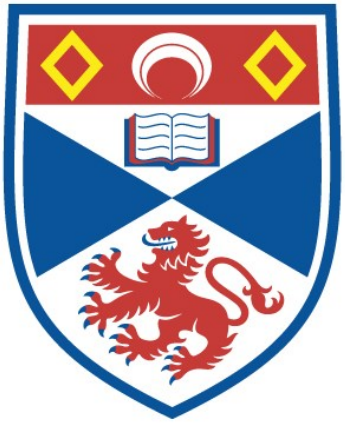
EFFECTIVE TERMINATION TECHNIQUES

Nick I. Cropper

A Thesis Submitted for the Degree of PhD
at the
University of St Andrews



1997

Effective Termination Techniques

Nick I Cropper

30 September 1996

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science
at the University of St Andrews

I, Nick Cropper, hereby certify that this thesis, which is approximately 30 000 words in length, has been written by me, that it is the record of work carried out by me, and that it has not been submitted in any previous application for a higher degree.

date 12/3/97 *signature of candidate* _____

I was admitted as a research student in October 1992 and as a candidate for the degree of doctor of philosophy in October 1993; the higher study for which this is a record was carried out in the University of St Andrews between 1992 and 1995.

date 12/3/97 *signature of candidate* _____

I hereby certify that the candidate has fulfilled the conditions of the Resolution and Regulations appropriate for the degree of doctor of philosophy in the University of St Andrews and that the candidate is qualified to submit this thesis in application for that degree.

date 12 March 97 *signature of supervisor* _____

In submitting this thesis to the University of St Andrews I understand that I am giving permission for it to be made available for use in accordance with the regulations of the University Library for the time being in force, subject to any copyright vested in the work not being affected thereby. I also understand that the title and abstract will be published, and that a copy of the work may be made and supplied to any bona fide library or research worker.

date 12/3/97 *signature of candidate* _____

Order leads to all the virtues!
but what leads to order?

Georg Christoph Lichtenberg

To Sue

Abstract

An important property of term rewriting systems is termination: the guarantee that every rewrite sequence is finite. This thesis is concerned with orderings used for proving termination, in particular the Knuth-Bendix and polynomial orderings.

First, two methods for generating termination orderings are enhanced. The Knuth-Bendix ordering algorithm incrementally generates numeric and symbolic constraints that are sufficient for the termination of the rewrite system being constructed. The KB ordering algorithm requires an efficient linear constraint solver that detects the nature of degeneracy in the solution space, and for this a revised method of complete description is presented that eliminates the space redundancy that crippled previous implementations.

Polynomial orderings are more powerful than Knuth-Bendix orderings, but are usually much harder to generate. Rewrite systems consisting of only a handful of rules can overwhelm existing search techniques due to the combinatorial complexity. A genetic algorithm is applied with some success.

Second, a subset of the family of polynomial orderings is analysed. The polynomial orderings on terms in two unary function symbols are fully resolved into simpler orderings. Thus it is shown that most of the complexity of polynomial orderings is redundant. The order type (logical invariant), either τ or λ (numeric invariant), and precedence is calculated for each polynomial ordering. The invariants correspond in a natural way to the parameters of the orderings, and so the tabulated results can be used to convert easily between polynomial orderings and more tangible orderings.

The orderings of order type ω^ω are two of the recursive path orderings. All of the other polynomial orderings are of order type ω or ω^2 and each can be expressed as a lexicographic combination of τ (weight), λ (matrix), and lexicographic (dictionary) orderings.

The thesis concludes by showing how the analysis extends to arbitrary monadic terms, and discussing possible developments for the future.

Acknowledgements

I would like to thank my supervisor, Ursula Martin, for guiding me through the intractabilities of termination and for the many stimulating and insightful discussions. I am very grateful to the termination community, particularly the participants of the termination workshops, with particular thanks to Joachim Steinbach, Jürgen Giesl, and Elizabeth Scott.

It was a pleasure to study in the Computer Science Division, and I would like to thank my colleagues in St Andrews and York who supported this work.

I must also take this opportunity to thank my friends and family who have been so tolerant during this indulgent but immensely rewarding period of my life.

Contents

List of Tables

List of Figures

Notation

Abbreviations	
a.k.a.	also known as
cf.	compare
e.g.	for example
etc.	and so on
i.e.	that is
iff	if and only if
resp.	respectively
s.t.	such that
wlog	without loss of generality
w.r.t.	with respect to

Page refs for symbols	
$\doteq$	5
$\twoheadrightarrow$	22
$\rightarrow$	22
$\succ_{pol}$	74
$\succsim_{\tau}$	78
$\succsim_{\lambda}$	79
$\rhd^{\min}_{\max}$	81
$\overset{d}{\blacktriangledown}$	83

0 Introduction

In this chapter we describe what is meant by 'termination', set a context for the subsequent discussion, and give an overview of the document.

0.0 Terminating Processes

Let us begin by considering the progress of a hypothetical slide show. Suppose the projectionist has a number of boxes of slides, each box containing slides numbered sequentially from 0. For this viewing the projectionist will present one box of slides, projecting the slides in reverse order: $s_0, (s_0 - 1), \ldots, 1, 0$. Before the first slide is projected, all the audience knows regarding running length is that the first slide will be indexed by a positive whole number, that each successive index will be smaller, and that no slide has a negative index. Thus the audience deduces that only a finite number of slides will be shown and therefore the viewing must eventually terminate.

If the projectionist promises that no slide will be projected for longer than some upper bound, five minutes say, then as soon as the index of the first slide (s_0) is known, an upper bound ($(s_0 + 1) \times 5$ minutes) for the entire viewing is known. This upper bound holds even if the projectionist omits some slides from the box.

Now suppose that the projectionist intends to present 2 boxes of slides: $s_1, (s_1 - 1), \ldots, 1, 0, s_0, (s_0 - 1), \ldots, 1, 0$. The audience can no longer place an upper bound on the length of the viewing, even after the index of the first slide is known. It is not until the slides of the second box are projected

that 'time until termination' can be bounded. Nevertheless, the audience can still characterise the terminating nature of the viewing; the 'double box' viewing is simply the sequential composition of two versions of the 'single box' viewing. Logicians say the 'single box' sequence (natural numbers under the usual ordering) has 'order-type' ω. (This terminology will be formally defined in Chapter 1.) The order-type of the 'double box' sequence $(\cdots > 2 > 1 > 0 > \cdots > 2 > 1 > 0)$ is $\omega + \omega$, written $\omega.2$.

Finally suppose that the projectionist will present an unspecified number of boxes of slides, but that each box is indexed in the same manner as the slides. Before arriving at the viewing, the audience are ignorant not only of the number of slides in each box, but also of the number of boxes. The sequence of boxes will have order-type ω, and the slides of each box will have order type ω, so the termination nature of the viewing can be characterised by $\omega.\omega$, written ω^2. (Although these order-types are daunting for the projectionist's audience and yet 'tiny' in comparison to those sometimes used in termination practice, they are roughly as large as we will find useful in the sequel.)

In 1949 Alan Turing used order-types to prove the termination of a section of computer program, represented in a modern programming language in Figure 0.0. Turing used

$$\text{“}\omega^2.(n-r)+\omega.(r-s)+k\text{”}$$

as a sequentially decreasing expression to demonstrate its termination.[0]

Noticing that the expression pair $(n-r, r-s)$ follows the same sequence as the pair (*box-index*, *slide-index*) from the example above, we can conclude that the termination of the program is also characterised by the order-type

[0]Turing suggested 2^{40} could be substituted for ω since that was an upper bound for the variables the particular computer could represent. The variable k was due to an expression that fluctuated from line to line in the original representation.

```
r := 1; u := 1; v := 1;
while r < n do
  begin
    s := 1;
    while s <= r do
      begin
        u := u + v;
        s := s + 1
      end;
    r := r + 1;
    v := u
  end
```

Figure 0.0: A nested loop (in Pascal)

ω^2. In abstracting away the slides and Pascal code, we can focus on the expressions that characterise the advancement towards termination. If we can prove that such an expression follows (or is contained in) a decreasing sequence that cannot decrease indefinitely, we have proved termination.

0.1 Termination in Practice

Many computer systems in use produce meaningful data only in the event of the computation reaching a conclusion. A common means of proving a procedure correct is to first prove partial correctness: if the procedure terminates then its result will be correct with respect to its specification; and then to prove that the procedure will indeed terminate given any valid input. Even intentionally non-terminating systems (e.g. operating systems) that are designed to run indefinitely are frequently designed to spawn sub-processes whose individual termination can be crucial to the functionality of the overall system. In a parallel computation the non-termination of a single thread may be enough to block the overall result. In a sense useful computation is only achieved via termination.

In practice, however, much computation is achieved without the developers being aware of a theory of termination as such. Single-threaded programs are often assured terminating using little more than the well-foundedness of the natural numbers. Multi-threaded programs can employ techniques such as time-outs to 'pull the plug' on tardy processes.

For safety-critical software a degree of termination assurance can be imposed by the project manager. Part of the design rationale for the SPARK ADA language ([CJM+92]) is to make termination arguments clearer by banning certain 'dangerous' constructs such as `gotos` that are in the full ADA language. Currie's NewSpeak ([Cur89]) goes all the way by excluding unbounded loops entirely, at the cost of Turing completeness.

One application area targeted by such languages is hard real-time systems, where satisfying temporal constraints is as important as functional correctness. For such software it is extremely desirable to have at compile time bounds on both its space and time requirements. This necessitates bounds on depth of calls to sub-programs as well as bounds on all program loops. Whereas NewSpeak achieves this via a restricted syntax, SPARK ADA is a strictly more expressive language and instead aims to facilitate proof that requirements are met. Hence such systems require the proof of termination to be strengthened by proof that the components terminate within given time bounds.

In [Tur95] Turner also advocates the preclusion of non-terminating computations, this time in the context of functional programming. By syntactically guaranteeing all functions be terminating (primitive recursive), Turner presents what he claims to be the first truly functional programming language (since, for example, referential transparency relies on all values being defined).

evaluation, so a functional program may be evaluated as a non-deterministic term rewriting system.

These are approaches designed to ease the task of avoiding non-termination in certain computer systems, but as the systems become larger and more complex, and the potential costs of system failure rise, it becomes increasingly important to prove the correctness of the system. In turn we need more powerful and sophisticated theorem provers, which need to be partially automated, and whose processes therefore need to be proved terminating. Here we are concerned with the term rewriting 'equational reasoning engine' for automated theorem proving.

0.2 Term Rewriting

Equations play an important part in the mathematical sciences. They are used, for example, to specify algebraic properties of data structures. We may want to specify that all the stacks in a computer system have the property $\text{pop}(\text{push}(e, s)) \doteq s$, for instance.[1] The set of such defining equations for a data type form the axioms of the data type's equational theory. Part of the process of verification, where the specification is examined to establish if it captures those properties that were intended (and does not entail properties that are undesired), involves checking that equations identified as being important to the system being specified can be derived from the defining equations. For example, after defining the `list` data type we may wish to verify that they have the property $\text{reverse}(\text{reverse}(l)) \doteq l$. Formal reasoning about specifications is just one example where we wish to employ automation to check that propositions are theorems of an equational theory.

In [Bir35] Birkhoff showed that the rules

[1] We will use $\doteq$ for *defined* equality, to avoid confusion with identity, $=$.

$$\frac{}{t \doteq t} \text{ (reflexivity)} \qquad \frac{t \doteq u}{u \doteq t} \text{ (symmetry)} \qquad \frac{t \doteq u \quad u \doteq v}{t \doteq v} \text{ (transitivity)}$$

$$\frac{t \doteq u}{f(\ldots t \ldots) \doteq f(\ldots u \ldots)} \text{ (context)} \qquad \frac{t \doteq u}{t\sigma \doteq u\sigma} \text{ (substitution)}$$

are complete for first order (universally quantified) equational reasoning; the equations derivable from a given set of equational axioms are exactly those equations that hold in all models of the axioms. (We focus our attention on first-order universally quantified clauses; much of what will be discussed applies to the various extensions that exist.)

Using these derivation rules, a proposition $lhs \doteq rhs$ is a theorem of a given equational theory if the left-hand side can be rewritten to the right-hand side by applying the derivation rules to the axioms, so that a derivation chain $lhs \doteq lhs' \doteq \cdots \doteq rhs$ can be formed.

As it stands, the process just described cannot be automated effectively. First, the derivation rules of reflexivity and symmetry can prevent any progress being made, $lhs \doteq lhs' \doteq lhs \doteq lhs' \doteq lhs' \doteq \cdots$. Second, there is no goal-direction for the next link in the derivation chain, so a less trivial chain may look like it is making progress, but it may never reach the right-hand side, $lhs \doteq lhs' \doteq lhs'' \doteq \cdots$. That is, the search tree is too wide and too deep.

To tackle these problems, each axiom $l_i \doteq r_i$ can be turned into a 'directed equation', $l_i \twoheadrightarrow r_i$, known as a *rewrite rule*. The set of axioms $\{l_i \twoheadrightarrow r_i\}$ now defines the rewrite relation $\rightarrow$ as the smallest transitive binary relation containing $\twoheadrightarrow$ and closed under context and substitution,

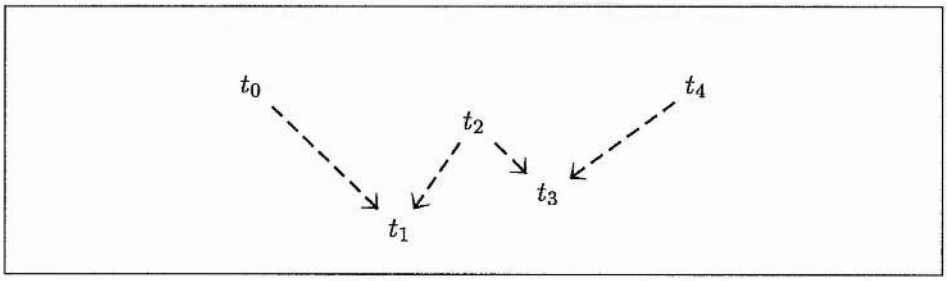
$$\frac{t \rightarrow u \quad u \rightarrow v}{t \rightarrow v} \text{ (transitivity)}$$

$$\frac{t \to u}{f(\ldots t \ldots) \to f(\ldots u \ldots)} \text{ (context)} \qquad \frac{t \to u}{t\sigma \to u\sigma} \text{ (substitution)}$$

If there exists a chain $lhs \to lhs' \to \cdots \to rhs$ then $lhs \doteq rhs$ is a theorem of the theory being examined. Therefore the set of rewrite rules $\twoheadrightarrow$ provides a semi-decision procedure for the original equational theory; if rewriting relates two terms then those terms are equationally related, but the converse does not necessarily hold.

The rewriting process may be proved terminating by showing that the rewrite relation $\to$ is well-founded, i.e. it admits no infinite chains. If the process is terminating it can be fully automated: apply rewrite rules to *lhs* and *rhs* until no more rewrites can be applied, producing terms *lhs'* and *rhs'*; if $lhs' = rhs'$ then the theorem $lhs \doteq rhs$ is proved.

A terminating rewrite system[2] solves the problem of depth since all chains are finite. However, two problems remain. The search tree may still be too wide; a given term may rewrite to many different final forms (called *normal forms*) and at each rewrite the system has no way of determining which possible chain may be most fruitful. Second, by making the axioms mono-directional the rewrite procedure may have lost some of the power of the original equational system. In their seminal paper [KB67] Knuth and Bendix showed that these problems were one and the same, as represented schematically below.

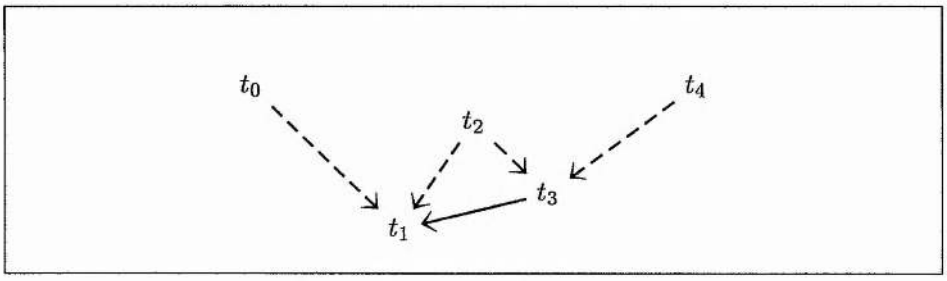


We would like to prove that $t_0 \doteq t_4$ is a theorem, but the rewriting of t_0 and

[2]For the purposes of this text, *rewrite/rewriting system* is synonymous with 'rule set'.

t_4 stop at t_1 and t_3 respectively; the rewrite relation is 'missing' the ability to relate t_1 and t_3 directly.

The Knuth-Bendix completion procedure ([KB67]) takes a terminating rule set and detects pairs of terms t, t' such that $t \doteq t'$ but which cannot be related directly. A new rule, either $t \twoheadrightarrow t'$ or $t' \twoheadrightarrow t$, is added to the system if the termination property can be maintained. This is represented on the diagram as



where $t_3 \twoheadrightarrow t_1$ is the new rule to be added. The augmented system containing the new rule is then tested for more rules to be added. If this procedure of adding pairs terminates then the augmented system is confluent: any two chains from a given term re-join at a common descendent. Since the property of termination has been maintained, any such system is convergent: all terms have unique normal forms. Thus, if successful, completion provides a complete decision procedure since two terms are equal if and only if their normal forms are the same.[3]

As described in the following section, termination of term rewriting systems is usually proved by finding a terminating relation (termination ordering) that includes the rewrite relation. For the purpose of a completion procedure, there are several important aspects to the termination ordering used.

First, the ordering has to contain each term pair in the rule set. This is

[3]That is, identical up to renaming of (universally quantified) variables.

unitary termination: all term pairs may be examined before any constraints are placed on a termination ordering. To be of practical use for a completion procedure, minimal time should be required to determine whether a particular class of orderings (e.g. Knuth-Bendix orderings) is able to correctly orient each of the rewrite rules. If a class of orderings is unsuccessfully applied, another class (e.g. polynomial orderings) may be tried. A completion procedure often generates many term pairs and so the ordering technique may be required to consider many sets of term pairs. Part of the reason that term rewriting is increasingly popular as an efficient proof mechanism is because current termination techniques are often able to fulfill these requirements. However, if we are to increase the range and speed of rewrite-based automated theorem provers we need to look at how termination techniques can be made more effective.

