that is reflexive, antisymmetric, and transitive.

Definition: A total order is a partial order R where $s\ R\ t$ or $t\ R\ s$ for any two terms s and t.

Definition: A strict order is a binary relation that is irreflexive and transitive (and hence asymmetric).

Definition: A well-founded order is an order with no infinitely descending chains.

Definition: A well-order is a well-founded total order.

3.2 Extensions of Orders

We now take existing orders and show how they can be extended over different structures while preserving important properties like well-foundedness.

3.2.1 Lexicographic Extension

Definition: Given an order $>$ on a set A, the lexicographic order $>_{lex}$ on $A \times A$ is defined as:

$$(a_1, a_2) >_{lex} (b_1, b_2) :\equiv a_1 > b_1 \lor a_1 = b_1 \land a_2 >_{lex} b_2$$

The ordering can be extended to arbitrarily sized tuples. The lexicographic ordering preserves well-foundedness.

3.2.2 Multiset Extension

Definition: Given a well-founded order $>$ on a set A, the multiset order $>_{mul}$ on $\mathcal{M}(A)$, the multisets of A, is defined as:

$$A_0 >_{mul} A_1 :\equiv A_0 \neq A_1 \land \forall a_1 \in A_1 - A_0.\ \exists a_0 \in A_0 - A_1.\ a_0 > s_1$$

In words, the multiset order on multisets looks at every element in A_1, and checks if there is some element in A_0 that is larger. The multiset ordering preserves well-foundedness.

3.2.3 Term Extension (Recursive Path Ordering)

Definition: Given a well-founded order $>$ (called a precedence) on an alphabet $\mathcal{F}$ of function symbols, the term order $>_{rpo}$ on terms $\mathcal{T}(\mathcal{F})$, is defined as:

$$s = f(s_1, \ldots, s_m) >_{rpo} g(t_1, \ldots, t_n) = t$$

if either (of the three) cases hold:

1. $s_i = t$ or $s_i >_{rpo} t$, for some $i \in 1, \ldots, n$
2. $s >_{rpo} t_i$, for all $i \in 1, \ldots, m$, and
   (a) $f > g$ or
   (b) $f = g$ and $(s_1, \ldots, s_n) >_{rpo}^{\tau(f)} (t_1, \ldots, t_m)$ where $\tau(f)$ is a status function of f: an ordering on the tuples of f's subterms.

In words, the rpo on terms recursively compares terms syntactically based on the status and precedence of the functions symbols. The recursive path ordering preserves well-foundedness.

3.3 Orders on sets of terms

There are orders specific to sets of terms, relating different aspects of term rewriting.

Definition: A stable ordering is an ordering $>$ such that $s > t$ implies $\sigma(s) > \sigma(t)$ for any substitution σ.

Definition: A monotone ordering is an ordering $>$ such that $s > t$ implies $C[s] > C[t]$ for any context C.

Monotonicity makes sense when function symbols have meaning, like if *plus* or *times*, but it also makes sense abstractly, when the function symbols don't have any inherent meaning.

Definition: A rewrite ordering is a stable, monotonic, strict ordering.

Definition: A reduction ordering is a well-founded rewrite ordering.

4 Proving Termination of Term Rewriting Systems

Termination is an important property in term rewriting. In conjunction with the general importance of termination for programs, results that depend on well-founded induction such as Newman's Lemma also depend on termination.

4.1 Preliminaries

Definition: A TRS $\mathcal{R}$ is terminating if there is no infinite sequence of reductions $t_1 \to t_2 \to t_3 \ldots$

4.2 Converting Programs to TRSs

The examples from section 2.2 can easily be converted into roughly-equivalent TRSs.

Example 4.2.1: TRS for 2.2.1 The TRS has signature $\mathcal{F} = \{s, z\}$, where s is unary and z is a constant.

$$\begin{aligned} \langle s(x), s(y)\rangle &\to \langle x, s(s(y))\rangle \\ \langle s(x), s(y)\rangle &\to \langle s(x), y\rangle \end{aligned}$$

Example 4.2.2: TRS for 2.2.2 The TRS has signature $\mathcal{F} = \{s, z\}$, where s is unary and z is a constant.

$$\begin{aligned} &\langle s(x), s(y)\rangle \to \langle x, w\rangle \text{ where } w \text{ is any arbitrary term} \\ &\langle s(x), s(y)\rangle \to \langle s(x), y\rangle \end{aligned}$$

Example 4.2.3: TRS for 2.2.3 The TRS has signature $\mathcal{F} = \{s, z\}$, where s is unary and z is a constant.

$$\begin{aligned} \langle s(x), s(y)\rangle &\to \langle s(s(x)), s(y)\rangle \\ \langle s(x), s(y)\rangle &\to \langle s(x), s(s(y))\rangle \end{aligned}$$

4.3 Characterizing Termination

There are several ways to characterize termination. We will state several formulations of termination, with the goal of arriving upon an approach that is potentially decidable.