Second, as completion proceeds the rule set will in general grow, potentially to an unlimited number of term pairs, with new rules being integrated one rule at a time. This is *incremental* termination: the termination ordering has already been constrained before the new term pairs are available for consideration. It is clearly desirable to employ an incremental termination technique; one that can minimally augment its constraint data to integrate the new constraints rather than throw away previous data to start from scratch.

Third, the termination technique may be frequently required to consider the possible termination of $R \cup \{t \twoheadrightarrow u\}$ and then $R \cup \{u \twoheadrightarrow t\}$ to determine whether neither, one, or both orientations are possible. A termination technique that is able to minimally augment its existing data when integrating a new term pair, rather than throwing the data away and starting afresh, is described as *incremental*. It is also desirable if the termination technique

employed can re-use information from the first orientation instead of having to start afresh with the second orientation. More difficult in general is the ability to optimally decrement the constraints, so that if a rule is removed from the rule set the termination technique isn't unnecessarily constrained. It is usually more practical either to proceed with the unnecessary constraints or to restart the process of proving the current system terminates.

Fourth, different orderings may orient candidate term pairs in opposing directions (or allow either orientation). Whichever orientation is taken when the term pair becomes a new rewrite rule may affect the success of the completion procedure. Orienting a rule in one direction may cause the completion procedure to generate infinitely many new rules when orienting in the other direction may have led to a finite convergent rewriting system (see for example [Les86]).

Finally, the orientation of rules can affect the efficiency of the resulting rewrite relation, meaning that the number of rewrite steps applied to a given term may be different in two semantically equivalent rewriting systems. It may be possible to attribute upper bounds to the derivations of the rewriting system knowing properties of the containing termination ordering (its order type).

0.3 Proving Termination

Any (non-random) terminating process progresses 'closer' to termination. The task of proving termination is to find the appropriate 'measure'. If a rewrite relation[4] is terminating then there exists a well-founded rewrite relation that contains it: for example, the rewrite relation itself. Thus the

[4]Recall that a rewrite relation is a transitive relation closed under context and substitution.

task of proving termination of a rewrite relation $\rightarrow$ defined by $\twoheadrightarrow$ is to find a rewrite relation $\rightarrow'$ known to be well-founded such that $\rightarrow \subseteq \rightarrow'$. The relation $\rightarrow'$ is called a termination ordering, and is usually written $\succ$. From the formulation of $\rightarrow$ it suffices to show that $\twoheadrightarrow' \subseteq \succ$, where $\twoheadrightarrow'$ is the (possibly infinite) set of rewrite rules derived from $\twoheadrightarrow$ by instantiating all variables with ground terms.

The problem of proving termination of a rewriting system can be reduced to the halting problem and so is undecidable, even for one-rule systems with only unary function symbols. Therefore the standard approach is try a certain family of orderings, and if unsuccessful to find a member suitable for proving termination, try a different family of orderings. In the literature there appear formulations for many families of termination orderings (for example, Knuth-Bendix orderings [KB67, Mar87], polynomial orderings [MN70, Lan75] [Ste94] [BL87b, CL92], recursive path orderings [KL80], transformation orderings [BL87a], subterm path orderings [Pla78], and recursive decomposition orderings [Les84]).

These families of orderings can be roughly grouped into two classes: syntactic orderings and semantic orderings. A syntactic ordering compares terms by the syntactic structure of the term tree, typically examining first the function symbol at the root and then recursively examining the sub-trees. The task of proving termination by such an ordering is to find an appropriate precedence: the ordering on function symbols. The recursive path orderings (formulated on p 27) are an important family of syntactic orderings. Steinbach examined a variety of such orderings in [Ste88] and found that many seemingly disparate families produced the same orderings when they were total (i.e. sufficiently defined to compare all terms).

A semantic ordering first interprets terms into a well-founded strictly

monotonic algebra[5] $(A, >_A)$ and then compares the resulting elements according to $>_A$. The task of proving termination by such an ordering is to find an appropriate interpretation of the function symbols so that the interpreted terms are decreasing in the underlying algebra:

$$\begin{array}{ccc} t & \succ & u \\ \Downarrow & & \Downarrow \\ [\![t]\!] & >_A & [\![u]\!] \end{array}$$

We will concentrate on two families of so-called 'semantic' orderings: Knuth-Bendix orderings and polynomial orderings. At the heart of a Knuth-Bendix ordering (formulated on p 26) is a weight ordering: function symbols are assigned non-negative weights, and terms are oriented by comparing weights. This is extended to non-ground terms by assigning a positive weight to all variables and requiring the multiset of variables in a term to be a submultiset of all terms greater than that term. To compare terms with equal weights, the ordering is supplemented with a precedence (ordering on the function symbols) so that equal-weight terms are compared by their root symbols and then their subterms.

Knuth-Bendix orderings are popular due to their relative simplicity. When manually looking for an appropriate ordering it is often possible to make a good judgement by inspection alone as to whether to try the Knuth-Bendix family. Similarly for an automated tool, it is computationally easy to derive the constraints that a feasible KB ordering would need to satisfy. Indeed, Martin gave a complete decision procedure in [Mar87] to determine whether or not a given rewriting system is KB terminating. For these reasons, KB orderings are appropriate as the 'first hammer' when seeking to prove termination. Since their power is limited (due to their 'flat' interpretation of

[5] An algebra in which all operators are strictly monotonic increasing.

terms and the restriction on occurrences of variables) there need to be more sophisticated ordering families in the line of attack.

A polynomial ordering is defined by assigning polynomial functions over a domain, $\mathbb{N}$ say, to each function symbol. (See p 53 for an example.) For ground terms this is like a weight ordering, and non-ground terms are compared by examining whether one dominates the other over a sub-domain that includes the values of all ground terms.

The search space for polynomial orderings is far larger than that for Knuth-Bendix orderings, and the relationship between the interpretation of function symbols and the resulting orientation of terms is much less intuitive. The hunt for a suitable polynomial ordering usually takes the form of

- restrict the form of polynomials to be considered, then
- search the polynomials of that form until a successful combination of interpretations is found.

Use of the first step means that the rewriting system may be polynomial terminating but not under the subset of interpretations being considered. Steinbach in [Ste94] identified a class of polynomial interpretations that are successful for the majority of examples tried and for which he was able to give a decision procedure. Giesl in [Gie95a] gave a simpler decision procedure based on a method of Lankford ([Lan79]) which he showed to be equivalent to that of Steinbach. In Chapter 3 we will use the decision procedure of Giesl coupled with a population-based search technique (genetic algorithms) to provide another tool for polynomial termination.

0.3.0 Monadic Terms

Monadic terms are constructed from unary function symbols, and so are isomorphic to strings. For example, we can swap between thinking of $\mathtt{f}(\mathtt{f}(\mathtt{g}(x)))$ and $\mathtt{ffg}$.

Finding an interpretation such that the defined ordering contains all pairs of the rewrite relation is far from trivial for general terms (see [BL87b, Ste94]). However, we will see that, on monadic terms at least, polynomial orderings are much simpler than we might suppose. In fact we are able to classify them in terms of certain invariants explained below.

Although polynomial orderings on monadic terms are in only one variable, there are unboundedly many parameters to set, and the author expected to see a great variety of orderings. In addition, it was unclear how the parameters of the interpretations related to the resulting ordering. Not only has much of the complexity of the interpretations proved redundant, but the properties of each ordering follow naturally and simply from the few significant parameters.

Knowing the properties of the available orderings is important when selecting an ordering family in an attempt to prove a rewrite system is terminating. Different families of orderings may have orderings in common, for example the recursive path orderings are common to several families. Indeed several independent formulations for recursive path orderings have been shown to be equivalent ([Les81]). Even with respect to a single formulation, an ordering may have unboundedly many definitions. For example, if a Knuth-Bendix ordering is defined by the weight assignment $\{\mathtt{wta} = w_1,\ \mathtt{wtb} = w_2\}$ then, since the relative rather than absolute weights are significant, the same ordering is defined by $\{\mathtt{wta} = pw_1,\ \mathtt{wtb} = pw_2\}$ where p is any positive real. It is clearly the ratio $\tau = \frac{\mathtt{wtb}}{\mathtt{wta}}$ that matters.

On the other hand, by an appropriate choice of weights we can produce continuum many distinct Knuth-Bendix orderings, as shown in [Mar93]. Clearly it is unsatisfactory to classify orderings solely by their formulation family, and we look to more fundamental characterisations of orderings: ordering invariants.

0.3.1 Order Invariants

Two ordered sets are order-isomorphic if there is an order-preserving bijection between them. Just as cardinality is an abstraction over size, order type is an abstraction over order-isomorphism. Every well-founded total ordering (well-ordering) is order-isomorphic to an ordinal – its order type (logical invariant). More than simply a coarse tool for separating orderings, order types provide a logical measure of the reduction 'power' of total orderings. If a well-ordering $\succ$ with order type θ contains a rewrite system $\mathcal{R}$ (thus proving $\mathcal{R}$ is terminating) then θ can be related to the derivation complexity of $\mathcal{R}$ (see [Cic90, Hof92]), which in turn can be related to the proof theoretic and algorithmic complexities of the relation being computed by $\mathcal{R}$ (see [Wai93]).

In [MS93] detailed results are obtained for simple conditions determining the order types of total termination orderings on binary strings. Martin and Scott show that any total termination ordering on strings in two letters, say $\mathtt{a}$ and $\mathtt{b}$ with $\mathtt{a} \succ \mathtt{b}$, has order type ω, ω^2, or ω^ω, according to Table 0.0. In addition they show that the only such orderings of order type ω^ω are the recursive path orderings.

Once the order type of an ordering $\succ$ is known, we can further refine the analysis. For order type ω^ω the particular recursive path ordering can be identified (almost trivially since there are only four possibilities). For order type ω the numeric invariant is a real number $0 < \tau \leqslant 1$ that identifies the

Table 0.0: Order types of total simplification orderings on binary strings.

Conditions		Order Type
$b^j \succ a$	for some $j \in \mathbb{N}$,	ω
$a \succ b^j$	for all $j \in \mathbb{N}$ and	ω^2
both $b^k a \succ ab$	for some $k \in \mathbb{N}$	
and $ab^k \succ ba$	for some $k \in \mathbb{N}$,	
$a \succ b^j$	for all $j \in \mathbb{N}$ and	ω^ω
either $ab \succ b^k a$	for all $k \in \mathbb{N}$	
or $ba \succ ab^k$	for all $k \in \mathbb{N}$.	

particular weight pre-order $\succsim_\tau$ such that $\succ = \langle \succsim_\tau; \ldots \rangle$. For order type ω^2 the numeric invariant is a real number $\lambda > 0$ that identifies the particular matrix pre-order $\succsim_\lambda$ such that $\succ = \langle \succsim_\tau; \succsim_\lambda; \ldots \rangle$ (where $\tau = 0$). These pre-orders are considerably easier to work with than polynomial orderings, and are detailed in Section 4.2. One of the surprising results of this work is that for almost all polynomial orderings on monadic terms in two function symbols we are able to describe the orderings completely: they are the extensions of these pre-orders with the standard lexicographic orderings from the right ($\rhd^{\mathsf{lexR}}$).

0.3.2 Polynomial Analysis

In Section 4.3.0 the main results of the polynomial analysis are presented in three parts: the order types, the numeric invariants, and the lexicographic combination equivalents. These may be summarised as follows.

Let $\succ_{pol}$ be a polynomial ordering on monadic terms $\mathcal{T}(\{f, g\}, \{v\})$ defined by the interpretations

$$[\![f]\!](x) = a_m x^m + \cdots + a_1 x + a_0, \qquad a_m \ldots a_0 \geqslant 0,\ m \geqslant 1,$$

$$[\![g]\!](x) = b_n x^n + \cdots + b_1 x + b_0, \qquad b_n \ldots b_0 \geqslant 0,\ n \geqslant 1,$$

Table 0.1: Polynomial orderings on binary monadic terms (summary).

	Parameters	Lexicographic Combination	Order Type	Numeric Invariant
$\succ_{polA}$	$m \geqslant n > 1$	$\langle \succsim_\tau ; \rhd^{\mathsf{lexR}} \rangle$	ω	$\tau = \frac{\ln n}{\ln m}$
$\succ_{polB}$	$m > n = 1, \quad b_1 > 1$	$\langle \succsim_\tau ; \succsim_\lambda ; \rhd^{\mathsf{lexR}} \rangle$	ω^2	$\lambda = m$
$\succ_{polC}$	$m > n = 1, \quad b_1 = 1, \quad b_0 > 0$	rpo	ω^ω	–
$\succ_{polD}$	$m = n = 1, \quad a_1 \geqslant b_1 > 1$	$\langle \succsim_\tau ; \rhd^{\mathsf{lexR}} \rangle$	ω	$\tau = \frac{\ln b_1}{\ln a_1}$
$\succ_{polE}$	$m = n = 1, \quad a_1 > b_1 = 1, \quad b_0 > 0$	$\langle \succsim_\tau ; \succsim_\lambda \rangle$	ω^2	$\lambda = a_1$
$\succ_{polF}$	$m = n = 1, \quad a_1 = b_1 = 1, \quad a_0 \geqslant b_0 > 0$	$\succsim_\tau$	ω	$\tau = \frac{b_0}{a_0}$

and $[\![\nu]\!](x) = x$, such that $\mathsf{f}(\nu) \succ_{pol} \mathsf{g}(\nu) \succ_{pol} \nu$. The set of all such orderings is partitioned into subsets $\succ_{polA}$, $\succ_{polB}$, $\succ_{polC}$, $\succ_{polD}$, $\succ_{polE}$, and $\succ_{polF}$, and the properties of the orderings are summarised in Table 0.1.

Thus we see that almost all the properties of the ordering are determined by the degree and coefficient of the leading monomial. Moreover, apart from two recursive path orderings, all the effectiveness of polynomial orderings can be obtained using linear interpretations only. These theoretical results tie in nicely with the experimental results of Steinbach in [Ste94] where he found a significant proportion of term rewriting systems orderable by general polynomial orderings could also be ordered by his so-called simple-mixed polynomial orderings.

We can illustrate some of the results of this work by examples:

0. $[\![\mathsf{f}]\!](x) = 3x^4 + 2x^2 + 1 \quad$ and $\quad [\![\mathsf{g}]\!](x) = 5x^3 + x^2 + 9x$

   From row $\succ_{polA}$ we see that this is simply a weight ordering, with f having weight $\ln 4$ and g having weight $\ln 3$, extended with the lexicographic ordering having $\mathsf{g} \rhd \mathsf{f}$.

1. $[\![\mathsf{f}]\!](x) = x + 3 \quad$ and $\quad [\![\mathsf{g}]\!](x) = x + 2$

   This is also a weight ordering, as shown in row $\succ_{polD}$, with f and g

having weights 3 and 2 respectively.

2. $[\![\mathsf{f}]\!](x) = 4x + 1 \quad$ and $\quad [\![\mathsf{g}]\!](x) = x + 2$

   This is a so-called λ ordering (defined in Section 4.2) as shown in row $\succ_{polE}$. It has order type ω^2 and in this case $\lambda = 4$. In fact the leading coefficient of $[\![\mathsf{f}]\!](x)$ is the only significant parameter.

3. $[\![\mathsf{f}]\!](x) = x^2 + 4x + 1 \quad$ and $\quad [\![\mathsf{g}]\!](x) = x + 2$

   This is the standard (i.e. from the left) recursive path ordering (rpo) with f greater than g in the precedence. In fact we will see in Section 4.3.0 that the same ordering is given whenever the interpretation of g has leading monomial x (i.e. $[\![\mathsf{g}]\!]$ is strongly linear) and the leading index of the interpretation of f is greater than 1 (i.e. $[\![\mathsf{f}]\!]$ is non-linear).

4. $[\![\mathsf{f}]\!](x) = 3x^4 \quad$ and $\quad [\![\mathsf{g}]\!](x) = 5x + 1$

   This is also a λ ordering, from row $\succ_{polB}$, even though the interpretations are of completely different form to those in example 2. In fact it is exactly the same ordering with a lexicographic extension! If we use the ordering in example 2 extended with a lexicographic ordering having $\mathsf{g} \rhd \mathsf{f}$ then the interpretations in this example are completely redundant.

0.4 Document Structure

In Chapter 1 the preliminary definitions and theory are presented. Chapter 2 discusses the incremental Knuth-Bendix ordering algorithm and its constraint satisfaction engine, the method of complete description. The problems with the existing MCD are highlighted, and a revised version is presented, the Revised MCD. In Chapter 3 the problem of finding suitable polynomial orderings is addressed. Genetic algorithms are proposed as an efficient

search engine for such orderings, and a tailored GA is presented. Chapter 4 takes a more abstract stance, investigating the polynomial orderings definable on terms constructed from two unary function symbols. Each such ordering is shown to be a composition of simple orderings. We conclude in Chapter 5 with a summary of the results and a discussion of future developments.

1 Preliminaries

This chapter introduces most of the notation and concepts required for subsequent chapters. Fuller accounts of term rewriting and termination can be found in [DJ90, Klo87, Pla93, JL87].

1.0 Notation

The set of natural numbers $\{0, 1, 2, 3, \ldots\}$ is denoted $\mathbb{N}$ and the set of positive natural numbers $\{1, 2, 3, \ldots\}$ is denoted $\mathbb{N}_+$. The set of real numbers (resp., positive real numbers) is denoted $\mathbb{R}$ (resp., $\mathbb{R}_+$). If A is a set then $\mathcal{P}(A)$ denotes the power set of A. The cardinality of a set A is denoted $\#(A)$.

An ascending sequence $i, \ldots, j$ denotes the empty sequence if $j < i$, so $(a_1, \ldots, a_n)$ is the empty tuple if $n < 1$. For convenience $\vec{a}$ denotes a tuple $(a_1, \ldots, a_n)$ where n should be clear from the context. The set of n-length tuples over a set A is denoted A^n, and the set of non-negative-length tuples is denoted A^+.

1.1 Terms

Let $\mathcal{F}$ be a finite set of function symbols and $\mathcal{V}$ be a countable set of variables. Each function symbol f is associated with a natural number, its *arity* $\mathsf{ar}(f) \in \mathbb{N}$, signifying the number of arguments taken by the function. Function symbols of arity 0, 1, 2, and 3 are described respectively as *constant*, *unary*, *binary*, and *ternary*. In this document we will consider only finite sets of function symbols, each having fixed arity. Where convenient we will

reserve the symbols $\mathrm{f}, \mathrm{g}, \mathrm{h}, f, g, h, f_1, f_2, \ldots$ to denote function symbols and $v, x, y, z, v_1, v_2, \ldots$ to denote variables. (We assume an infinite supply of variables so that we never run out of new names when we come to renaming variables, but it is even more convenient to think of the set $\mathcal{V}$ as being finite.)

The set of *terms* $\mathcal{T}(\mathcal{F}, \mathcal{V})$ is the set of variables $\mathcal{V}$ closed under construction by $\mathcal{F}$ as $f(t_1, \ldots, t_n)$, where n is the arity of $f \in \mathcal{F}$. (If f is a constant then the empty brackets are elided.) The number of occurrences of a function symbol f in a term t is denoted $\#(f, t)$, and similarly for variables. The set of *ground* terms $\mathcal{T}(\mathcal{F}, \varnothing)$ is also denoted $\mathcal{T}(\mathcal{F})$, and the set of *non-variable* terms is the set of terms containing function symbols, $\mathcal{T}(\mathcal{F}, \mathcal{V}) \setminus \mathcal{V}$. A term is *monadic* if all its function symbols are unary.

A term $t = f(t_1, \ldots, t_n)$ has *immediate subterms* $t_1, \ldots, t_n$. The *proper subterms* of t are its immediate subterms and their proper subterms. A term is a (non-proper) subterm of itself. Specific subterms are located by their *position*, a sequence of positive naturals. The empty position locates the term itself, $t|_{\langle\rangle} = t$, a position i locates the i^{th} immediate subterm, $t|_{\langle i \rangle} = t_i$, and longer positions locate deeper subterms, $t|_{\langle i, j_1, \ldots, j_k \rangle} = t_i|_{\langle j_1, \ldots, j_k \rangle}$. The term produced by replacing the subterm of t at position p by the term u is denoted $t[u]_p$. A *substitution* $\sigma = \{v_1 \mapsto u_1, \ldots, v_m \mapsto u_m\}$ is a mapping from variables to terms, and $t\sigma$ denotes the result of simultaneously applying σ to all variables in t. A *ground substitution* $\sigma = \{v_1 \mapsto u_1, \ldots, v_m \mapsto u_m\}$ has no variables in the u_i. (Note that a ground substitution applied to a term may produce a non-ground term, e.g. $\mathrm{f}(x, y)\{x \mapsto 0\}$ still contains y.)

1.2 Term Rewriting

A *rewrite relation* is a transitive binary relation closed under context and substitution. A term rewriting *rule set* R is a set of term pairs $\{l_i \twoheadrightarrow r_i\}$

$(l_i, r_i \in \mathcal{T}(\mathcal{F}, \mathcal{V}))$. This set defines a rewrite relation $\rightarrow$, the smallest transitive relation[0] closed under context and substitution containing $\twoheadrightarrow$. A term $t \in \mathcal{T}$ *rewrites* to a term $u \in \mathcal{T}$ by a rule $l \twoheadrightarrow r$ in R if l matches a subterm of t, in which case the appropriate instantiation of r replaces that subterm of t to produce a term, u. In other words, $t \rightarrow u$ if $t|_p = l\sigma$ for some position p, substitution σ, and rule $l \twoheadrightarrow r$ in R, and $u = t[r\sigma]_p$.

For example, the rewrite system below (from [Der95]) converts a term to disjunctive normal form.

$$\begin{aligned}
\neg\neg x &\twoheadrightarrow x \\
\neg(x \vee y) &\twoheadrightarrow \neg x \wedge \neg y \\
\neg(x \wedge y) &\twoheadrightarrow \neg x \vee \neg y \\
x \wedge (y \vee z) &\twoheadrightarrow (x \wedge y) \vee (x \wedge z) \\
(y \vee z) \wedge x &\twoheadrightarrow (x \wedge y) \vee (x \wedge z)
\end{aligned}$$

Since we are concerned with the termination of rewriting, we consider only finite rule sets and therefore finite sets of function symbols. Also, the left-hand sides of rules are non-variable (i.e. $\mathcal{T} \setminus \mathcal{V}$) and the variables on the right-hand side of a rule are a subset of the variables on the left-hand side.

A term t is *normalised* to the term $\downarrow t$ w.r.t. R if $t \rightarrow t' \rightarrow \cdots \rightarrow \downarrow t$ and $\downarrow t$ cannot be rewritten further.[1] This is what we will mean in the sequel by term rewriting. To employ a term rewriting system R for automated rewriting, it is desirable to know that the term rewriting process cannot continue indefinitely on any term, i.e. that the rewrite relation contains no infinite chains $t \rightarrow t' \rightarrow \cdots$. This desire becomes a necessity when rewriting systems are used for (semi-)automated equational reasoning where

[0] The discussion is slightly simplified by ignoring the convention of making $\rightarrow$ reflexive.

[1] Note that, since the relation $\rightarrow$ is transitive, the description of multiple rewrites is solely for illustration.

a convergent rewriting system (one in which $\downarrow t$ is unique for each t) is to be used to determine equality.

1.3 Orderings

A binary relation $(A, \succ)$ is an *ordering*[2] iff $\succ$ is transitive and irreflexive. The ordering $\succ$ is *total* iff, for all distinct $s, t \in A$, either $s \succ t$ or $t \succ s$. The ordering $(A, \succ)$ is *well-founded* iff it contains no infinite descending chains $s \succ t \succ u \succ \cdots$.

A binary relation $(A, \succsim)$ is a *pre-order*[3] iff $\succsim$ is transitive and reflexive. A pre-order $\succsim$ defines an ordering (its strict part) by $\succ = \succsim \setminus \precsim$, and an equivalence relation by $\sim = \succsim \cap \precsim$. A pre-order is said to be *well-founded* iff its strict part (ordering) is well-founded. We can always obtain a pre-order from an ordering $\succ$ by taking its reflexive closure $\succeq$, in which case the equivalence relation is simply identity.

An ordering $\succ'$ is an *extension* of an ordering $\succ$ iff $\succ \subseteq \succ'$. A pre-order $\succsim'$ is an *extension* of a pre-order $\succsim$ iff $\succ \subseteq \succ'$ and $\sim \subseteq \sim'$.

An important means of defining a pre-order on terms is by interpretation into some well-founded strictly monotonic algebra, A, formulated as $t \succsim_A u$ iff $[\![t]\!] \geq [\![u]\!]$.

A common technique for building useful orderings from simple 'building blocks' is to take their lexicographic combination.

DEFINITION 0 (LEXICOGRAPHIC COMBINATION) Given a sequence of pre-orders $\succsim_1, \ldots, \succsim_r$ on a set S, their *lexicographic combination* is the relation $\langle \succsim_1; \succsim_2; \ldots; \succsim_r \rangle$,

[2] a.k.a. strict partial order.
[3] a.k.a. quasi-order.

where $\langle \succsim_1 \rangle = \succsim_1$ and $\langle \succsim_1; \succsim_2; \ldots; \succsim_r \rangle = \succ_1 \cup (\sim_1 \cap \langle \succsim_2; \ldots; \succsim_r \rangle)$ if $r > 1$. ◇

If one of the orderings $\succ_1, \ldots, \succ_r$ is total then the lexicographic combination of their pre-orders is an antisymmetric-order. Sometimes the equivalence part of the combination will be precluded by writing, for example, $\langle \succsim_1; \succsim_2; \ldots; \succsim_{r-1}; \succ_r \rangle$, which defines an ordering.

Thus a lexicographic combination is a sequential application of pre-orders, where the combination has the same domain as each of the constituent relations. This should not be confused with the lexicographic lifting of an ordering from terms to tuples of terms.

DEFINITION 1 (LEXICOGRAPHIC LIFTING (FROM THE LEFT)) Let $(\mathcal{T}, \succ)$ be a term ordering, let n be a fixed natural number, and let $\mathcal{T}^n$ be the set of n-tuples over $\mathcal{T}$. Then $(\mathcal{T}^n, \succ^{\mathsf{lexL}})$ is the *lexicographic lifting from the left* of $\succ$ to $\mathcal{T}^n$ defined by

$(t_1, \ldots, t_n) \succ^{\mathsf{lexL}} (u_1, \ldots, u_n)$ if

for some $1 \leqslant j \leqslant n$ we have $t_1 = u_1, \ \ldots, \ t_{j-1} = u_{j-1}, \ t_j \succ u_j$. ◇

The *lexicographic lifting from the right* is defined similarly. These can be generalised to an arbitrary (fixed) permutation of the elements of each tuple. Let π be a bijection on $\{1, \ldots, n\} \subset \mathbb{N}$. Then the *$\pi$-permutation* of $(a_1, \ldots, a_n)$ is $(a_{\pi(1)}, \ldots, a_{\pi(n)})$.

DEFINITION 2 (LEXICOGRAPHIC LIFTING (WITH PERMUTATIONS))
Let $(\mathcal{T}, \succ)$ be a term ordering, let n be a fixed natural number, let π be a bijection on $\{1, \ldots, n\}$, and let $\mathcal{T}^n$ be the set of n-tuples over $\mathcal{T}$. Then $(\mathcal{T}^n, \succ^{\mathsf{lex}\pi})$ is the *lexicographic lifting* (with respect to π) of $\succ$ to $\mathcal{T}^n$ defined by

$(t_1, \ldots, t_n) \succ^{\mathsf{lex}\pi} (u_1, \ldots, u_n)$ if

for some $1 \leqslant j \leqslant n$ we have $t_{\pi(1)} = u_{\pi(1)}, \ \ldots, \ t_{\pi(j-1)} = u_{\pi(j-1)}$,
$t_{\pi(j)} \succ u_{\pi(j)}$. ◇

Another popular lifting of orderings is the multiset lifting ([DM79, JL82]).

DEFINITION 3 (MULTISET LIFTING) Let $(\mathcal{T}, \succ)$ be a term ordering, and let $\mathcal{T}^n$ be the set of n-tuples over $\mathcal{T}$. Then $(\mathcal{T}^n, \succ^{\mathsf{mul}})$ is the *multiset lifting* of $\succ$ to $\mathcal{T}^n$ defined by
$(t_1, \ldots, t_n) \succ^{\mathsf{mul}} (u_1, \ldots, u_n)$ if
for all u_j there exists a t_i s.t. $t_i \succ u_j$. ◇

DEFINITION 4 (STATUS) The *status* function, $\mathsf{stat} : \mathcal{F} \to \{\mathsf{lexL}, \mathsf{lexR}, \mathsf{lex}\pi, \mathsf{mul}\}$, maps each function symbol to a label indicating the order in which its child terms should be considered.

We now define a class of orderings suitable for proving the termination of term rewriting systems: simplification orderings ([Der79, MZ94]).

DEFINITION 5 (SIMPLIFICATION ORDERINGS) Let $(\mathcal{T}, \succ)$ be an ordering. Then $\succ$ is closed under

context[4]	if $t \succ u$ implies $c[t]_p \succ c[u]_p$ for all $t, u, c \in \mathcal{T}$ and positions p of c,
substitution[5]	if $t \succ u$ implies $t\sigma \succ u\sigma$ for all $t, u \in \mathcal{T}$ and substitutions σ,
the subterm relation[6]	if $t[s] \succ s$ for all $t \in (\mathcal{T} \setminus \mathcal{V})$ and all proper subterms s of t.

A *rewrite ordering* is an ordering closed under context and substitution. A

[4] a.k.a. monotonic
[5] a.k.a. stable (under variable substitutions)
[6] a.k.a. has the subterm property

reduction ordering is a well-founded rewrite ordering. A *simplification ordering* is a reduction ordering closed under the subterm relation. ◇

1.3.0 Weight Orderings

A *weight* function $[\![\_]\!]_{wt} : \mathcal{F} \to \mathbb{N}$ associates each function symbol with a natural number. This mapping is lifted to terms by the obvious homomorphism $[\![f(t_1, \ldots, t_m)]\!]_{wt} = [\![f]\!]_{wt} + \sum_{i=1}^{m} [\![t_i]\!]_{wt}$. (The tag '$wt$' may be elided when the type of mapping is clear from the context.)

DEFINITION 6 (WEIGHT ORDERINGS) Let $[\![\_]\!]_{wt} : \mathcal{T}(\mathcal{F}) \to \mathbb{N}$ be a weight function such that at least one function symbol has non-zero weight. Then the *weight pre-order* $\succsim_{wt}$ is defined by

$$t \succsim_{wt} u \quad \text{if} \quad [\![t]\!]_{wt} \geqslant [\![u]\!]_{wt}$$

The *weight ordering* $\succ_{wt}$ defined by $[\![\_]\!]_{wt}$ is the strict part of $\succsim_{wt}$. A weight ordering is extended to non-ground terms $t, u \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ by

$$t \succ_{wt} u \quad \text{if} \quad t\sigma \succ_{wt} u\sigma \text{ for all ground substitutions } \sigma.$$

◇

1.3.1 Knuth-Bendix Orderings

DEFINITION 7 (KNUTH-BENDIX ORDERINGS) Let w be a positive natural number. Let $[\![\_]\!] : \mathcal{T}(\mathcal{F}, \mathcal{V}) \to \mathbb{N}$ be a weight function such that

0. the weight of all variables is w,
1. the weight of each constant symbol is at least w, and
2. at most one unary function symbol (f) has weight 0.

Let $\rhd$ be a precedence on $\mathcal{F}$ such that if there is a unary $\mathsf{f} \in \mathcal{F}$ with weight 0, then f is the maximum in $(\mathcal{F}, \rhd)$. Then the *Knuth-Bendix ordering* $\succ_{kb}$

is defined by

$t \succ_{kb} u$ if

for all $v \in \mathcal{V} : \#(v,t) \geqslant \#(v,u)$ and either

$[\![t]\!] > [\![u]\!]$ or

$[\![t]\!] = [\![u]\!]$ and $t = \mathsf{f}^n(v)$ and $u = v$ for some $n \geqslant 1$ and $v \in \mathcal{V}$, or

$$t = f\vec{t},\ u = g\vec{u} \text{ and } f \rhd g, \text{ or}$$

$$t = f\vec{t},\ u = f\vec{u} \text{ and } \vec{t} \succ_{kb}^{\mathsf{stat}(f)} \vec{u}. \qquad \diamond$$

1.3.2 Recursive Path Orderings

DEFINITION 8 Let $(\mathcal{F}, \rhd)$ be a precedence, and let each function symbol be associated with either lexicographic or multiset status. Then the *recursive path ordering* $\succ = \rhd^{\mathsf{rp}}$ is the lifting of $\rhd$ to terms $\mathcal{T}(\mathcal{F}, \mathcal{V})$ defined by

$t = f(t_1, \ldots, t_m) \succ g(u_1, \ldots, u_n) = u$ if

$t_i \succeq u$ for some $1 \leqslant i \leqslant m$,

or $f \rhd g$ and $t \succ u_j$ for all $1 \leqslant j \leqslant n$,

or $f = g$ and $(t_1, \ldots, t_m) \succ^{\mathsf{stat}(f)} (u_1, \ldots, u_n)$ $\diamond$

When a recursive path ordering as formulated above is defined on monadic terms it is known as a *left* recursive path ordering, denoted $\rhd^{\mathsf{rpL}}$. The *right* recursive path ordering, denoted $\rhd^{\mathsf{rpR}}$, on monadic terms is also important and is formulated as $t \rhd^{\mathsf{rpR}} u$ iff $\mathrm{rev}(t) \rhd^{\mathsf{rpL}} \mathrm{rev}(u)$, where

$$\mathrm{rev}(f_1(\ldots(f_n(v)))) = f_n(\ldots(f_1(v))).$$

1.3.3 Polynomial Orderings

DEFINITION 9 (POLYNOMIAL) Let S be a semi-ring (a set closed under addition and multiplication). Then a *monomial expression* $ax_1^{e_1} \ldots x_m^{e_m}$, where $a \in S$ is a *coefficient*, $e_i \in \mathbb{N}$ are *indices*, and x_i are variables, determines a

monomial function from S^m to S, the function obtained by evaluating the expression for each argument in S^m. A *polynomial function* is obtained similarly from a *polynomial expression*, a finite sum of monomial expressions. ◇

We will be considering polynomials with $S = \mathbb{N}$ and with $S = \mathbb{R}_+$.

Recall that a function g is said to be *monotonic increasing* if $x \leqslant x'$ implies $f(\ldots, x, \ldots) \leqslant f(\ldots, x', \ldots)$, and is said to be *strictly monotonic increasing* if $x < x'$ implies $f(\ldots, x, \ldots) < f(\ldots, x', \ldots)$.

DEFINITION 10 (POLYNOMIAL INTERPRETATION IN $\mathbb{N}$) A *polynomial interpretation* in $\mathbb{N}$ of a function symbol $f \in \mathcal{F}$ is a polynomial function $[\![f]\!]$ that has the same arity as f and is strictly monotonic increasing. ◇

The following definition is due to Dershowitz [Der79] and allows polynomial interpretations in the set of positive real numbers, even though the set is not well-founded, by requiring the interpretations provide the subterm relation, i.e. that $[\![f]\!](\ldots, x, \ldots) > x$ for all arguments of $[\![f]\!]$.