Proposition 4.3.1: A TRS is terminating iff all instances of its LHS terminate. This is Lemma 1 from [9].

This is obviously not decidable, since there are an infinite number of instances of any given non-ground term. Corollary: Termination is decidable for ground TRSs [16].

Proposition 4.3.2: A TRS is terminating iff there exists a well-founded ordering $>$ such that $s \to^* t$ implies $s > t$, for all terms s and t. This is Theorem 5 from [9], and originally stated in [24].

This is not a desirable condition either; there are an infinite number of rewrites to consider.

Proposition 4.3.3: A TRS $\mathcal{R}$ is terminating iff there exists a compatible reduction ordering $>$.

If an ordering is closed under contexts and substitutions, then the order $>$ only has to be checked for the given rewrite rules, instead of every possible reduction.

This is much better. There are a finite number of cases, so it is potentially possible to check. Note: the challenge is still the challenge of determining an order $>$.

4.4 Different classes of termination

There are many classes of termination. By termination class, we mean a set of TRSs that can be shown terminating all by the same method.

Example 4.4.1: term-depth ordering: $s >_{depth} t$ iff term s has a strictly greater depth than t. "term-depth" termination: if the RHS of every rule is strictly smaller (term-depth ordering) than the LHS, then the TRS is term-depth terminating. This is very intuitive, simple induction. The TRS $f(f(x)) \to f(x)$ is term-depth terminating because each reduction removes one level from terms, so eventually there will be no more f terms.

Example 4.4.2: The TRS $f(x) \to f(f(x))$ is not terminating, since it is possible to derive the infinite chain $f(x) \to f(f(x)) \to f(f(f(x))) \to \ldots$

Example 4.4.3: The TRS $f(x) \to g(g(x))$ is not term-depth terminating, but it is terminating. Each reduction removes f terms like the previous example, but the entire term grows in size. So you could say that the system is "f-term terminating".

Example 4.4.4: TRS $f(f(x)) \to f(g(f(x)))$ is terminating because the number of consecutive f terms decreases.

So we see that term-depth termination does not completely capture the notion of termination. Indeed, there can not be a single effective proof method for all terminating TRSs. Remember: as long as there is *some* way of ordering terms so that each rewrite rule causes a decrease, then the system is terminating.

4.5 Undecidability

Termination of term rewriting is undecidable [16]. The proof proceeds by reducing termination of term rewriting to the Uniform Halting Problem for Turing machines. Termination of 1 rule TRS is undecidable [6]. Termination of length-preserving, unary TRS is undecidable [4].

But there are decidable subcases. Termination is decidable for ground TRSs [16].

4.6 Modularity

Termination is not modular. The famous result from Toyama [25] is an example.

Example 4.6.1: The TRSs given by:

$$\begin{aligned} R_0 &\equiv \{f(0,1,x) \to f(x,x,x)\} \\ R_1 &\equiv \{g(x,y) \to x, g(x,y) \to y\} \end{aligned}$$

are both terminating. But the combined TRS is not, as seen by the infinite reduction

$$f(g(0,1),g(0,1),g(0,1)) \to f(0,1,g(0,1)) \to f(g(0,1),g(0,1),g(0,1)) \to \dots$$

5 Kruskal's Tree Theorem

Kruskal's Tree Theorem is the Fundamental Theorem of Simple Termination. Simply stated, it shows that if your function symbols have an order with a property slightly stronger than well-foundedness, then the set of trees of those function symbols has a well-founded order. Thus, certain TRSs can be shown terminating by checking a simple property of the rules. Before we can jump into the theorem, we must cover some preliminaries.

5.1 Homeomorphic Embedding

Definition: The homeomorphic embedding relation $\trianglelefteq_{emb}$ is defined as follows: $s \trianglelefteq_{emb} t$ between terms s and t iff one of the conditions is true:

1. $s = x = t$ for a variable $x \in \mathcal{V}$
2. $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ and $s_1 \trianglelefteq_{emb} t_1, \ldots, s_n \trianglelefteq_{emb} t_n$
3. $t = f(t_1, \ldots, t_n)$ and $s \trianglelefteq_{emb} t_j$

Homeomorphic embedding captures the sense of syntactic simplicity; embedded terms are arrived at by removing subterms.

Example 5.1.1:

$$f(f(a, x), x) \trianglelefteq_{emb} f(f(h(a), h(x)), f(h(x), a))$$

5.2 Well-Partial-Orders

As discussed in section 2, proof termination usually follows from showing that some ordered set of program states is well-founded.

But working with well-founded sets is cumbersome. If we could choose to work with a subset of a well-founded order, such that nice properties are preserved, then termination would be easier to show for certain classes. That is what well-partial-orders are for.

Definition: A partial order $\geq$ on a set A is a well-partial-order (wpo) iff for every infinite sequence $a_1, a_2, a_3, \ldots$ of elements of A there exist indices $i < j$ such that $a_i \leq a_j$.

Well-partial-orders are not equivalent to well-founded partial orders. This is very important.

With a wpo, you have the guarantee that some two elements will be comparable. A regular po provides no such guarantee. A wpo can be viewed as a po with a restriction, the $a_i \leq a_j$ property. We will see that this property is very powerful.

There are several equivalent ways of defining well-partial-orders.

Definition: A partial order $\geq$ on a set A is a well-partial-order iff every subset of A has at least one, but no more than a finite number, of minimal elements. c.f., a well-founded set, which has at least one minimal element, and may have an infinite numbers of minimal elements.

Definition: A partial order $\geq$ on a set A is a well-partial-order iff every strictly decreasing sequence is finite and every set of pairwise incomparable elements is finite. c.f., a well-founded set, which have an infinite number of pairwise in comparable elements.

Well-partial-orders satisfy a number of useful properties.

Proposition 5.2.1: If a set X is wpo, then any subset of X is wpo.

Proposition 5.2.2: wpo is preserved under finite unions.

Proposition 5.2.3: A set X being well-ordered implies X is wpo.

Proposition 5.2.4: Every infinite sequence has an ascending infinite subsequence (wrt a wpo $\geq$).

Proposition 5.2.5: The extension of a wpo $\geq$ over cartesian product is wpo.

5.3 Kruskal's Tree Theorem

The proof of Kruskal's Tree Theorem in this article is due to Nash-Williams [29], which is simpler than the original in [19].

Definition: An infinite sequence $a_1, a_2, a_3, \ldots$ is *good* iff there exist $i < j$ such that $a_i \leq a_j$. Otherwise, the sequence is *bad*.

First, we will prove some useful lemmas.

Lemma 5.3.1: Given a bad sequence, a minimal bad sequence exists

Proof. Assume that a bad sequence of terms well-partial-ordered by $\trianglelefteq$ exists. We construct a minimal bad sequence by induction.

Assume that a bad sequence exists starting with terms $t_1, t_2, t_3, \ldots, t_n$ Let t_{n+1} be a minimal term (according to term-size) among the set of all terms that occur at position $n+1$ of a bad sequence that starts with $t_1, t_2, t_3, \ldots, t_n$. By the IH, there exists at least one sequence.

This defines a minimal bad sequence $t_1, t_2, t_3, \ldots$ □

Lemma 5.3.2: $\trianglelefteq$ is a wpo on the proper subterms of the elements of a minimal bad sequence

Proof. We define S to be the set of proper subterms of a minimal bad sequence. For each t_i in a minimal bad sequence, define S_i to be $\{\}$ if t_i is a variable, otherwise $t_i = f_i(s^i_1, \ldots, s^i_{n_i})$, for some function symbol f_i with arity n_i and we define $S_i = \{s^i_1, \ldots, s^i_{n_i}\}$. Therefore $S = \cup_i S_i$.

Assume $s_1, s_2, s_3, \ldots$ is a bad sequence in S. Let k be such that $s_1 \in S_k$. Because $\trianglelefteq$ is reflexive, the sequence can only be bad if all s_i are distinct. Thus, since $U = S_1 \cup \ldots \cup S_{k-1}$ is finite, there exists an $l \geq 1$ such that $s_i \in S - U$ for all $i \geq l$. Because s_1 is a proper subterm of t_k and by minimality of $t_1, t_2, t_3, \ldots$, the sequence $t_1, \ldots, t_{k-1}, s_1, s_l, s_{l+1}, \ldots$ is good. Since the sequences $t_1, t_2, t_3, \ldots$ and $s_1, s_2, s_3, \ldots$ are bad, this can only be the case if there exist indices $i \in \{1, \ldots, k-1\}$ and $j \in \{1, l, l+1, \ldots\}$ such that $t_i \trianglelefteq s_j$. If $j = 1$, then $s_j = s_i$ is a subterm of t_k and thus $t_i \trianglelefteq t_k$, a contradiction. Otherwise, let m be such that $s_j \in S_m$. Since $j \geq l$, we know that $s_j \notin U$, which yields $i < k \leq m$. However, $s_j \in S_m$ means that s_j is a subterm of t_m, and thus $t_i \trianglelefteq t_m$, a contradiction. In both cases, we have seen that assuming the sequence $s_1, s_2, s_3, \ldots$ is bad lead to a contradiction, so therefore there can be no bad sequence $s_1, s_2, s_3, \ldots$, and the set of proper subterms of the elements of a minimal bad sequence is a wpo. □

Theorem 5.3.3: Kruskal's Tree Theorem (Finite Form)

Let $\mathcal{F}$ be a finite signature and $\mathcal{V}$ be a finite set of variables. Then the homeomorphic embedding $\trianglelefteq$ on $\mathcal{T}(\mathcal{F}, \mathcal{V})$ is a wpo.