DEFINITION 11 (POLYNOMIAL INTERPRETATION IN $\mathbb{R}_+$) A *polynomial interpretation* in $\mathbb{R}_+$ of a function symbol $f \in \mathcal{F}$ is a polynomial function $[\![f]\!]$ that has the same arity as f, is (not necessarily strictly) monotonic increasing and has the subterm property. ◇

DEFINITION 12 (POLYNOMIAL INTERPRETATION OF TERMS) A *polynomial interpretation* of $\mathcal{F}$ is a set of polynomial interpretations, one for each $f \in \mathcal{F}$. Let $\mathcal{X}$ be a countable set of variables in one-to-one correspondence with $\mathcal{V}$. Then the *polynomial interpretation* of a term $f(t_1, \ldots, t_{\mathsf{ar}(f)}) \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ is

given by the homomorphism

$$[\![v_i]\!] = x_i$$

$$[\![f(t_1, \ldots, t_{\mathsf{ar}(f)})]\!] = [\![f]\!]([\![t_1]\!], \ldots, [\![t_{\mathsf{ar}(f)}]\!])$$

◇

Both of these definitions above can be extended to allow negative coefficients, but this creates the additional proof obligation that no ground term is interpreted smaller than μ, and it is not clear that there is any gain with this added complication, so we will consider only non-negative coefficients.

DEFINITION 13 (POLYNOMIAL ORDERING (A)) Let $[\![\_]\!]$ be a polynomial interpretation of $\mathcal{T}(\mathcal{F}, \mathcal{V})$ in $\mathbb{N}$. Then $t \succ_p u$ if

$[\![t\sigma]\!] > [\![u\sigma]\!]$

for all ground substitutions $\sigma : \mathcal{V} \to \mathbb{N}$. ◇

It may appear at first that to compare two terms we must calculate and substitute the interpretations of all terms. However, substituting ground terms for term variables and then calculating the interpretation is equivalent to calculating the (non-ground) interpretation and then evaluating the polynomial expression at all values of interpretations of ground terms.

$$\begin{array}{ccc} [\![t\{v_i \mapsto s_i\}]\!] & > & [\![u\{v_i \mapsto s_i\}]\!] \\ \Downarrow & & \Downarrow \\ [\![t]\!]\{x_i \mapsto [\![s_i]\!]\} & > & [\![u]\!]\{x_i \mapsto [\![s_i]\!]\} \end{array}$$

This means that if, for example, $\mathcal{F} = \{\mathsf{s}(\_), 0\}$ and $[\![\mathsf{s}]\!](x) = x+2$ and $[\![0]\!] = 2$ then we only need test for substitutions of positive even numbers. In practice it is simpler to test for a contiguous superset of $[\![\mathcal{T}(\mathcal{F})]\!]$ (e.g. $\{2, 3, 4, 5, \ldots\}$ above) since that is sufficient, leading to the following formulation.

DEFINITION 14 (POLYNOMIAL ORDERING (B)) Let $[\![\_]\!]$ be a polynomial interpretation of $\mathcal{T}(\mathcal{F},\mathcal{V})$ in $\mathbb{N}$ (resp. $\mathbb{R}_+$) as defined above such that $[\![t]\!] \geqslant \mu$ for all ground $t \in \mathcal{T}(\mathcal{F})$ and some $\mu \in \mathbb{N}$ (resp. $\mu \in \mathbb{R}_+$). Then $t \succ_p u$ if

$[\![t]\!](x_1,\ldots,x_n) > [\![u]\!](x_1,\ldots,x_n)$

for all $x_1,\ldots,x_n \geqslant \mu$. ◇

2 Knuth-Bendix Ordering Algorithm

In this chapter we look at the incremental Knuth-Bendix ordering algorithm, which is a complete decision procedure for whether a suitable Knuth-Bendix ordering exists for a given term rewriting system. A revised Method of Complete Description is proposed as an efficient feasibility engine for the algorithm.

2.0 Introduction

The Knuth-Bendix family of orderings (formulated in Definition 7 on p 26) has proved a popular and often effective means of proving termination of term rewriting systems ([KB67, Mar87, Ste94]). Being based on weight orderings, it is possibly the most intuitive class of 'useful' orderings, making it a common first choice for attempting a termination proof.

An algorithm for proving Knuth-Bendix termination is presented in [Mar87] and [DKM90]. The algorithm constructs a system of homogeneous linear inequalities, the numerical constraints necessary (but not sufficient) for the existence of a suitable Knuth-Bendix ordering. The ordering algorithm consults a linear programming engine to test whether the numerical constraints can be satisfied and to detect degeneracy (defined below). If the constraints cannot be satisfied then there is no suitable Knuth-Bendix ordering. Degeneracy being detected may also indicate failure or it may entail further symbolic or numeric constraints (depending on the nature of the degener-

acy).

The Method of Complete Description (MCD) is proposed in [DKM90] as an appropriate linear programming engine. Consisting of elementary operations, the MCD is an elegant and straightforward technique that gives precisely the information required by the ordering algorithm. Moreover, due to the incremental nature of the MCD itself, it fulfills the incremental potential of the ordering algorithm. However, as demonstrated in [Cro92]), the original MCD can be grossly inefficient due to its doubly-exponential space requirements. In the remainder of this chapter we examine techniques for making the MCD, and hence the incremental Knuth-Bendix ordering algorithm, able to handle moderate sized problems.

2.1 Preliminaries

This section introduces the mechanisms from linear algebra required to study the MCD. The reader is referred to linear programming books such as [Kre68, Dan63, AHU58, Gar60] for a more thorough introduction.

2.1.0 Vectors and Matrices

A vector $\mathbf{x}$ (over the real numbers) in n-space (a.k.a. an n-vector) is an n-tuple $(x_1, \ldots, x_n) \in \mathbb{R}^n$, usually written as a column vector,

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix},$$

and having as transpose a row vector,

$$\mathbf{x}^{\mathsf{T}} = \begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix}.$$

If $\mathbf{x}$ was a row vector, the orientations of $\mathbf{x}$ and $\mathbf{x}^{\mathsf{T}}$ would be interchanged. (Vectors are indicated by bold type, so that for example, $\mathbf{x_i}_j$ is the jth component of the ith vector.) Particular vectors are $\mathbf{0} = 0^n$, $\mathbf{1} = 1^n$, and $\mathbf{e_1} = (1, 0, \ldots, 0, 0)$, $\ldots$, $\mathbf{e_n} = (0, 0, \ldots, 0, 1)$. A point in n-space is used interchangeably with its position vector (w.r.t. the origin).

The usual ordering on real numbers is lifted to vectors in the following ways.

DEFINITION 15 (VECTOR ORDERING)

$$\begin{aligned}
\mathbf{x} \geq \mathbf{y} \quad &\text{iff} \quad x_i \geqslant y_i, \text{ for all } i \ (1 \leqslant i \leqslant n) \\
\mathbf{x} > \mathbf{y} \quad &\text{iff} \quad \mathbf{x} \geq \mathbf{y} \text{ and } x_j > y_j, \text{ for some } j \ (1 \leqslant j \leqslant n) \\
\mathbf{x} \gg \mathbf{y} \quad &\text{iff} \quad x_i > y_i, \text{ for all } i \ (1 \leqslant i \leqslant n)
\end{aligned}$$

The *inner product* of two n-vectors, $\mathbf{x}$ and $\mathbf{y}$, is the real number

$$\mathbf{x}.\mathbf{y} = x_1 y_1 + \cdots + x_n y_n.$$

An (m, n)-matrix (over the real numbers) is an array having m rows and n columns. We may sometimes consider such a matrix as having m rows of n-vectors

$$\begin{bmatrix} \mathbf{a_{11}} & \cdots & \mathbf{a_{1n}} \\ \vdots & \vdots & \vdots \\ \mathbf{a_{m1}} & \cdots & \mathbf{a_{mn}} \end{bmatrix} = \begin{bmatrix} \mathbf{a_1}^{\mathsf{T}} \\ \vdots \\ \mathbf{a_m}^{\mathsf{T}} \end{bmatrix}$$

and sometimes consider such a matrix as having n columns of m-vectors

$$\begin{bmatrix} \mathbf{a}_{11} & \cdots & \mathbf{a}_{n1} \\ \vdots & \cdots & \vdots \\ \mathbf{a}_{1m} & \cdots & \mathbf{a}_{nm} \end{bmatrix} = \begin{bmatrix} \mathbf{a_1} & \cdots & \mathbf{a_n} \end{bmatrix}$$

(Note that the vector denoted by, say, $\mathbf{a_1}$ is different according to the orientation we take of the matrix.) The *matrix product* of an (m,n)-matrix and an (n,p)-matrix is the (m,p)-matrix given by

$$\begin{bmatrix} \mathbf{a_1}^\mathsf{T} \\ \vdots \\ \mathbf{a_m}^\mathsf{T} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{b_1} & \cdots & \mathbf{b_p} \end{bmatrix} = \begin{bmatrix} \mathbf{a_1}.\mathbf{b_1} & \cdots & \mathbf{a_1}.\mathbf{b_p} \\ \vdots & & \vdots \\ \mathbf{a_m}.\mathbf{b_1} & \cdots & \mathbf{a_m}.\mathbf{b_p} \end{bmatrix}$$

The product of a matrix and a vector is defined by treating the vector as a matrix.

The length of a vector $\mathbf{x} = [x_1, \ldots, x_n]^\mathsf{T}$ is defined as $|\mathbf{x}| = \sqrt{x_1^2 + \cdots + x_n^2}$.

A ray is a vector of which only the direction (and not the magnitude) is significant, that is, $\mathbf{r}$ is identified with $a\mathbf{r}$ for all positive a.

2.1.1 Linear Inequalities

The system of m linear inequalities

$$\left.\begin{array}{rcl} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n & \geqslant & b_1 \\ \vdots & \geqslant & \vdots \\ a_{m,1}x_1 + a_{m,2}x_2 + \cdots + a_{m,n}x_n & \geqslant & b_m \end{array}\right\}$$

subject to $\mathbf{x} \geq \mathbf{0}$, can be represented as

$$\left.\begin{array}{l} A.\mathbf{x} \geq \mathbf{b}, \quad \text{subject to } \mathbf{x} \geq \mathbf{0} \end{array}\right\}$$

where A is the (m,n)-matrix having entries $a_{i,j}$ $(1 \leqslant i \leqslant m,\ 1 \leqslant j \leqslant n)$, $\mathbf{x} = [x_1, \ldots, x_n]^\mathsf{T}$, and $\mathbf{b} = [b_1, \ldots, b_m]^\mathsf{T}$.

The question of whether such a system is satisfiable can be represented as the problem: Given $A \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$, does there exist $\mathbf{x} \in \mathbb{R}^n$ such that $A.\mathbf{x} \geq \mathbf{b}$?

Using trivial arithmetic, the above system is equivalent to

$$\left.\begin{array}{l} A.\mathbf{x} - \mathbf{b} \geq \mathbf{0}, \quad \text{subject to } \mathbf{x} \geq \mathbf{0} \end{array}\right\}$$

and by introducing a dummy variable x_{n+1}, the above system of linear inequalities is equivalent to the homogeneous system of linear inequalities

$$\left.\begin{array}{rcl} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n - b_1x_{n+1} & \geqslant & 0 \\ \vdots & \geqslant & \vdots \\ a_{m,1}x_1 + a_{m,2}x_2 + \cdots + a_{m,n}x_n - b_mx_{n+1} & \geqslant & 0 \end{array}\right\}$$

subject to $\mathbf{x} \geq \mathbf{0}$ and $x_{n+1} = 1$. Equally trivial is the replacement of a constraint $kx_i \geqslant l$ by $y_i \geqslant 0$, where $x_i = (y_i + l)/k$. Thus we can assume that the systems of linear inequalities of our discourse are homogeneous and that all constraints on variables are non-negative.

2.1.2 Hyperplanes

Given a vector $\mathbf{v} \in \mathbb{R}^n$ and a scalar $a \in \mathbb{R}$, the set of points

$$\{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{v}.\mathbf{x} = a\}, \quad (\mathbf{v} \neq \mathbf{0})$$

is a *hyperplane*. An inequality $\mathbf{v}.\mathbf{x} \geqslant a$ defines a closed halfspace, and a strict inequality $\mathbf{v}.\mathbf{x} > a$ defines an open halfspace. The vector $\mathbf{v}$ is orthogonal to the hyperplane and points in the direction of the halfspace that satisfies the above inequalities. For convenience, when $a = 0$ we will identify the hyperplane with its orthogonal ray $\mathbf{v}$.

Given a system of linear inequalities $A.\mathbf{x} \geq \mathbf{b}$, each inequality $\mathbf{a_i}.\mathbf{x} \geqslant b_i$ $(1 \leqslant i \leqslant m)$ defines its non-negative (closed) halfspace H_i of points that satisfy the inequality. Thus the solution set of the system of inequalities is the intersection of the halfspaces $\bigcap_i H_i$.

2.1.3 Cones and Polyhedra

A (polyhedral) *cone* is the subset of $\mathbb{R}^{n+1}$ that satisfies a (finite) system of homogeneous linear inequalities in $n+1$ variables, as above. A cone C has the properties

$$\frac{}{\mathbf{0} \in C} \quad \text{(origin)} \qquad \frac{\mathbf{x_1}, \mathbf{x_2} \in C \; a_1, a_2 \in \mathbb{R}}{a_1\mathbf{x_1} + a_2\mathbf{x_2} \in C} \quad \text{(convex closure)}$$

A *polyhedron* is the subset of $\mathbb{R}^n$ that satisfies a (finite) system of linear inequalities in n variables, as above. Note that a cone is a special case of a polyhedron (i.e. when the system is homogeneous). A *polytope* is a bounded polyhedron.

Given a polyhedron $P \subseteq \mathbb{R}^n_{0+}$, there is a unique smallest cone $C_P \subseteq \mathbb{R}^{n+1}_{0+}$ s.t. P is the cross-section of C_P at $x_{n+1} = 1$. Given a cone $C \subseteq \mathbb{R}^{n+1}_{0+}$, there is

a unique polyhedron $P_C \subseteq \mathbb{R}^n_{0+}$ s.t. P_C is the cross-section of C at $x_{n+1} = 1$. This is the geometric view of the translation between homogeneous and non-homogeneous systems of linear inequalities described in Section 2.1.1.

2.1.4 Double Description

The solution space to a system of linear inequalities can be formulated equivalently as the intersection of a finite set of half-spaces and as the convex closure of a finite set of points and rays. The double description method essentially maintains both of these representations, modifying the latter as further inequalities are added to the former.

A pair (A, C) of matrices $A \in \mathbb{R}^{m\times n}$ and $C \in \mathbb{R}^{n\times p}$ is a *double description* iff

$$A.\mathbf{x} \geq \mathbf{0} \quad \text{iff} \quad \mathbf{x} = C.\mathbf{g}, \text{ for some } \mathbf{g} \geq \mathbf{0}$$

0 Lemma (Minkowski's Theorem for Polyhedral Cones)
For any $A \in \mathbb{R}^{m\times n}$, there exists some $C \in \mathbb{R}^{n\times p}$ s.t. (A, C) is a double description.

1 Lemma (Weyl's Theorem for Polyhedral Cones)
For any $C \in \mathbb{R}^{n\times p}$, there exists some $A \in \mathbb{R}^{m\times n}$ s.t. (A, C) is a double description.

2 Lemma (Duality of Double Description)
The matrix pair (A, C) is a double description if and only if $(C^\mathsf{T}, A^\mathsf{T})$ is a double description.

2.1.5 Degeneracy

We say that an inequality $\mathbf{a_i}.\mathbf{x} \geqslant 0$ is degenerate w.r.t. a system of homogeneous linear inequalities $A.\mathbf{x} \geq \mathbf{0}$ ($\mathbf{x} \geq \mathbf{0}$) iff $\mathbf{a_i}.\mathbf{x} = 0$ for all solutions $\mathbf{x}$ to the system of inequalities. In other words, all possible solutions of the

system barely satisfy that inequality. Viewed geometrically, this means all solutions lie on the hyperplane associated with the inequality.

EXAMPLE 0 The inequality $-x_1 - x_2 \geqslant 0$ ($x_1 \geqslant 0,\ x_2 \geqslant 0$) is degenerate (w.r.t. itself).

Obviously if $\mathbf{a_i}$ and $-\mathbf{a_i}$ occur in a system, then both are degenerate. Less obvious is the system

$$\begin{bmatrix} -2 & 1 & 1 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & 1 & -2 & 0 \\ 2 & 1 & 1 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \geq \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{x} \geq \mathbf{0}$$

where the first three inequalities are degenerate (but not the fourth).

2.2 Incremental KB Ordering Algorithm

In this section we present the algorithm of [DKM90][0] for determining whether a given (finite) set of rewrite rules can be ordered by a Knuth-Bendix ordering.

Recall (from p 26) that a Knuth-Bendix ordering is defined by a weight mapping and a precedence on the function symbols. The algorithm proceeds by maintaining a minimal system of these numeric and symbolic constraints as entailed by each term pair considered. If these constraints (A and $\rhd$ below) become unsatisfiable then there is no suitable Knuth-Bendix ordering for the rewrite rules (in the orientation considered). Otherwise the constraints are extended as necessary, ready for further term pairs to be integrated.

[0] with minor corrections

The homogeneous inequalities are represented by the rows of the matrix A in the expression $A.\mathbf{x} \geq \mathbf{0}$. Since degenerate rows of A may entail failure or further constraints, according to the source of the row, each row is labelled to identify its source. For a term pair (p, q), the inequalities generated are for variables:

$$\mathbf{a_v}: \quad w_v \geqslant 0$$

for each constant symbol:

$$\mathbf{a_f}: \quad w_f - w_v \geqslant 0$$

for each non-constant symbol:

$$\mathbf{a_g}: \quad w_g \geqslant 0$$

for each term pair:

$$\mathbf{a_{p,q}}: \quad \sum_{f \in \mathcal{F}} w_f(\#(f,p) - \#(f,q)) + w_v \sum_{v \in \mathcal{V}} (\#(v,p) - \#(v,q)) \geqslant 0$$

EXAMPLE 1 For the term pair $(i(x*y),\ i(y)*i(x))$, the inequalities generated are

$$\begin{array}{r} \mathbf{a_v}: \\ \mathbf{a_i}: \\ \mathbf{a_*}: \\ \mathbf{a_{(i(x*y),\ i(y)*i(x))}}: \end{array} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} w_v \\ w_i \\ w_* \end{bmatrix} \geq \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

◇

Suppose that a set of term pairs R has already been oriented by the IKBO algorithm, producing numeric constraints (i.e. a system of linear inequalities)

A and symbolic constraints (i.e. the precedence on function symbols) $\rhd$. If these constraints are satisfiable then the rewriting system R is terminating by Knuth-Bendix orderings. Now suppose another term pair (t, u) is to be incorporated into R, that is, the augmented rewriting system $R \cup \{t \rightarrow u\}$ is to be tested for KB termination.