Proof. We use the minimal bad sequence method introduced by Nash-Williams.

Consider the minimal bad sequence $t_1, t_2, t_3, \ldots$. Since $\mathcal{F} \cup \mathcal{V}$ is finite, there is some sequence of indices $i_1 < i_2 < i_3 < \cdots$ such that $t_{i_1}, t_{i_2}, t_{i_3}, \ldots$ all have the same root symbol. If this root symbol is a variable or constant, we have $t_{i_1} = t_{i_2}$, implying $t_{i_1} \trianglelefteq t_{i_2}$. Otherwise, the common root symbol must be a function symbol f, with some arity n.

Because of Lemma 5.3.2 and Lemma 5.2.5, the sequence $(s^{i_1}_1, \ldots, s^{i_1}_n)$, $(s^{i_2}_1, \ldots, s^{i_2}_n), (s^{i_3}_1, \ldots, s^{i_3}_n), \ldots$ of proper subterms of $t_{i_1}, t_{i_2}, t_{i_3}, \ldots$ is good with respect to the order $\trianglelefteq$ extended over cartesian products.

Because the sequence $(s^{i_1}_1, \ldots, s^{i_1}_n), (s^{i_2}_1, \ldots, s^{i_2}_n), (s^{i_3}_1, \ldots, s^{i_3}_n), \ldots$ is good, there are indices $q < r$ such that $s^{i_q}_1 \trianglelefteq s^{i_r}_1 \wedge \ldots \wedge s^{i_q}_n \trianglelefteq s^{i_r}_n$. This implies $t_{i_q} \trianglelefteq t_{i_r}$.

So irrespective of the common root symbol of $t_{i_1}, t_{i_2}, t_{i_3}, \ldots$, the sequence must be good and the minimal bad sequence $t_1, t_2, t_3, \ldots$ must be good, a contradiction.

Our assumption that there is a bad sequence in $\mathcal{T}(\mathcal{F}, \mathcal{V})$ leads to a contradiction, so all infinite sequences in $\mathcal{T}(\mathcal{F}, \mathcal{V})$ are good and thus $\trianglelefteq$ is wpo. □

6 Simple Termination

Simple termination is a class of termination proofs that uses Kruskal's Tree Theorem to show that rewrite systems having a very simple property are terminating.

6.1 Preliminaries

Definition: An ordering $>$ on terms has the *subterm property* if $t > t|_p$, for all subterms $t|_p$ of t.

Assuming that $>$ is a rewrite order, then you just need to show $f(\ldots, x, \ldots) > x$ for the subterm property.

Definition: A *simplification ordering* is a rewrite ordering that has the subterm property.

Notice that we don't necessarily have a reduction ordering.

Definition: A TRS $\mathcal{R}$ is simplifying if admits a compatible simplification order.

Might be confusing, since it might seem obvious that a simplifying TRS is terminating. Simplifying does not assume well-foundedness.

Definition: A TRS $\mathcal{R}$ is *simply terminating* if admits a compatible *well-founded* simplification order.

6.2 Embedding

Definition: Let $\mathcal{F}$ be a signature. Define the TRS $\mathcal{E}mb(\mathcal{F})$ as the set of all rules

$$f(\ldots, x_i, \ldots) \to x_i$$

Proposition 6.2.1: For a finite TRS $\mathcal{R} = (\mathcal{F}, R)$, the following statements are equivalent:

1. $\mathcal{R}$ is simply terminating.
2. $\mathcal{R} \cup \mathcal{E}mb(\mathcal{F})$ is simply terminating.
3. $\mathcal{R} \cup \mathcal{E}mb(\mathcal{F})$ is terminating.

But we still haven't shown that simple termination implies termination.

6.3 Termination

Proposition 6.3.1: $\trianglelefteq_{emb}$ implies $<_{simp}$

Simplification orderings contain the homeomorphic embedding relation.

Theorem 6.3.2: Let $\mathcal{F}$ be a finite signature. Every simplification order $>$ on $\mathcal{T}(\mathcal{F}, \mathcal{V})$ is well-founded.

Proof. $>$ is a simplification order. Assume an infinite chain of terms $t_1 > t_2 > t_3 \ldots$ Assume that $x \in \mathcal{V}ar(t_{i+1}) - \mathcal{V}ar(t_i)$ and the substitution $\sigma = \{x \to t_i\}$. Clearly, $t_i = \sigma(t_i)$. Because $>$ is a rewrite order, then also $\sigma(t_i) > \sigma(t_{i+1})$. But $>$ is simplification order, which means it has the subterm property. t_i being a subterm of $\sigma(t_{i+1})$ means that $\sigma(t_{i+1}) \geq t_i$. We have $t_i = \sigma(t_i) > \sigma(t_{i+1}) \geq t_i$, a contradiction. So there is not an $x \in \mathcal{V}ar(t_{i+1}) - \mathcal{V}ar(t_i)$. for all terms in the chain. The set of variables in the chain must be the finite set $X := Var(t1)$. Since both the signature $\mathcal{F}$ and the variables $\mathcal{V}$ are finite, Kruskal implies that this sequence is good. For some i and j, $t_i \trianglelefteq t_j$. Lemma 6.3.1 implies $t_i \leq t_j$. But this is a contradiction, since we know $t_i > t_j$. Thus, there are no infinite chains of terms $t_1 > t_2 > t_3 \ldots$, and $>$ is well-founded. $\square$

6.4 History

The notion of "simplification ordering" and its well-foundedness were proved in [7]. The terminology "simple termination" was introduced in [21]. In the same paper, Kurihara and Ohuchi proved that simple termination was modular. Simple termination of one-rule systems is undecidable [27].

7 Well-Quasi-Orders

Until now, we have only talked about partial orders and well-partial-orders. References to quasi-orders and well-quasi-orders are common in the literature. These are related but distinct concepts.

Definition: A quasi order $\precsim$ is a binary relation that is reflexive and transitive.

Definition: A partial order $\geq$ on a set A is a well-quasi-order (wqo) iff for every infinite sequence $a_1, a_2, a_3, \ldots$ of elements of A there exist indices $i < j$ such that $a_i \precsim a_j$.

The machinery for reasoning about quasi-orders is largely the same as that for strict orders. "Note that, except for technical details, well-quasi-ordered is the same as well-partial-ordered. [20]" "At the casual level it is easier to work with po than qo, but in advanced work the reverse is true. [20]" Using wqo or wpo for proving Kruskal's Tree Theorem yield equivalent results. The original statement of the Theorem [19] was in terms of wqo.

7.1 Differences

Well-partial-orders and well-quasi-orders do not have the same expressive power. Despite their similarities in other fields, in term rewriting, there are cases where qo is strictly more powerful than po.

Example 7.1.1: The TRS

$$f(f(x)) \to g(x)$$
$$g(g(x)) \to f(x)$$

cannot be proved terminating with a lexicographic path order based on partial orders, but it can with an lpo based on a quasi-orders.

Proof. There is no strict precedence for f and g that admits a compatible lpo on the TRS, since the function symbols are symmetric in the TRS. However, a quasi-order where f and g are equal is sufficient for showing termination. $\square$

Example 7.1.2: If f has multiset status, then $f(a, b)$ and $f(b, a)$ are equivalent, and if $f > c > d$ then it is possible to show

$$f(f(a, b), c) >_{rpo} f(f(b, a), d)$$

Without a mpo based on quasi-orders, this is not possible.

7.2 History

Well-quasi-orders have a rich history. Kruskal seems to be the first to use the term "well-quasi-ordering", but the concept dates back to at least 1952 [13]. [20] is a survey of the field up to 1970, with several comments about its origins. [3] contains a historical perspective at the end.

8 Examples of Simplification Orderings

Simple termination is a useful concept, but it does not provide any concrete methods for showing termination. In this section we explore various popular simplification orderings, and their strengths and weaknesses.

8.1 Lexicographic Path Ordering

Lexicographic path ordering is an instance of RPO from Section 3.2.3, with the lexicographic extension as the status for function symbols. It was introduced in [17].

Example 8.1.1: Ackermann's function The TRS

$$\begin{aligned} ack(0, y) \rightarrow & \quad succ(y) \\ ack(succ(x), 0) \rightarrow & \quad ack(x, succ(0)) \\ ack(succ(x), succ(y)) \rightarrow & \quad ack(x, ack(succ(x), y)) \end{aligned}$$

can be shown to terminate by using the precedence $ack \succ succ$.

8.2 Multiset Path Ordering

Multiset path ordering is an instance of RPO from Section 3.2.3, with the multiset extension as the status for function symbols. [8] introduced the multiset path ordering, but called it *recursive path ordering*.

8.3 Polynomial Simplification Orders

By interpreting function symbols of a TRS as polynomials over the reals, it is possible to prove well-foundedness if the polynomials satisfy certain requirements. First, the polynomials must be monotonic. Second, the polynomials must have the subterm property. By Tarski's decision procedure, the first-order theory of the reals is decidable, and so determining whether a given set of polynomials is a valid interpretation of a given TRS is decidable.

9 Below Simple Termination

There are simpler methods of proving termination than simple termination. In fact, there is a hierarchy of termination proofs for term rewriting systems.

9.1 The Hierarchy

Simple termination contains all common notions of termination.