First the linear inequalities entailed by (t, u) are generated as above and added to the inequalities A. If this extended system of inequalities, A', can be strictly satisfied, then the system R' is KB terminating. However, if an inequality $\mathbf{a}$ cannot be strictly satisfied by any choice of weights for the KB ordering then $\mathbf{a}$ is degenerate for that system of inequalities and the next step depends on the nature of the degeneracy of $\mathbf{a}$.

If the inequality $\mathbf{a_v}$ is degenerate, this means that a positive weight cannot be assigned to variables, and so R' cannot be proved terminating by KB orderings. If an inequality $\mathbf{a_f}$ is degenerate, where the function symbol f is unary, then f requires to be maximal in the precedence. If the precedence cannot be extended with f maximal, then KB orderings fail again. If an inequality $\mathbf{a_{r,s}}$ is degenerate it means the term pair (r, s) has equal weight on both sides. If the head function symbol (i.e. at the root of the term) of r and s is different, then the precedence requires to be extended in favour of r. If the head symbols are the same, then the immediate subterms of r and s are pairwise compared and the first two distinct subterms, r' and s' say, need to be incorporated by the algorithm so that r' is greater than s'.

Thus for each new term pair (t, u), the algorithm minimally extends the constraints to make t greater than u in all contexts and under all substitutions. The algorithm is summarised in Figure 2.0. The family of KB orderings handled by the algorithm can be extended to allow the precedence to be a pre-order instead of an ordering and to allow function symbols to

have status so that their subterms are compared as multisets or permuted sequences. Since these extensions are straightforward and do not affect the numeric constraints, the focus of this chapter, the interested reader is referred to [Ste88].

Input: Existing system of homogeneous inequalities A and precedence $\rhd$, along with new term pair (t, u).
Output: Either succeed with extended system A' and precedence $\rhd'$, or fail.
/* Initialise set of pairs to be incorporated: */
$P := \{(t, u)\}$;
while $P \neq \varnothing$ **do**
  choose $(p, q) \in P$;
  /* Check feasibility of variable occurrences: */
  if $p \in \mathcal{V}$ or $\#(v, p) < \#(v, q)$ for some $v \in V$ **then** fail;
  else
    $A := A \cup \mathsf{ineqs}(p, q)$; $P := P \setminus \{(p, q)\}$;
    /* Check weight of variables is positive: */
    if $\mathbf{a_v} \in \mathsf{degen}(A)$ **then** fail;
    /* Check zero-weight unary function symbols: */
    for each $\mathbf{a_f} \in \mathsf{degen}(A)$, f unary **do**
      extend $\rhd$ with f maximal, otherwise fail;
    /* Check equal-weight terms: */
    for each $\mathbf{a_{r,s}} \in \mathsf{degen}(A)$ **do**
      if $\mathsf{hd}(r) \neq \mathsf{hd}(s)$ **then**
        extend $\rhd$ with $\mathsf{hd}(r) \rhd \mathsf{hd}(s)$, otherwise fail;
      else
        $P := P \cup \mathsf{reduce}(r, s)$;

Figure 2.0: Incremental Knuth-Bendix Ordering Algorithm

2.3 Method of Complete Description

The method of complete description is a technique for linear programming (optimising a linear function with respect to linear constraints) due to Uzawa ([Uza58b, Uza58a, Kre68]). The following description of the MCD owes its cleanliness to the double description method of [MRTT53], which is similar

but was found independently.

The MCD checks the satisfiability of a system of homogeneous linear inequalities A by maintaining a double-description pair (A, C). When a new inequality is incorporated into A, the set of bounding rays C is updated. The solution set, the cone whose bounding rays are the columns of C, always contains the trivial solution $\mathbf{0}$, which of course is not a valid choice of weights for a Knuth-Bendix ordering. The IKBO algorithm checks satisfiability by testing where any degeneracy occurs, that is, by testing which rows of AC are $\mathbf{0}^\mathsf{T}$. The algorithm is given in Figure 2.1 and an example is worked through below.

Input: Existing double description (A, C), along with new constraint $\mathbf{a}'.\mathbf{x} \geqslant 0$.
Output: Constrained double description (A', C').
/* Compute agreement of existing rays: */
$\mathbf{z} := \mathbf{a}'.C$;
/* Retain rays satisfying new constraint: */
$C' := \{\mathbf{c_i} \in C \mid z_i \geqslant 0\}$
/* Generate new rays to satisfy new constraint: */
$N := \{z_j\mathbf{c_i} - z_i\mathbf{c_j} \mid z_i < 0 < z_j\}$
/* Accept all new rays: */
$C' := C' \cup N$
$A' := A \cup \{\mathbf{a}'\}$

Figure 2.1: Method of Complete Description

2.4 Illustration of Redundancy in MCD

We now work through an example of applying the MCD to a linear programme, both as an illustration of the algorithm and as a witness to the redundant data that can be generated.

Let us test the satisfiability of the set of linear (homogeneous) inequalities

$$\left.\begin{aligned} x_1 + x_2 + -2x_3 &\geqslant 0 \\ 2x_1 + -x_2 + -x_3 &\geqslant 0 \end{aligned}\right\}$$

where $x_1, x_2, x_3 \geqslant 0$.

Each constraint is handled in turn, but since the initial constraints

$$\left.\begin{aligned} x_1 &\geqslant 0 \\ x_2 &\geqslant 0 \\ x_3 &\geqslant 0 \end{aligned}\right\}$$

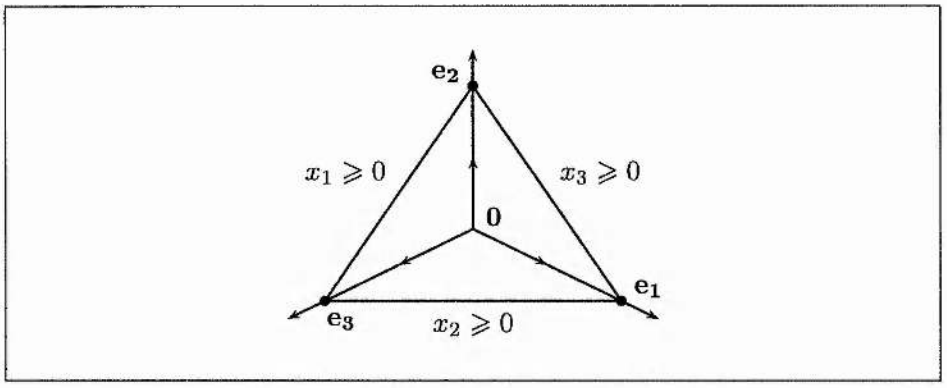
described by the $m \times n$ matrix

$$A_3 = \left[\begin{array}{ccc} 1 & 0 & 0 \\ \hdashline 0 & 1 & 0 \\ \hdashline 0 & 0 & 1 \end{array}\right]$$

are trivial, they are dealt with at once, having the solution (n, p)-matrix

$$C_3 = \left[\begin{array}{c:c:c} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}\right].$$

The current state can be visualised as in the following diagram. We are looking towards the origin fromt the positive octant of 3-space. It may help to imagine looking into an upside-down (tetrahedral) pyramid.

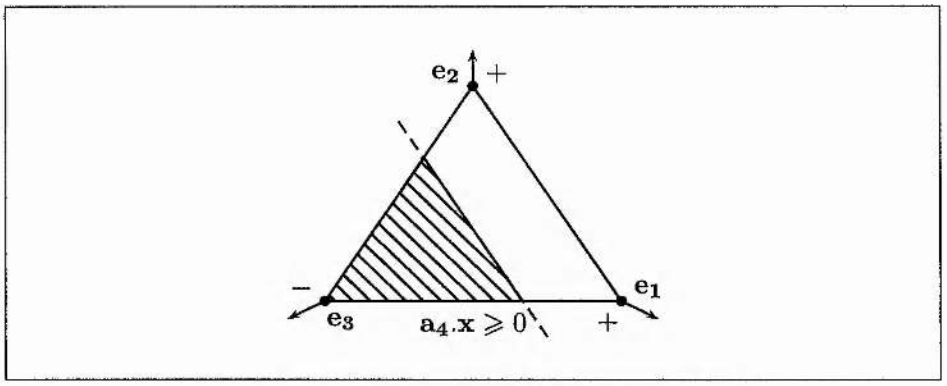


The current solution vectors $\mathbf{c_1}, \mathbf{c_2}, \mathbf{c_3}$ $(\in C_3)$ are shown lying along the three axes $\mathbf{e_1}, \mathbf{e_2}, \mathbf{e_3}$. For the purposes of illustration, a cross-section of this 3-edged cone has been taken on the plane $x_1 + x_2 + x_3 = k \geqslant 0$, for some constant $k \in \mathbb{R}_+$. (For example, $k = 3$, giving a plane containing the point $(1, 1, 1)$.) The boundary of the cross-section is formed by the intersections of the bounding planes, the planes where the constraints are only just satisfied. For the remainder of this example we will concentrate on this cross-section, indicating by points where vectors intersect the plane.

Now the algorithm deals with the next inequality vector, $\mathbf{a_4} = (1, 1, -2)$, testing its agreement with the existing solution vectors by computing their inner products:

i	1	2	3
$\mathbf{a_4}.\mathbf{c_i}$	+1	+1	−2

This means that the inequality $\mathbf{a_4}$ is (strictly) satisfied by vectors $\mathbf{c_1}$ and $\mathbf{c_2}$, and is not satisfied by vector $\mathbf{c_3}$. This is illustrated by



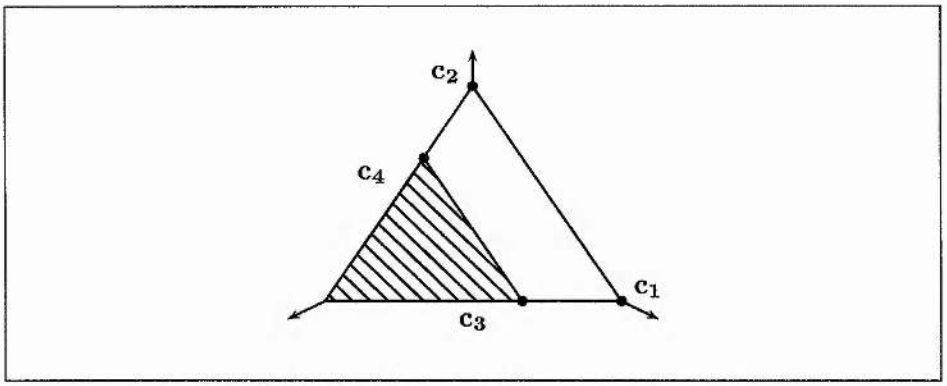
where the satisfying vectors are marked with '+' and the non-satisfying vector is marked with '−'. The solution vectors of the current linear inequalities are found by copying the two positive vectors $\mathbf{c_1}$ and $\mathbf{c_2}$ as well as creating two new solution vectors from the weighted means of $\mathbf{c_1}, \mathbf{c_3}$, and of $\mathbf{c_2}, \mathbf{c_3}$.

$$2\mathbf{c_1} + \mathbf{c_3} = \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix}, \qquad 2\mathbf{c_2} + \mathbf{c_3} = \begin{bmatrix} 0 \\ 2 \\ 1 \end{bmatrix}$$

giving the new solution matrix,

$$C_4 = \left[\begin{array}{c|c|c|c} 1 & 0 & 2 & 0 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 1 \end{array}\right],$$

illustrated by



Now the algorithm incorporates the next inequality vector, $\mathbf{a_5} = (2, -1, -1)$, testing its agreement with the existing solution vectors by computing their inner products:

i	1	2	3	4
$\mathbf{a_5}.\mathbf{c_i}$	+2	−1	+3	−3

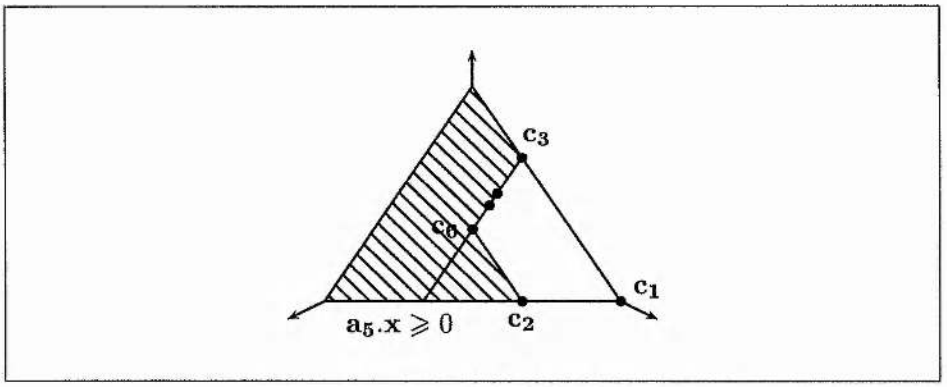
Again, the satisfying vectors ($\mathbf{c_1}$ and $\mathbf{c_3}$) are copied, and new vectors are created using weighted means between vectors on opposing sides of the latest inequality.

$$\mathbf{c_1} + 2\mathbf{c_2} = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}, \; 3\mathbf{c_1} + 2\mathbf{c_4} = \begin{bmatrix} 3 \\ 4 \\ 2 \end{bmatrix}, \; \mathbf{c_3} + 3\mathbf{c_2} = \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}, \; \mathbf{c_3} + \mathbf{c_4} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

giving the current solution matrix,

$$C_5 = \left[\begin{array}{c|c|c|c|c|c} 1 & 2 & 1 & 3 & 2 & 1 \\ 0 & 0 & 2 & 4 & 3 & 1 \\ 0 & 1 & 0 & 2 & 1 & 1 \end{array}\right],$$

illustrated by



In the diagram above, the solution space is the convex closure of $\{\mathbf{c_1}, \mathbf{c_2}, \mathbf{c_3}, \mathbf{c_6}\}$. Therefore, the other two rays $\mathbf{c_4}$ and $\mathbf{c_5}$ are redundant. It is important to note that these superfluous rays will lead to much greater redundancy as the algorithm proceeds: every future ray produced in combination with a redundant ray will itself be redundant. Looking again at the example above, in only three dimensions and after only two inequalities, if the next inequality separates $\mathbf{c_1}$ and $\mathbf{c_2}$ from $\mathbf{c_3}$ and $\mathbf{c_6}$ then 8 new rays (4×2) will be generated for the new boundary plane, whereas we can see that exactly 2 would be necessary.

It is easy to see that for cones in three dimensions, each new boundary plane is fully defined by 2 rays and in its creation must remove at least 1 ray, and so the number of rays required after m inequalities have been introduced is bounded above by $m + n$, which is achieved by the algorithm presented below. Unfortunately such a nice relationship does not hold in higher dimensions, since even in four dimensions the number of rays required may grow exponentially in the number of inequalities considered.

2.5 Removing Redundancy from the MCD

The key to identifying redundant rays is to note that a non-redundant ray is defined by the hyperplanes on which it lies, and so must lie on at least

$(n-1)$ hyperplanes. Since redundant rays can only occur upon the creation of a new bounding hyperplane, a new non-redundant ray must lie on at least $(n-2)$ previous hyperplanes.

3 CLAIM (REDUNDANT RAYS)

Let (A, C) be a double description. A ray $\mathbf{v} \in C$ is redundant if it barely satisfies fewer than $(n-1)$ inequalities of A.

JUSTIFICATION[1] Let $\mathbf{v} \in C$ be a ray that barely satisfies k $(1 \leqslant k < n-1)$ inequalities, that is, $\mathbf{v}$ lies on k bounding hyperplanes, and let $\mathbf{a}$ be the inequality that generated $\mathbf{v}$ (i.e. that caused $\mathbf{v}$ to be introduced into C). Then it is claimed that there is a a vector $\mathbf{w}$ such that $\mathbf{v} + \delta\mathbf{w}$ and $\mathbf{v} - \delta\mathbf{w}$ $(\delta \in \mathbb{R})$ also lie on the hyperplane $\mathbf{a}$ for small enough $\delta > 0$ and satisfy all the inequalities. (The vector space of all such $\mathbf{w}$ has dimension $(n-k)$.)

4 COROLLARY (REDUNDANT INEQUALITIES)

Let (A, C) be a double description. An inequality $\mathbf{a_i}.\mathbf{x} \geqslant 0$ $(\mathbf{x} \in \mathbb{R}^n_+)$ is redundant if it is barely satisfied by fewer than $(n-1)$ rays of C.

JUSTIFICATION Follows from Claim 3 and the fact that (A, C) is dual to (C', A').

The above claims are known to hold for cones in dimensions up to and including 4, but since higher dimensional geometry can be counter-intuitive, potential implementors have to rely on anecdotal evidence until proofs are found.[2] No counter-example has been found among the many examples tried, but of course that does not rule out the existence of overlooked special cases.

Note that if the claims prove to be false, the revised MCD may be incomplete but it will still be sound, since in that case it may mistakenly exclude valid solutions.

[1] A rigorous proof remains elusive.

[2] It is hoped that someone with a deeper understanding of higher dimensional geometry will be able to prove these claims.

2.6 Revised MCD

Figure 2.2 presents the revised method of complete description, the correctness of which is subject to reservations expressed in Section 2.5. The algorithm follows the original, but after new rays are generated for a new bounding hyperplane, the inequalities and rays are checked for redundancy as described above.

Input: Existing double description (A, C), along with new constraint $\mathbf{a}'.\mathbf{x} \geqslant 0$.
Output: Constrained double description (A', C').
/* Compute agreement of existing rays: */
$\mathbf{z} := \mathbf{a}'.C$;
/* Retain rays satisfying new constraint: */
$C' := \{\mathbf{c_i} \in C \mid z_i \geqslant 0\}$
/* Generate new rays to satisfy new constraint: */
$N := \{z_j\mathbf{c_i} - z_i\mathbf{c_j} \mid z_i < 0 < z_j\}$
/* Retain non-redundant inequalities: */
$A' := \{\mathbf{a} \in A \mid \#(\{\mathbf{c} \in C \mid \mathbf{a}.\mathbf{c} = 0\}) \geqslant n-1\} \cup \{\mathbf{a}'\}$
/* Retain non-redundant new rays: */
$C' := C' \cup \{\mathbf{v} \in N \mid \#(\{\mathbf{a} \in A' \mid \mathbf{a}.\mathbf{u} = 0\}) \geqslant n-1\}$

Figure 2.2: Revised Method of Complete Description

A simple way to implement the redundancy checks is to examine $A \times C$ for rows and columns having fewer than $n-1$ entries of 0.