The Termination Hierarchy is:

$$\begin{aligned}\text{polynomial termination} &\Rightarrow\\ \omega\text{-termination} &\Rightarrow\\ \text{total termination} &\Rightarrow\\ \text{simple termination} &\Rightarrow\\ \text{termination}&\end{aligned}$$

Each level of the hierarchy represents a class of proof methods or some mathematical completion of a concept.

Polynomial termination is the set of TRSs that terminate via the polynomial interpretation from [22]. The idea is that terms can be interpreted as monotonic polynomials on $\mathbb{N}$, so that monotonicity and well-foundedness are preserved after composition.

ω-termination is the set of TRSs that terminate via some interpretation into monotonic functions in $\mathbb{N}$. Exponential interpretation is in this set, as well as any functions on the integers that satisfy the requirements.

Total termination is the set of TRSs that terminate via some monotonic total order.

Each level of the hierarchy is strict: there are examples of each class termination that the previous class cannot prove. For an overview of the hierarchy, see the [36] and section 6.3.3 of [26].

Theorem 9.1.1: Total termination implies simple termination

Let $>$ be a reduction order total on ground terms. Then $>$ satisfies the subterm property.

Proof. Assume we have $t \not> t|_p$ for all $p \neq \epsilon$. By totality, we must have $t|_p > t$. This leads to the infinite descent $t|_p > t = t[t|_p]_p > t[t]_p > t[t[t]_p]_p > \dots$. This is a contradiction, so $>$ must have the subterm property. $\square$

For all reduction orderings, we always have $!(t|_p > t)$, a subterm is never greater than a term. But can sometimes the two can be incomparable (both

$!(t|_p > t)$ and $!(t > t|_p)$). You could also sometimes have $t > t|_p$, but it is not required. For a simplification ordering, you always have $t > t|_p$.

9.2 Simple Termination does not imply Total Termination

While it is somewhat easy to see that total termination implies simple termination, the converse is not true. There are simply terminating systems that are not totally terminating.

Theorem 9.2.1: The TRS $\mathcal{R}$ consisting of the signature $\mathcal{F} = \{f, g, a, b\}$ and the rules

$$f(a) \to f(b)$$
$$g(b) \to g(a)$$

is not totally terminating.

Proof. $\mathcal{R}$ cannot be totally terminating because there is no total order such that both $f(a) > f(b)$ and $g(b) > g(a)$. By monotonicity, any total order would imply $a > b$ and $b > a$, a contradiction. □

Theorem 9.2.2: The TRS $\mathcal{R}$ consisting of the signature $\mathcal{F} = \{f, g, a, b\}$ and the rules

$$f(a) \to f(b)$$
$$g(b) \to g(a)$$

is simply terminating.

Proof. By theorem 6.2.1, $\mathcal{R}$ is simply terminating if $\mathcal{R} \cup \mathcal{E}mb(\mathcal{F})$ is terminating. There is an easily defined simplification order on $\mathcal{T}(\mathcal{F}, \mathcal{V})$ satisfying this requirement, namely the order defined by:

$$\begin{aligned} f(x) &> x \\ g(x) &> x \\ f(a) &> f(b) \\ g(b) &> g(a) \\ x > y &\to f(x) > f(y) \\ x > y &\to g(x) > g(y) \end{aligned}$$

for variables $x \in \mathcal{V}$. □

9.3 Popular Simplification Orderings

Several popular simplification orderings are actually totally terminating. LPO is totally terminating [36] [34], and MPO is ω-terminating [36] [14].

Even though it is useful to base the definitions of LPO and MPO on simplification orders, it is not necessary. It is interesting to note that actual examples of simplification orderings are not strictly simply terminating. The proofs of total termination came later, after the definitions.

9.4 History

The Termination Hierarchy was defined in [35]. Polynomial termination was first mentioned in [22]. ω-termination was never formally defined, as it intuitively follows from polynomial termination. It was first mentioned in [35]. Total termination was defined in [10].

10 Beyond Simple Termination

Simple termination is useful as a descriptive class of terminating systems, but it does not contain all terminating systems. There are systems that are terminating, but not simply terminating.

In this section we explore various systems that are terminating but not simply terminating, and some proof methods that can prove systems terminate that escapes simple termination.

10.1 Non-self-embedding systems

Non-self-embedding systems are a strict superset of simply terminating systems, but they are not very useful in practice.

Definition: A reduction chain $t_1 \to t_2 \to t_3 \to \dots$ is *self-embedding* if $t_i \trianglelefteq t_j$ for some $i < j$. A TRS is *self-embedding* if it allows self-embedding reductions.

Theorem 10.1.1: For a finite TRS, non-termination implies self-embeddingness.

Proof. Follows from 5.3.3, Kruskal's Tree Theorem. $\square$

Corollary 10.1.2: Non-self-embeddingness implies termination.

We will see that there is a strict separation between simple termination and non-self-embedding.

Theorem 10.1.3: Simple termination implies non-self-embedding.