2.7 Conclusions

There are more efficient algorithms for linear programming. For example, the simplex method (see for example [Dan63, Kre68]) usually has a time-complexity linear in the number of inequalities, and Karmarkar's method ([Kar84]) has polynomial worst-case time-complexity (although the constants are very large). However, it is not clear how any of these methods can be

used as an efficient test for the location of degeneracy. The revised MCD takes $O(np^2)$ steps to incorporate each inequality, but since the output (the extreme rays) has size $O(np^2)$ this might be considered as optimal.[3]

In this chapter we have considered the Knuth-Bendix orderings of [Mar87]. Clearly the revised MCD is equally useful for extended formulations of the Knuth-Bendix orderings, including formulations for terms with status ([Ste94]), modulo equational theories (e.g. associative-commutative, [Ste94]), and being order-sorted ([Mat93]).

The inequalities generated by the ordering algorithm could be made strict, since strict satisfaction is sufficient for the existence of a Knuth-Bendix ordering. Then there would be no issue regarding degeneracy, and so a more space- and time-efficient method such as the Simplex Method could be employed for testing satisfiability. However, this is testing for the existence of a much weaker ordering: a weight ordering.

Another approach would be to retain the power of the present orderings, but to sacrifice completeness. A hillclimbing or stochastic-sampling technique could be used, and if an inequality is not strictly satisfied after a predetermined number of attempts then degeneracy is judged to be likely . Any technique that does not consider all points in the solution space may fail to find a non-degenerate solution (in a practical length of time) and may therefore misdirect the ordering algorithm to conclude non-satisfiability (i.e. termination not provable by Knuth-Bendix orderings). This could be an attractive compromise when the size of the termination problem grows beyond the limits of even the revised MCD.

The revised MCD is of value independent of the ordering algorithm. For instance [Ste94] employs the MCD in searching for termination by polyno-

[3]Strictly speaking the extreme rays do not have to be part of the output, so there may exist more efficient algorithms to perform the same test.

mial orderings. complex, potentially less efficient, and less reliable techniques such as the Simplex Method. The revised MCD may also prove useful in other domains, such as 3-D modelling where the detection of interior points is a goal rather than a curse.

In Section 3.3.1 we will consider another application of the revised MCD; in combination with genetic algorithms (described in Chapter 3) we have an attractive technique for optimising a function subject to linear constraints such that no assumption of differentiability or even continuity is made of the objective function.

3 Genetic Termination

In this chapter genetic algorithms are proposed as an efficient and adaptive means of proving term rewriting systems terminating by polynomial ordering.

3.0 Introduction

In the previous chapter we saw an effective (full) decision procedure for determining whether a given term rewriting system terminates via Knuth-Bendix orderings. However, a rewrite relation may be terminating but not Knuth-Bendix terminating, and so it is desirable to extend the class of term rewriting systems that we can prove terminating. Polynomial orderings (Definition 14) are more 'sophisticated' than Knuth-Bendix orderings in the sense that the relative ordering of two terms is determined by the position of symbols in the terms and not just the number of their occurrences. For example, a distributivity rule such as

$$r \wedge (s \vee t) \rightarrow (r \wedge s) \vee (r \wedge t)$$

cannot be contained in a Knuth-Bendix ordering because the duplication of x on the right-hand side means the weight constraint can be violated since an arbitrarily large term can be substituted for r. A polynomial ordering is defined by interpreting function symbols as polynomial functions on $\mathbb{N}$, so one possible polynomial interpretation for the above rule is

$$[\![\wedge]\!](x, y) = xy + y, \qquad [\![\vee]\!](x, y) = x + y + 1,$$

which gives

$$[\![r \wedge (s \vee t)]\!] = ([\![r]\!] + 1)([\![s]\!] + [\![t]\!]) + [\![r]\!] + 1$$
$$[\![(r \wedge s) \vee (r \wedge t)]\!] = ([\![r]\!] + 1)([\![s]\!] + [\![t]\!]) + 1$$

Since this polynomial ordering satisfies the well-foundedness requirements detailed below, the rule is contained in a termination ordering as long as all ground terms are positive (or at least those terms that may form the first argument of multiplication).

Unfortunately the price paid for this power is that there is no known efficient algorithm for deciding whether a suitable polynomial ordering exists. Stronger than this, even if a particular polynomial ordering is chosen, it is undecidable in general (for polynomials over the natural numbers) whether or not the ordering proves the given rewrite system terminating ([Lan79]). Polynomial constraints are decidable if the polynomials range over the real numbers, due to the decomposition algorithm of Collins ([Col75]), but the algorithm is exponential in its input and is impractical even for modest examples.

Nevertheless, polynomial orderings are relatively popular for tackling termination problems in term rewriting, and some work has been successful in making their use more amenable, notably the methods of BenCherifa & Lescanne [BL87b], Steinbach [Ste91, Ste92], and Giesl [Gie95a], described later in this chapter. These methods are semi-automatic, in that given a polynomial interpretation and a rule set they try to show that the rewrite relation is terminating under that ordering. If a method fails to show an ordering is suitable, another polynomial interpretation must be tried. Thus each method sequentially traverses the search space of possible polynomial orderings in the hope of finding an ordering that it can demonstrate to be

suitable.

The main idea of this chapter is to employ a genetic algorithm coupled with the gradients method of testing polynomials in the search for a suitable polynomial ordering. A population of polynomial interpretations is 'bred' through successive generations, with bias towards those orderings that are in some measure (based on the result of the gradients test) closer to proving termination of the given rewrite relation, until either a suitable ordering is produced, population stagnation has been detected, or a predetermined number of generations have elapsed without success.

John Holland proposed in [Hol75] that the schema theorem is the reason why GAs can often traverse the search space more cheaply than traditional approaches. In this theorem each individual in the population, for example,

```
110100101
```

is seen as sampling the search space on each of the individual's constituent segments. (In the theorem, segments are 1-bit long.) It is shown by means of segment templates (schemas), for example

```
--01---0-,
```

that segments beneficial to the solution of the problem have a higher probability of being represented in the next generation. Unfortunately the theorem assumes a vast, uniformly distributed population, so may not even be valid at the start of processing, and certainly not afterwards. An individual in a generation is evaluated only once, yet that value is pertinent to the evaluation of all the segment templates matching that individual. Therefore the segment templates corresponding to solutions of the problem will become, through the evaluation of individuals matching those templates, more likely to prevail in the population. In the case of binary encoded GAs, the sampling of segment templates can be seen as partitioning the hypercube of possible

individuals (Figure 3.0), and sampling all such partitions in parallel.

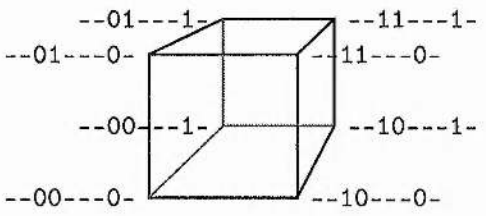


Figure 3.0: Partitioning the hypercube.

3.1 Termination by Polynomial Ordering

The generate-and-test approach to finding a suitable polynomial ordering consists of picking a polynomial interpretation for $\mathcal{F}$ and then testing whether that ordering contains the rewrite relation in question. If the polynomial interpretations of function symbols are allowed to be arbitrarily complex, the polynomials resulting for the terms may be horrendously complex (in terms of time to compute and number of monomials produced), making any test for polynomial dominance unworkable. Therefore practical approaches involve applying heuristics restricting the form the interpretations may take. For example, we could insist that only linear interpretations be considered, such as

$$[\![+]\!](x, y) = c_0 + c_1 x + c_2 y + c_3 xy$$

Restricting to linear interpretations is too limiting, however. Often if a rewrite relation is terminating under a linear polynomial ordering, it can be shown terminating under a Knuth-Bendix ordering (which is quicker and easier). This is particularly true for interpretations of the form

$$[\![f]\!](\vec{x}) = c_0 + \sum \vec{x}$$

since these are simply weight interpretations.

Steinbach found (in [Ste94]) that the overwhelming majority of rewriting systems he looked at could be proved terminating by 'simple-mixed' polynomial interpretations.

An approach to proving termination via polynomial orderings is to generate an interpretation of $\mathcal{F}$ and then test whether $[\![l_i]\!](\vec{x}) > [\![r_i]\!](\vec{x})$ for all $\vec{x}$ and all rules. This is equivalent to testing whether the polynomial $[\![l_i]\!](\vec{x}) - [\![r_i]\!](\vec{x}) > 0$ for all $\vec{x}$ and all rules. The algorithm of Collins ([Tar51, Col75]) can decide the positiveness of a polynomial over the positive reals, but it is not used in practice due to its prohibitive time complexity.

The gradients method of Lankford ([Lan75]) is more practical in terms of speed, testing whether the first-order partial derivatives of each polynomial are eventually positive. However, 'eventually' positive is not sufficient here because it is not well-founded in the presence of ground terms. For instance, it erroneously orients the rewriting system

$$\left.\begin{array}{l} \mathsf{f}(x) \rightarrow \mathsf{g}(x) \\ \mathsf{g}(\mathsf{c}) \rightarrow \mathsf{f}(\mathsf{c}) \end{array}\right\}$$

via the interpretation $[\![\mathsf{f}]\!](x) = 2x$, $[\![\mathsf{g}]\!](x) = x + 2$, and $[\![\mathsf{c}]\!] = 1$. This is because it doesn't take account of μ, the lower-bound for the interpretations of constants, which can be seen in the graph of Figure 3.1.

The three methods described below provide semi-decision procedures for polynomial termination.

3.1.0 Method of BenCherifa & Lescanne

The method of Ben Cherifa and Lescanne, in [BL87b], attempts to reduce a complex polynomial by successive approximations until its positiveness is

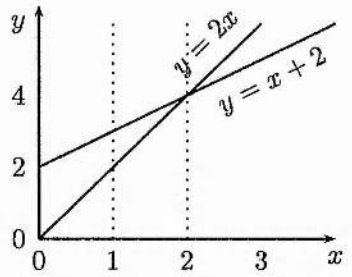


Figure 3.1: Well-foundedness of polynomial orderings.

obvious.

$$p_0(\vec{x}) > p_1(\vec{x}) > \cdots > p_{i-1}(\vec{x}) > p_i(\vec{x}) > 0$$

If a (non-vacuous) polynomial contains no negative monomials, then it must be positive. Otherwise, the algorithm employs a heuristic to select a positive monomial to diminish the negative monomial. This continues until either there are no remaining negative monomials (success) or no positive monomials are capable of diminishing a negative monomial (failure). The method is implemented by way of a rewriting system in the ORME theorem prover.

Although Steinbach found this method to be successful in the majority of feasible examples (those collected in [SK93]), the heuristics involved can detrimentally affect the outcome if applied in an unfortunate order, and feasible examples can be made infeasible by the approximation step.

3.1.1 Method of Steinbach

In [Ste94] Steinbach extends the method of BenCherifa and Lescanne by introducing backtracking search and letting μ rise above the (heuristically set) value of 2. The method applies a variety of heuristics for approximating the polynomial inequalities to a system of linear inequalities, whereupon the

system is tested for satisfiability (using the method of complete description – Section 2.3). The method is implemented in the TETRES termination tool, which is integrated with the COMTES theorem prover.

3.1.2 Method of Giesl

In [Gie95a] Giesl incorporates the value of μ into the gradients method of Lankford, and shows that this corrected version is equivalent in effectiveness to the methods of BenCherifa & Lescanne and of Steinbach. We will examine the gradients method below, since it is used for the objective function of the genetic algorithms in this chapter. Giesl's method is implemented in the POLO ([Gie95b]) termination tool.

3.1.3 Gradients Method

Given an interpretation of $\mathcal{F}$ into polynomials over $\mathbb{N}$, the task of proving termination is to show that $[\![t]\!] - [\![u]\!] > 0$ for all rules $t \rightarrow u$ in the given rewriting system. To prove $f(\vec{x}) > 0$, where $x_i > \mu \geqslant 0$, for a polynomial function f, the gradients method ([Lan79, Gie95a]) checks that

$$f(\vec{x})\{x_i \mapsto \mu\} > 0$$

and that the gradient in the direction of x_i is non-negative. It does so by applying the following two rules upwards to replace the polynomial inequality, containing term variables, with a system of simpler inequalities containing no term variables:

$$\frac{f(\vec{x})\{x_i \mapsto \mu\} > 0, \quad \frac{\delta}{\delta x_i} f(\vec{x}) \geqslant 0}{f(\vec{x}) > 0}$$

$$\frac{f(\vec{x})\{x_i \mapsto \mu\} \geqslant 0, \quad \frac{\delta}{\delta x_i} f(\vec{x}) \geqslant 0}{f(\vec{x}) \geqslant 0}$$

where the domain is $\{x \in \mathbb{N} \mid x \geqslant \mu\}$. Repeatedly applying the above rules results in a system of linear inequalities in coefficients of the polynomial interpretations (and the variable μ). This system of inequalities is used below as a measure of fitness of individuals in a population of candidate coefficients.

In addition to the usual constraint that $[\![t]\!] > [\![u]\!]$ for all rewrite rules $t \rightarrow u$, the gradients method of Giesl demands

- $\frac{\delta}{\delta x_i}([\![f]\!](\vec{x})) > 0$, for all $f \in \mathcal{F}$, for all non-constant function symbols,
- $[\![f]\!](\mu, \ldots, \mu) \geqslant \mu$, for all function symbols,
- $[\![c]\!] \geqslant 0$, for all constants.

The first two demands are automatically met by ensuring that all variables are represented in the polynomial. For example, suppose

$$[\![f]\!](x, y) = c_0 + c_1 x + c_2 y + c_3 xy$$

Then the representation of variables condition requires

- $c_1 > 0$ or $c_3 > 0$, and
- $c_2 > 0$ or $c_3 > 0$.

The third demand is met by setting μ to the minimum interpretation of constants.

3.2 Genetic Algorithms

There are many variants of genetic algorithms, but for the purposes of this work we adopt a scheme close to the original formulation by Holland in [Hol75]. The components that make up a genetic algorithm will be described below with reference to how they were used for polynomial termination and what choices were found to be optimal on the examples tried. For a general survey of genetic algorithms the reader is directed to [Gol89a, Whi93, BBM93a, BBM93b].

Input: (finite) set of oriented pairs defining TRS to be proved terminating
Output: either succeed with polynomial interpretation, or fail
choose form of polynomial interpretations
derive system of inequalities in interpretation coefficients
generate initial random population, P
evaluate each individual against inequalities
while termination not proved and search limit not reached **do**
for $j = 1, \ldots, n$ **do**
randomly grab g (tournament size) individuals from P
select best individual w.r.t. objective function from $1 \ldots g$
randomly apply recombination
randomly apply mutation
evaluate each individual against inequalities
output best individual

Figure 3.2: Genetic Algorithm for Termination

Trials were carried out using the GALOPPS ([Goo96]) code library applied to problems from Steinbach's collection [SK93].

3.2.0 Encoding

The standard encoding of sample points in a genetic algorithm is as fixed length binary strings.

The individuals of our GA are instantiations for the coefficients of the

polynomial interpretation of $\mathcal{F}$.

Suppose that $\mathcal{F} = \{a, f(\_), g(\_,\_)\}$ and the interpretations chosen are of the form

$$\begin{aligned} a &= c_0 \\ f(x) &= c_1 + c_2 x + c_3 x^2 \\ g(x,y) &= c_4 + c_5 x + c_6 y + c_7 xy \ . \end{aligned}$$

Then an individual in the population will be of the form

$$c_0 \mid c_1 \mid c_2 \mid c_3 \mid c_4 \mid c_5 \mid c_6 \mid c_7.$$

For simplicity the fields were made of uniform length, and during trials a field length of 3 bits was used, so that coefficients ranged over the values $0, \ldots, 7$. This choice was fairly arbitrary; a smaller range would have made the problem easier for the genetic search but limited the rewriting systems provably terminating.

The value of μ (the minimum value of a ground polynomial) was originally treated in the same manner as the coefficients by attaching it to each individual as an extra field. This allowed the value of μ to be sought along with the other parameters, but was found to be detrimental to the performance (cost in time) of the technique, so instead a heuristic was adopted of setting μ to the value of the lowest constant in the individual when that individual came to be evaluated.

The initial population was created randomly. The examples tried involved 20–30 fields, for which a population of 50–100 individuals was found to minimise the number of evaluations needed. (This value is considerably lower than the values recommended in, for example, [Gol89b]. That is because they are aiming to minimise number of generations required, whereas

here we are wanting to minimise time taken.)

3.2.1 Fitness

The objective function to be optimised by the genetic algorithm is often referred to as the fitness function. No constraints such continuity or differentiability are placed on the fitness function, and experiences related in the literature suggest that genetic algorithms are relatively robust to noise and multi-modality in the fitness function.

The fitness function was initially defined as the number of rewrite rules correctly oriented by the individual. When the difference in number of rules oriented between the best and worst individuals of the population was small, this produced a very 'stepped' fitness function, which was detrimental to guiding the population towards a successful individual. Instead, the fitness was measured as the number of inequalities (from the gradients method) that the individual satisfied, resulting in a more discriminating and perhaps more accurate measure of an individual's fitness. In addition, the selection scheme described below was found to play a significant part.

3.2.2 Selection

Selection is the means by which the better individuals have a higher probability of contributing to the next generation. The classic GA of Holland and in fact most GAs in the literature use fitness-proportionate selection, such as roulette wheel selection. In such schemes the probability of an individual being selected is a function (usually a linear scaling) of its fitness value.

Initially the most surprising aspect of the GA implemented was that the popular selection schemes such as stochastic universal sampling, roulette wheel, stochastic remainder, and linear ranking selection (see [BT95] for

details) all gave very poor results. The GA as described did little better than (the mean of a batch of) random search. When tournament selection was used, however, the GA performed very well, giving the results described below.

In tournament selection g individuals are examined and the best one goes through to the next generation, so that every individual in the next generation has had to better $(g-1)$ randomly selected 'opponents'. It shouldn't have been such a surprise when the fitness function is considered. A large proportion of inequalities are satisfied in the initial population, and as the population average improves, the difference between worst and best diminishes and the granularity of the fitness function gives little direction to further improvement. On the other hand, tournament selection places the same relative selection pressure on individuals regardless of the stage of the run, and so the drive towards satisfying the inequalities is maintained much stronger.

3.2.3 Recombination

Recombination is the means by which different combinations of building blocks can be tested. Also known as sexual reproduction, this operation takes pairs of individuals and swaps a randomly chosen segment in them. Recombination is usually viewed as a 'concentrating' operation, focusing the sampling on promising combinations of building blocks.

Two point recombination, where the bit string is viewed as a circle, was found to be best, as is usually the case when being situated at the end of the individual has no special significance to the field. Since the encoding was field-based, and it is meaningless to swap part of the value of one coefficient with part of the value of another, recombination points were restricted to field boundaries. On the examples tested, the optimal value of recombination

appeared to be quite low (around 30%) compared to more common values in the literature (40–60%) but the difference in performance was inconclusive from the small sample size.

3.2.4 Mutation

Mutation is the means by which diversity is maintained and new data points may be introduced. Each individual has a small probability of having part of it changed to a new value. (With classical binary encodings, this means a bit-flip.)

Since the encoding was field-based, mutation meant changing the value of a coefficient in an individual to a new value. The probability of mutation was found to be one of the most important parameters for ensuring speedy convergence to a solution, with a per-field probability of around 10%, which is rather high in comparison to most GA applications (usually less than 1%). This and the recombination probability suggest that the search for polynomial orderings relies more on diverse coverage of the search space than specialising in recombination of coefficients, at least for the handful of examples so far tried. However, the GA was not simply hillclimbing, since it performed very poorly if recombination was turned to 0%.

3.2.5 Trials

In [SK93] 93 examples are given of rule sets that were shown to be terminating using their Tetres termination tool. Of these rule sets, 81 are terminating by polynomial orderings (Steinbach's POL ordering). When looking for examples on which to try genetic search, those that could be better tackled using other techniques presented in this document (RMCD, Chapter 2; Invariant analysis, Chapter 4) were discarded, so of the 81, 45 were discarded

for being Knuth-Bendix terminating and a further 2 were discarded for being monadic. Of the remaining 34, 3 were discarded for having only one rule, and a further 20 were discarded for being terminating by 'first guess' interpretations (e.g. by interpreting binary function symbols as $x + y + 1$ and constants as 2). Thus 11 examples (2.15, 2.18, 2.28, 2.31, 2.38, 2.42, 2.43, 2.44, 2.57, 3.6, 3.7) remained for the trials.

The values for the genetic operators described above were found by taking an example rule set, varying the value of each genetic operator separately until a locally optimal value was found for that operator and that rule set. Then subsequent rule sets were tried and the operator values were adjusted to see whether they could be improved. Of the 11 examples, the difference in performance (ratio of number of evaluations required to find a solution) between the general operator values and the optimal values for that example was usually within a factor of 2 (i.e. fine-tuning the operator values on a per-problem basis typically produced less than doubling of success speed) which suggests the technique is fairly stable for this application area, at least on the limited set of examples tried.

To illustrate the fine-tuning process, the results for solving Example 2.31 are shown below. The operator values were maintained at

Population size	Crossover proportion	Mutation probability
70	30%	10%

with a tournament size of 3. Each of the genetic operator values was varied, and the number of evaluations taken is tabulated below (averaged over 5 runs and rounded to the nearest multiple of 10).

Population size

Size	40	50	60	70	80	90
Evaluations	580	880	480	500	950	1010

Crossover proportion

Percentage	15	20	25	30	35	40	45	50
Evaluations	1220	600	760	500	520	740	530	760

Mutation probability

Percentage	4	6	8	10	12	14	16
Evaluations	810	490	1010	500	820	1990	2030

Once reliable parameters for the GA had been found, the GA produced a solution so quickly (typically in under 1000 evaluations, taking under 2 seconds) that statistical results are of little value. Moreover, since the implementations are so different, direct comparisons can't be drawn from execution timings on a small set of similar examples. However, to give an idea of the orders of magnitude involved, for Example 2.31 and Example 2.43 in [SK93], TETRES took 15–30 seconds, POLO took 15–40 minutes, and the GA consistently took less than 8 seconds.

It will be interesting to try the GA on a larger, more varied suite of examples since the method of Steinbach is exponential in the number of monomials, the method of Giesl is exponential in the number of variables, and the time complexity of the GA is an unknown. If the GA literature can be believed, the GA may have a bigger advantage on problems with more coefficients and more inequalities.

3.3 Further Developments

There is clearly a great deal of modularity to the genetic algorithm approach to termination ordering; the search engine itself uses no information about the ordering family being tried (e.g. polynomial orderings) other than in the objective function for scoring members of the population. Had the ordering family been different (say, syntactic orderings such as the recursive path orderings) the only necessary changes to the procedure would be in the module for evaluating how close an individual came to proving termination. It is easy, therefore, to envisage a generic termination tool that has many objective functions, one for each of an array of ordering families, and that carries out the same search (modulo the user's choice of objective function) regardless of which ordering family is being tried. Better yet might be a tool that maintains separate populations, with distinct populations encoding members of distinct ordering families, and having the populations compete with each other, thus freeing the user from even having to choose which family to try.

3.3.0 Learning Termination Tool

This chapter has concentrated on one facet of GA research, namely optimisation. However, there is another branch of current GA research that could prove useful both to termination research and to practical termination tools: automated learning.

The use and development of evolutionary computation for automated learning is a growing area of artificial intelligence, and a general account is outside the scope of this document. However, the basic idea is straightforward: emphasise the exploratory nature of the genetic algorithm and weaken the pressures on the population to converge, thereby encouraging the exploration of local optima. This can be achieved by reducing the dominance given

to fitter individuals during mate selection (the selection pressure), and by rejecting duplicate individuals. The intention is for the GA to converge more slowly, considering a wider variety of points in the search space and inhibiting the ability of fitter individuals to dominate the population.

This approach is not directly useful to an interactive or real-time termination tool, since on a moderately sized termination problem the GA will typically be left to run for some hours, but there are several uses of such an 'offline' approach to termination theory, and polynomial termination in particular. The most obvious use is to give the GA a better chance of finding a successful termination ordering. Often, particularly for academic problems, no appropriate polynomial ordering is apparent, initial attempts with termination tools are unsuccessful, yet it is not clear that polynomial orderings are infeasible. Where computational processing is cheaper than intellectual analysis, a slowly converging GA may be used to give a more thorough test of the search space (although a GA search can never be complete). A GA usually deals with a population of fixed-length strings, which, when searching for a suitable polynomial ordering, places an upper bound on the values of polynomial coefficients, resulting in a finite search space. This restriction can be weakened by modifying the algorithm to dynamically lengthen all strings in the population if a significant proportion of the fittest individuals are utilising the full length of their strings, in a way similar to reverse annealing. The user could periodically monitor the progress of the GA (e.g. by looking at a list of the fittest 10 individuals every 1000 generations) to see whether the GA had yet converged and to judge whether to call a halt to its execution.

In addition to giving the GA a better chance of finding a single successful individual (i.e. an appropriate termination ordering), a similar approach

could be taken for finding a set of successful individuals (i.e. all those orderings found at a certain generation that satisfy the given termination problem). In the relevant research literature one finds many examples of orderings that prove certain term rewriting systems terminating. Whether the orderings are found manually or mechanically, the search finishes as soon as one is found.[0] However, as for the Knuth-Bendix orderings of Chapter 4, if there is one suitable ordering then in general there is a continuous set of suitable orderings, as will be shown in Chapter 4. Whereas finding one suitable ordering is sufficient for solving one particular termination problem, getting a picture of the solution sets for given termination problems may help to guide future search techniques and perhaps guide developments of the theoretical analysis of Chapter 4.

The schemata in GAs can be thought of as micro building blocks, but we can also consider macro building blocks: those individuals that are useful to have in the population for leading to successful outcomes. By running successive GAs with the same objective function but with a proportion of each initial population having been manually selected, it may be possible to determine which interpretations it is generally fruitful to ensure exist in the initial population. However, care would have to be taken that these potent individuals were not so much superior to their randomly selected peers that they dominated the population after a couple of generations and so hampered or prevented the search for a suitable individual. Also, so many GAs would have to be executed to derive results of any statistical worth that some means of automating this meta-search would be necessary.

The examples tested in [Ste94] tend to fall into distinct classes, such as group theory, arithmetic, and lists. It would be feasible to 'train' popula-

[0]The Tetres tool of Steinbach looks for all suitable orderings in its bounded search space.

tions of genetic algorithms on homogeneous examples and store the resulting populations, so that when presented with an example of that class there is already a pool of previously successful building blocks to begin.

During the execution of a GA for termination orderings, many orderings are evaluated against the given term rewriting system. In particular, when two individuals are mated (i.e. crossover is applied), the fitness values of the parents and the children are known, and so the information is available as to what changes made a local gain or loss. It may be possible that a sampling optimiser such as GAs may be able to learn from such information, so that genetic operators can be given bias in directions known to be statistically fruitful in the domain of termination orderings. Current genetic algorithms simply sample the fitnesses of the population, but do not learn from the short term relative gains or losses endowed by the genetic operators. Perhaps genetic algorithms can be adapted to learn from such information, although it is not clear to the author how this might be achieved.

3.3.1 Numerical Optimisation

As mentioned in Chapter 2, if we combine genetic algorithms with the Revised Method of Complete Description, we have a powerful tool for an important class of numerical optimisation problems: integer programming under linear constraints and an arbitrary objective function.

Suppose we have a finite set of linear constraints, $A\mathbf{x} \geq \mathbf{0}$, and an objective function, $\mathbf{f}.\mathbf{x}$, to be minimised, and suppose we require the elements of any solution vector $\mathbf{x}$ to be integers. Then we can apply the RMCD to the linear constraints in order to enumerate the extreme points of the solution space, $E = [Q|P]$. Any particular coefficient vector, $\mathbf{c}$, selects a point in the solution space, $\mathbf{s} = E\mathbf{c}$. Therefore we run a GA on a population of coeffi-

cient vectors, $\mathbf{c}$, and use $\mathbf{f}E\mathbf{c}$ as the objective function, thereby guaranteeing that all individuals are feasible solutions and providing an efficient means of finding the optimum vector.

The advantage of this approach over traditional approaches is the lack of restrictions on the objective function; the only restriction is that the objective function can be evaluated for any given feasible solution. Of course, the approach described above will really come into its own when the objective function is multimodal or discontinuous (thereby ruling out gradient methods) and the search space is too large to consider random search.

The disadvantage of this approach is that it employs both the RMCD and a GA, both of which can be computationally expensive, even on inputs where such heavy handed techniques might not be necessary. For a particular input, the only indication that a cheaper hammer might be available to crack the nut would be if the output of the RMCD was relatively small or if the GA converged relatively quickly. However, quick convergence of the GA presents another problem: since the GA is based on heuristics, there is no guarantee that the parameters of the genetic operators were chosen wisely and that a global optimum has in fact been reached. One approach could be to use a GA to identify troughs and then use local hillclimbing to identify global optima.

3.4 Conclusions

In order to simplify the process of composing interpretations to construct polynomial inequalities, the form of polynomials has been restricted in favour of linear polynomials. While it is believed that this is sufficient in the vast majority of cases (with some theoretical justification in Chapter 4), until more is known about the relative potency of monomials in general poly-

nomials[1], it may be worth investigating the lifting of these restrictions on form. A more general approach than the one taken in this chapter is afforded by genetic programming, in which individuals are unbounded trees rather than fixed-length strings. The field of genetic programming is still quite young relative to genetic algorithms, and there are still many problems to be worked out, such as how to perform crossover while maintaining well-formed individuals and if the form of individuals is made too general the search space will be much larger (by orders of magnitude) than it need be. We don't require this full generality for polynomial orderings, however, since the interpretations of the (finite) set of function symbols can be coded as a fixed-length list of polynomials with each polynomial coded as an unbounded list of coefficients (i.e. a summation of monomials). In such an implementation the methods described above for testing positiveness of polynomials would be too costly. Steve Linton proposed (private communication) that the polynomial functions be sampled along their domain, thus giving a fuzzy evaluation of each polynomial inequality. The idea of dealing with arbitrary monomials within each polynomial was rejected for the procedure described in this chapter because that would greatly complicate the implementation of the objective function, and necessarily slow down the progress of the GA, whereas the results of Chapter 4 suggest that consideration of only a few monomials of each polynomial is sufficient.

[1]This will be discussed in Chapter 4.

4 Analysis of Polynomial Orderings

In this chapter we focus on the class of polynomial orderings on monadic terms.

4.0 Introduction

As described in Chapter 0, the complexity of having polynomial interpretations ranging over arbitrarily long polynomials excludes them from practical use. We have seen in Chapter 3 that even when the form of interpretations is tightly restricted, the search space can still be slow and cumbersome. We also know that termination is undecidable for monadic terms, and for polynomial orderings over the naturals.

By restricting terms to contain only unary function symbols we will be able to determine the precise ordering defined by a given polynomial interpretation, and to specify that ordering as a lexicographic combination of simpler orderings.

4.1 Polynomial Orderings on Monadic Terms

Polynomial interpretations are defined as before, by interpreting each function symbol as a polynomial function, but now all function symbols in $\mathcal{F}$ are unary.

DEFINITION 16 (POLYNOMIAL INTERPRETATIONS) Let the set $\mathcal{F}$ be a finite nonempty set of unary function symbols. The polynomial interpretation of a function symbol $f_i \in \mathcal{F}$, denoted $[\![f_i]\!]$, is a finite polynomial function in x ranging over $\mathbb{R}_+$, $[\![f_i]\!](x) = a_m x^m + \cdots + a_1 x + a_0$, where $m \in \mathbb{N}_+$, $a_m \in \mathbb{R}_+$, $a_{m-1}, \ldots, a_0 \in \mathbb{R}$, and either

$$m > 1, \text{ or}$$

$$m = 1 \text{ and } a_m > 1, \text{ or}$$

$$m = 1,\ a_m = 1, \text{ and } a_0 > 0.$$

resulting in non-linear, weakly linear, and strongly linear interpretations, respectively. Each term variable v is interpreted as a polynomial variable v_p. A monadic term $t \in \mathcal{T}$ has polynomial interpretation $[\![t]\!]$ given by the obvious homomorphism. ◇

For notational convenience, we interpret the (only) variable ν as x.

Thus the polynomial interpretation of a monadic term is a polynomial function in one variable, x. Polynomials are ordered by the 'eventually dominates' ordering on functions, which for unary polynomials is total: if two polynomials in x are distinct then one is greater than the other for all values of x beyond[0] some value x_0.

DEFINITION 17 (POLYNOMIAL ORDERINGS) Let each function symbol in $\mathcal{F}$ be interpreted by a polynomial function. Then $\succ_{pol}$ is a polynomial ordering on $\mathcal{T}(\mathcal{F}, \{\nu\})$ given by

$$t \succ_{pol} u \quad \text{iff} \quad \exists x_0 \in \mathbb{R} : \forall x > x_0 : [\![t]\!](x) > [\![u]\!](x).$$

[0]Since there are no constant symbols we need not carry the baggage of domain restrictions, and can use instead the original formulation due to Lankford.

◇

Note that closure under context and substitution require that each function $[\![f_i]\!]$ be strictly increasing. Moreover, the subterm property requires that $[\![f_i]\!](x) > x$ for all $f_i \in \mathcal{F}$.

Notice that the 'eventually dominates' ordering on unary polynomials is simply the length-then-lex(left) ordering on their coefficients:

$$\begin{aligned} & a_m x^m + \cdots + a_1 x + a_0 \;\; >_p \;\; b_n x^n + \cdots + b_1 x + b_0 \\ \text{iff} \qquad & (a_m, \ldots, a_1, a_0) \;\; \langle >_{len}; >^{\mathsf{lexL}} \rangle \;\; (b_n, \ldots, b_1, b_0) \end{aligned}$$

In general, for a given polynomial interpretation on $\mathcal{F}$, two or more terms may be interpreted as the same polynomial ($t \sim_{pol} u$ iff $[\![t]\!] = [\![u]\!]$) giving a non-total polynomial ordering. However, polynomial orderings satisfy the conditions of Lemma 7 in [Mar93], and so any polynomial ordering on monadic terms can be extended to a total simplification ordering. In particular, for any polynomial ordering $\succ_{pol}$ on $\mathcal{T}$, the lexicographic combination $\langle \succsim_{pol}; \rhd^{\mathsf{lexL}} \rangle$ is total, where $\succsim_{pol}$ is the pre-order associated with $\succ_{pol}$ and $\rhd^{\mathsf{lexL}}$ is the lexicographic ordering from the left of any total precedence $\rhd$ on $\mathcal{F}$. Note that since the ordering on unary polynomials is total, so is the polynomial pre-order on terms.

An ordering $\succ$ on monadic terms is said to have the *cancellation property* if for all $f \in \mathcal{F}$ we have $f(t) \succ f(u)$ implies $t \succ u$ and also $t[f(v)] \succ u[f(v)]$ implies $t[v] \succ u[v]$.

5 LEMMA

Polynomial orderings have the cancellation property.

PROOF Assume that a polynomial ordering $\succ$ does not have the cancellation property. Then for some $t, u \in \mathcal{T}$, $f \in \mathcal{F}$ we have $t \not\succ u$ and either

$t[f(v)] \succ u[f(v)]$ or $f(t) \succ f(u)$. Since the polynomial pre-order is total, either $t \sim u$ or $t \prec u$.