Proof. Follows from 6.2.1, the equivalences between termination and simple termination. $\square$

Theorem 10.1.4: The TRS $f(g(g(x))) \to h(g(g(f(f(h(g(x)))))))$ is not simply terminating.

Proof. By 6.2.1, it suffices to show that $\mathcal{R} \cup \mathcal{E}mb(\mathcal{F})$ is not terminating. This is seen with the reduction:

$$\begin{aligned} f(g(g(x))) \to h(g(g(f(f(h(g(x))))))) \to^*_{Emb} f(f(g(x))) \to \\ f(h(g(g(f(f(h(x))))))) \to^*_{Emb} f(g(g(x))) \end{aligned}$$

$\square$

Theorem 10.1.5: The TRS $fgg \to hggffhg$ is non-self-embedding

Proof. Proposition 8 of [36] □

Proposition 10.1.6: Showing non-self-embedding is not sufficient for showing termination of all TRS.

Proof. The TRS
$$f(f(x)) \to f(g(f(x)))$$
is terminating, but not non-self-embedding. It is terminating because the number of consecutive f terms strictly decreases each step. It is not non-self-embedding because $f(f(x)) \lhd f(g(f(x)))$. □

Theorem 10.1.7: Determining whether a given TRS is self-embedding is undecidable.

Proof. Given in [31]. □

There are systems with infinite signatures that are non-self-embedding, but not terminating. Such as the TRS $a_i \to a_{i+1}$. Non-self-embedding is only useful for finite signatures, and is not a general class of termination.

10.2 Dependency Pairs

The Dependency Pair method can be used to show termination for systems that are not simply terminating.

Definition: A *defined symbol* $f \in \mathcal{F}$ of a TRS R is the root symbol of a left-hand side rule of R.

Definition: If $f(s_1, \ldots, s_n) \to C[g(t_1, \ldots, t_m)]$ is a rule in R and g is a defined symbol of R, then

$$\langle F(s_1, \ldots, s_n), G(t_1, \ldots, t_m)\rangle$$

is called a *dependency pair*.

Definition: An infinite sequence $(\langle s_i, t_i\rangle)_{i=1,2,3,\ldots}$ of dependency pairs of a TRS R is called an infinite R-chain if substitutions σ_i exist such that $t_i^{\sigma_i} \to^* s_{i+1}^{\sigma_{i+1}}$ for every $i = 1, 2, 3, \ldots$

Proposition 10.2.1 relates R-chains to termination.

Proposition 10.2.1: A TRS is terminating iff it does not admit an infinite R-chain.

Example 10.2.2: $f(f(x)) \to f(g(f(x)))$ is terminating.

Proof. f is the only defined symbol. f occurs twice on the RHS, so there are two dependency pairs:

$$p_1 = \langle F(f(x)), F(g(f(x)))\rangle \text{ and } p_2 = \langle F(f(x)), F(x)\rangle$$

We proceed with proof by contradiction. Assume that $(\langle s_i, t_i\rangle)_{i=1,2,3,\dots}$ is an infinite R-chain. There are no substitutions σ, τ such that $F(g(f(x)))^\sigma$ $r^* F(f(x))^\tau$, so the R-chain must be made up exclusively of p_2. Necessarily, $F(x)^{\sigma_i} \rightarrow^* F(f(x))^{\sigma_{i+1}}$. x^{σ_i} must have one more occurence of f than $x^{\sigma_{i+1}}$, for all $i = 1, 2, 3, \dots$. However, this is a contradiction since f-occurences in a term is well-founded. So there is no infinite R-chain, and the TRS is terminating. □

Dependency pairs were introduced in [1] and section 6.5.5 of [26] has a good survey on the method.

References

[1] Thomas Arts and Jürgen Giesl. Automatically proving termination where simplification orderings fail. In *TAPSOFT '97: Proceedings of the 7th International Joint Conference CAAP/FASE on Theory and Practice of Software Development*, pages 261–272, London, UK, 1997. Springer-Verlag.

[2] Franz Baader and Tobias Nipkow. *Term rewriting and all that.* Cambridge University Press, New York, NY, USA, 1998.

[3] Andreas Blass and Yuri Gurevich. Program termination and well partial orderings. *ACM Trans. Comput. Logic*, 9(3):1–26, 2008.

[4] A.-C. Caron. Linear bounded automata and rewrite systems: influence of initial configurations on decision properties. In *TAPSOFT '91: Proceedings of the international joint conference on theory and practice of software development on Colloquium on trees in algebra and programming (CAAP '91): vol 1*, pages 74–89, New York, NY, USA, 1991. Springer-Verlag New York, Inc.

[5] Byron Cook, Andreas Podelski, and Andrey Rybalchenko. Proving program termination. *Commun. ACM*, 2009. to appear.

[6] Max Dauchet. Simulation of turing machines by a regular rewrite rule. *Theor. Comput. Sci.*, 103(2):409–420, 1992.

[7] Nachum Dershowitz. A note on simplification orderings. *Inf. Process. Lett.*, 9(5):212–215, 1979.

[8] Nachum Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17(3):279–301, 1982.

[9] Nachum Dershowitz. Termination of rewriting. *J. Symb. Comput.*, 3(1-2):69–116, 1987.

[10] M.C.F. Ferreira and H. Zantema. Total termination of term rewriting. Technical Report RUU-CS-92-42, Department of Information and Computing Sciences, Utrecht University, 1992.

[11] R. W. Floyd. Assigning meanings to programs. *Mathematical aspects of computer science*, 19(19-32):1, 1967.

[12] Ronald L. Graham and Bruce L. Rothschild. *Ramsey theory (2nd ed.).* Wiley-Interscience, New York, NY, USA, 1990.

[13] G. Higman. Ordering by divisibility in abstract algebras. *Bull. London Math. Soc.*, 3:326–336, 1952.

[14] Dieter Hofbauer. Termination proofs by multiset path orderings imply primitive recursive derivation lengths. *Theor. Comput. Sci.*, 105(1):129–140, 1992.

[15] Holger H. Hoos and Thomas Stutzle. Evaluating las vegas algorithms — pitfalls and remedies. In *In Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98*, pages 238–245. Morgan Kaufmann Publishers, 1998.

[16] G. Huet and D. S. Lankford Lankford. On the uniform halting problem for term rewriting systems. Technical report, Rapport Laboria 283, INRIA, 1978.

[17] Sam Kamin and Jean-Jacques Levy. Attempts for generalizing the recursive path orderings. unpublished, 1980.

[18] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–267. Pergamon, New York, 1970.

[19] J. B. Kruskal. Well-quasi-ordering, the tree theorem, and vazsonyi's conjecture. *Transactions of the American Mathematical Society*, 95(2):210–225, 1960.

[20] J. B. Kruskal. The theory of well-quasi-ordering: A frequently discovered concept. *J. Combin. Theory Ser. A*, 13:297–305, 1972.

[21] Kurihara and Ohuchi. Modularity of simple termination of term rewriting systems. *Information Processing Society of Japan*, 5:1–2, 1990.

[22] Dallas Lankford. On proving term rewriting systems are noetherian. Technical Report Memo MTP-3, Mathematics Department, Louisiana Tech. University, 1992. 1979.

[23] Chin Soon Lee, Neil D. Jones, and Amir M. Ben-amram. The size-change principle for program termination, 2001.

[24] Zohar Manna and Stephen Ness. On the termination of markov algorithms. In *Proc. 3rd Hawaii Int. Conf. on Systems Science.*, pages 782–792, 1970.

[25] Yoshihito Toyama. Counterexamples to termination for the direct sum of term rewriting systems. *Inf. Process. Lett.*, 25(3):141–143, 1987.

[26] Roel de Vrijer Marc Bezem, Jan Willem Klop, editor. *Term Rewriting Seminar – Terese*. Cambridge University Press, 2003.

[27] Aart Middeldorp and Bernhard Gramlich. Simple termination is difficult. In *Applicable Algebra in Engineering, Communication and Computing*, pages 228–242. Springer, 1993.

[28] Aart Middeldorp and Hans Zantema. Simple termination of rewrite systems. *Theoretical Computer Science*, 175:127–158, 1997.

[29] C. S. J. A. Nash-Williams. On well-quasi-ordering finite trees. In *Proceedings of the Cambridge Philosophical Society*, volume 59 of *Proceedings of the Cambridge Philosophical Society*, pages 833–+, 1963.

[30] Enno Ohlebusch. A note on simple termination of infinite term rewriting systems. Technical report, 1992.

[31] David A. Plaisted. The undecidability of self-embedding for term rewriting systems. *Inf. Process. Lett.*, 20(2):61–64, 1985.

[32] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proc. London Math. Soc.*, 2(42):230–265, 1936.

[33] Alan M. Turing. Checking a large routine. In Anonymous, editor, *Report on a Conference on High Speed Automatic Computation, June 1949*, pages 67–69, Cambridge, UK, 1949. University Mathematical Laboratory, Cambridge University. Inaugural conference of the EDSAC computer at the Mathematical Laboratory, Cambridge, UK.

[34] Andreas Weiermann. Termination proofs for term rewriting systems by lexicographic path orderings imply multiply recursive derivation lengths. *Theor. Comput. Sci.*, 139(1-2):355–362, 1995.

[35] Hans Zantema. Termination of term rewriting: Interpretation and type elimination. *Journal of Symbolic Computation*, 17:23–50, 1994.

[36] Hans Zantema. The termination hierarchy for term rewriting. *Appl. Algebra Engrg. Comm. Comput*, 1999.