However, $t \sim u$ implies $[\![t]\!] = [\![u]\!]$,

implies $[\![t]\!] \circ [\![f(v)]\!] = [\![u]\!] \circ [\![f(v)]\!]$,

implies $[\![t[f(v)]]\!] = [\![u[f(v)]]\!]$,

implies $t[f(v)] \sim u[f(v)]$,

and $t \prec u$ implies $[\![t]\!] < [\![u]\!]$,

implies $[\![t]\!] \circ [\![f(v)]\!] < [\![u]\!] \circ [\![f(v)]\!]$,

implies $[\![t[f(v)]]\!] < [\![u[f(v)]]\!]$,

implies $t[f(v)] \prec u[f(v)]$,

contradicting the assumption. The argument is similar for composition from the left. Therefore every polynomial ordering has the cancellation property.

□

With monadic polynomial orderings defined, we prepare in the next section to analyse what orderings are possible.

4.2 Ordering Invariants

In this section we give the definitions and lemmas used in the Section 4.3.0 to analyse the order types, numeric invariants, and lexicographic extensions of the polynomial orderings.

The logical and numeric 'invariants' of an ordering $\succ$ are fixed for all extensions of $\succ$, signify fundamental properties of all total extensions of $\succ$, and will allow us to partition and categorise the polynomial orderings on monadic terms.

4.2.0 Order Types

Since our polynomial orderings are formulated (in Definition 16) to be simplification orderings, they are guaranteed to be well-founded. In addition, we can always extend a polynomial ordering to a total ordering (from results in [Mar93]), and all such extensions share the same invariants (trivially, since the invariants define properties of the total extensions). Thus for the analysis of invariants it is valid to treat the polynomial orderings as total.

Since all the orderings we will be analysing are well-founded and extendible to total orderings, the only order types we need consider are ordinals. Moreover, it is known from [MS93] that the only order types that can occur for orderings on monadic terms in two function symbols are ω, ω^2, and ω^ω. Therefore a simple description of order types will suffice, and for a fuller account of order types the reader is directed to, for example, [Wil65].

An ordered set $(S, \succ)$ has order type ω iff it is order-isomorphic to the usual ordering on the natural numbers, $(\mathbb{N}, >)$, i.e. there is an order-preserving bijection between $(S, \succ)$ and $(\mathbb{N}, >)$. An ordered set $(S, \succ)$ has order type ω^2 iff it is order-isomorphic to $(\mathbb{N}^2, >^{\mathsf{lexL}})$, the ordering on pairs given by first comparing the left components. An ordered set $(S, \succ)$ has order type ω^ω iff it is order-isomorphic to $(\mathbb{N}^+, \langle \geqslant_{len}; >^{\mathsf{lexL}} \rangle)$, the ordering on non-negative-length tuples of natural numbers given by first comparing lengths and then ordering lexicographically tuples having the same length. $\geqslant_{len}$ and $>^{\mathsf{lex}}$ are the length pre-order and lexicographic ordering.

To deduce the order types of the monadic polynomial orderings, we will use the following theorem from [MS93].

6 THEOREM (SCOTT)

Let $\succ$ be a total reduction ordering on monadic terms $\mathcal{T}(\{\mathsf{f}, \mathsf{g}\}, \{\mathsf{v}\})$ with

$\mathsf{f}(v) \succ \mathsf{g}(v)$. *Then* $\succ$ *has order type* ω, ω^2, *or* ω^ω. *More precisely,*

$$
\begin{array}{lllll}
(\mathcal{T},\succ) \text{ has order type } \omega & \text{iff} & \mathsf{g}^j \succ \mathsf{f} & \text{for some } j \in \mathbb{N}, & \\
(\mathcal{T},\succ) \text{ has order type } \omega^2 & \text{iff} & \mathsf{f} \succ \mathsf{g}^j & \text{for all } j \in \mathbb{N} \text{ and} & \\
 & & \text{both } \mathsf{g}^k\mathsf{f} \succ \mathsf{fg} & \text{for some } k \in \mathbb{N} & \\
 & & \text{and } \mathsf{fg}^k \succ \mathsf{gf} & \text{for some } k \in \mathbb{N}, & \\
(\mathcal{T},\succ) \text{ has order type } \omega^\omega & \text{iff} & \mathsf{f} \succ \mathsf{g}^j & \text{for all } j \in \mathbb{N} \text{ and} & \\
 & & \text{either } \mathsf{fg} \succ \mathsf{g}^k\mathsf{f} & \text{for all } k \in \mathbb{N} & (1) \\
 & & \text{or } \mathsf{gf} \succ \mathsf{fg}^k & \text{for all } k \in \mathbb{N}. & (2)
\end{array}
$$

Moreover, case (1) *is the left recursive path ordering*[1], *and case* (2) *is the right recursive path ordering.*

This theorem enables us to deduce the order type of a well-order on monadic terms over two-letter alphabets by comparing certain terms in the well-order. By partitioning interpretations into homogeneous sets we will be able to calculate general polynomial expressions for the above terms, and so deduce their order types.

4.2.1 Numeric Invariants

Once the order type has been determined, an appropriate numeric invariant will give a finer classification. We use the invariants τ and λ from [MS93] for orderings of order types ω and ω^2 respectively.

[1] Formulated on p 27.

DEFINITION 18 Given a real number τ ($0 \leqslant \tau \leqslant 1$), the pre-order $\succsim_\tau$ on monadic terms $\mathcal{T}(\{\mathsf{f},\mathsf{g}\},\{v\})$ is defined by

$$t \succsim_\tau u \quad \text{iff} \quad \#(\mathsf{f},t) + \tau.\#(\mathsf{g},t) \geqslant \#(\mathsf{f},u) + \tau.\#(\mathsf{g},u).$$

$\diamond$

This is a canonical pre-order by weight, where τ is the ratio $\frac{\mathsf{wt(g)}}{\mathsf{wt(f)}}$. Having $\tau = 1$ defines a pre-order by length, and having $\tau = 0$ defines a pre-order by the number of f's (ignoring g's). Clearly permutations are equivalent under τ pre-orders. In addition the strict part is not total if τ is rational. For example, if $\tau = 0{\cdot}5$ then $\mathsf{gfg} \sim_\tau \mathsf{ff}$.

Proofs of these lemmata are in [MS93].

7 LEMMA

The ordering $\succ_\tau$ is total up to permutations iff τ is irrational.

8 LEMMA

If $\succ$ is a total simplification ordering of order type ω on monadic terms $\mathcal{T}(\{\mathsf{f},\mathsf{g}\},\{v\})$ with $\mathsf{f}(v) \succ \mathsf{g}(v)$ then $\succ = \langle \succsim_\tau ; \sqsupset \rangle$ for some $0 < \tau \leqslant 1$ and some transitive relation $\sqsupset$.

Thus every well-order of order type ω contains a τ pre-order. Later we will determine τ and $\sqsupset$ for any polynomial ordering $\succ$ of order type ω.

DEFINITION 19 Given a real number $\lambda > 0$, the pre-order $\succsim_\lambda$ on monadic terms $\mathcal{T}(\{\mathsf{f},\mathsf{g}\},\{v\})$ is defined by $\mathsf{g}^{p_0}\mathsf{f}\mathsf{g}^{p_1}\mathsf{f}\ldots\mathsf{f}\mathsf{g}^{p_m} \succsim_\lambda \mathsf{g}^{q_0}\mathsf{f}\mathsf{g}^{q_1}\mathsf{f}\ldots\mathsf{f}\mathsf{g}^{q_n}$ iff $m > n$ or both $m = n$ and $p_n\lambda^n + \cdots + p_1\lambda + p_0 \geqslant q_n\lambda^n + \cdots + q_1\lambda + q_0$. $\diamond$

A λ pre-order first orders by the number of f's and then by the number of g's biased according to position among the f's. In general the strict part of a λ pre-order is not total, but we have the following, with proofs in [MS93].

9 LEMMA

The ordering $\succ_\lambda$ is total iff λ is transcendental.

10 LEMMA

If $\succ$ is a total simplification ordering of order type ω^2 on monadic terms $\mathcal{T}(\{f, g\}, \{v\})$ with $f(v) \succ g(v)$ then $\succ = \langle \succsim_\tau ; \succsim_\lambda ; \sqsupset \rangle$ for $\tau = 0$, some $\lambda > 0$, and some transitive relation $\sqsupset$.

As with $\succsim_\tau$ above, the extension $\sqsupset$ may be the trivial relation that makes all terms equivalent. Later we will determine λ and $\sqsupset$ for any polynomial ordering $\succ$ of order type ω^2.

4.2.2 Lexicographic Extensions

We now come to the final set of tools required for our analysis: those needed for identifying the relation that extends a polynomial ordering from its invariant pre-order. Identifying the extension defined by ω polynomial orderings would be rather simpler if equivalence was due only to permutations, since the form of equivalent terms would be more amenable to direct analysis. As noted above, however, non-permutable terms may also be equivalent under a τ pre-order. Therefore we employ a less direct test by way of a so-called $\rhd^{\mathsf{min}}_{\mathsf{max}}$ relation that is sufficient to prove the extension is a lexicographic ordering.

Given a simplification ordering $\succ$, a pre-order $\succsim$, and a lexicographic ordering $\rhd^{\mathsf{lex}}$ all on $\mathcal{T}$, we will want to be able to characterise when $\succ = \langle \succsim ; \rhd^{\mathsf{lex}} \rangle$. This will enable us to characterise when $\succ_{pol} = \langle \succsim_\tau ; \rhd^{\mathsf{lexR}} \rangle$, where $\succ_{pol}$ is a polynomial ordering of order type ω.

To assist this characterisation, we define an auxiliary relation $\rhd^{\mathsf{min}}_{\mathsf{max}}$ on $\mathcal{T}$ in terms of a precedence $\rhd$ on $\mathcal{F}$, and an equivalence relation $\sim$ and simplification ordering $\succ$ on $\mathcal{T}$. In this context we use $\widetilde{t}$ to denote the

equivalence class under $\sim$ containing t, that is, the set $\{s \in \mathcal{T} \mid s \sim t\} \in \mathcal{T}/\sim$.

DEFINITION 20 ($\rhd^{\mathsf{min}}_{\mathsf{max}}$) Let $\succsim$ be a given pre-order on $\mathcal{T}$. If every equivalence class $C \in \mathcal{T}/\sim$ has a minimum $min_{\succ}$ and a maximum $max_{\succ}$ (both unique), then $\rhd^{\mathsf{min}}_{\mathsf{max}}$ is defined by

$$t \rhd^{\mathsf{min}}_{\mathsf{max}} u$$
$$\text{iff} \quad t = t'[f_i(v)],\ t' = min_{\succ}\ \widetilde{t'},\ u = u'[f_j(v)],\ u' = max_{\succ}\ \widetilde{u'},\ f_i \rhd f_j.$$

$\diamond$

Clearly if $\rhd^{\mathsf{min}}_{\mathsf{max}}$ is defined, we have $\rhd^{\mathsf{min}}_{\mathsf{max}} \subset \rhd^{\mathsf{lexR}}$. The corresponding formulation of $\rhd^{\mathsf{min}}_{\mathsf{max}}$ from the left defines a different ordering, but as we will see in Section 4.3.2 we will be concerned only with comparisons from the right.

The following lemma utilises $\rhd^{\mathsf{min}}_{\mathsf{max}}$ to provide a test for when one simplification ordering is a lexicographic extension of another simplification ordering.

11 LEMMA ($\rhd^{\mathsf{lex}}$ CONTAINMENT)
Let $\mathcal{T}$ be the set of (finite) monadic terms $\mathcal{T}(\{\mathsf{f},\ \mathsf{g}\}, \{v\})$. Let $\succsim$ be a total pre-order on $\mathcal{T}$ with $\succ$ having the subterm property, let $\rhd$ be a total precedence on $\{\mathsf{f}, \mathsf{g}\}$, and let $\succ$ be a simplification ordering on $\mathcal{T}$. Then

$$\succ = \langle \succsim; \rhd^{\mathsf{lexR}} \rangle \quad \textit{iff} \quad \langle \succsim; \rhd^{\mathsf{min}}_{\mathsf{max}} \rangle \subseteq \succ$$

PROOF The proof is by showing that 0. implies 1. implies 2. implies 0.

0. $\langle \succsim; \rhd^{\mathsf{lexR}} \rangle \subseteq \succ$,

1. $\langle \succsim; \rhd^{\mathsf{lexR}} \rangle = \succ$,

2. $\langle \succsim; \rhd^{\mathsf{min}}_{\mathsf{max}} \rangle \subseteq \succ$.

For a total precedence $\rhd$, the lexicographic ordering $\rhd^{\mathsf{lexR}}$ is total. There-

fore, for any total pre-order $\succsim$, the ordering $\langle \succsim; \rhd^{\mathsf{lexR}} \rangle$ is total and so has no proper extension (without extending the domain). Therefore $\succ$ is not a proper extension of $\langle \succsim; \rhd^{\mathsf{lexR}} \rangle$, and so 0. implies 1.

It is obvious (from the formulations of the orderings) that $\langle \succsim; \rhd^{\mathsf{min}}_{\mathsf{max}} \rangle \subseteq \langle \succsim; \rhd^{\mathsf{lexR}} \rangle$, and so 1. implies 2.

To show 2. implies 0., it suffices to show that $\langle \sim; \rhd^{\mathsf{min}}_{\mathsf{max}} \rangle \subseteq \succ$ implies $\langle \sim; \rhd^{\mathsf{lexR}} \rangle \subseteq \succ$. Suppose $\langle \sim; \rhd^{\mathsf{min}}_{\mathsf{max}} \rangle \subseteq \succ$ and that $t \langle \sim; \rhd^{\mathsf{lexR}} \rangle u$ for some $t, u \in \mathcal{T}(\mathcal{F}, \mathcal{V})$. Since $\succ$ (the strict part of $\succsim$) has the subterm property, u is not a subterm of t, and we write $t = t'[f_i(v)]$ and $u = u'[f_j(v)]$ with $f_i \rhd f_j$ (common suffixes can be removed since $\rhd^{\mathsf{lexR}}$ has the cancellation property from the right).

Then
$$\begin{aligned} t'[f_i(v)] &\succeq (min_{\succ}\ \widetilde{t'})[f_i(v)] && \text{by def } min_{\succ}, \\ &\succ (max_{\succ}\ \widetilde{u'})[f_j(v)] && \text{by def } \rhd^{\mathsf{min}}_{\mathsf{max}}, \text{ and } \rhd^{\mathsf{min}}_{\mathsf{max}} \subseteq \succ, \\ &\succeq u'[f_j(v)] && \text{by def } max_{\succ}. \end{aligned}$$
Thus $t \succ u$, and so $\langle \succsim; \rhd^{\mathsf{lexR}} \rangle \subseteq \succ$. □

We will use this result in Section 4.3.2 to show that certain polynomial orderings of order type ω are of the form $\langle \succsim_\tau; \rhd^{\mathsf{lexR}} \rangle$.

4.3 Two Unary Function Symbols

In this section we use the lemmas of Section 4.2 to analyse the order types, numeric invariants, and lexicographic extensions of polynomial orderings on monadic terms $\mathcal{T}(\{\mathsf{f}, \mathsf{g}\}, \{v\})$ over a 2-letter alphabet, $\{\mathsf{f}, \mathsf{g}\}$.

Let $\succ_{pol}$ be a polynomial ordering on monadic terms $\mathcal{T}(\{\mathsf{f}, \mathsf{g}\}, \{v\})$ defined

by the interpretations

$$[\![\mathsf{f}]\!](x) = a_m x^m + \cdots + a_1 x + a_0, \qquad a_m \ldots a_0 \in \mathbb{R},$$
$$[\![\mathsf{g}]\!](x) = b_n x^n + \cdots + b_1 x + b_0, \qquad b_n \ldots b_0 \in \mathbb{R},$$

and $[\![\mathsf{v}]\!](x) = x$, such that $\mathsf{f}(\mathsf{v}) \succ_{pol} \mathsf{g}(\mathsf{v}) \succ_{pol} \mathsf{v}$. This means that

$$[\![\mathsf{f}]\!] >_p [\![\mathsf{g}]\!] >_p [\![\mathsf{v}]\!]$$

(where $>_p$ is the 'eventually dominates' ordering on polynomials of Definition 17), that is

$$(a_m, \ldots, a_0) \ \langle >_{len}; >^{\mathsf{lexL}} \rangle \ (b_n, \ldots, b_0) \ \langle >_{len}; >^{\mathsf{lexL}} \rangle \ (1, 0).$$

As seen above, a polynomial interpretation for a unary function symbol is defined by means of a sequence of parameters: the index of the greatest monomial and the values of the coefficients. There are three classes (in Definition 16) into which a polynomial interpretation may fall: non-linear, weakly linear, and strongly linear. By removing symmetric cases,[2] we can partition the set of monadic polynomial orderings on $\mathcal{T}(\{\mathsf{f}, \mathsf{g}\}, \{\mathsf{v}\})$ into six classes according to the range in which the parameters of the (two) constituent interpretations lie, resulting in the six subsets we will label A, B, C, D, E, and F as specified below. By considering an arbitrary ordering in each class (e.g. $\succ_{polA} \in A$) we will be able to categorise all total extensions (represented by an arbitrary total ordering $\succ$) of that ordering, with the results recorded in Table 4.0.

In most of the calculations that follow, the result is derived from the leading monomials involved, with little or no part played by the lesser monomials. Therefore to aid clarity we use the notation $\overset{d}{\blacktriangledown}$ to denote a polynomial of degree less than d if $d > 0$; otherwise $\overset{d}{\blacktriangledown}$ denotes 0.

[2] i.e. orderings that are the same up to consistent renaming of function symbols

	Parameters	Lexicographic Combination	Order Type	*NumericInvariant*	Precedence
$\succ_{polA}$	$m \geqslant n > 1$	$\langle \succsim_\tau ; \rhd^{\mathsf{lexR}} \rangle$ †	ω	$\tau = \frac{\ln n}{\ln m}$	$f \trianglerighteq g$ if $\frac{\ln a_m}{m-1} \geqslant \frac{\ln b_n}{n-1}$
$\succ_{polB}$	$m > n = 1, \quad b_1 > 1$	$\langle \succsim_\tau ; \succsim_\lambda ; \rhd^{\mathsf{lexR}} \rangle$ †	ω^2	$\lambda = m$	$f \trianglerighteq g$ if $\frac{a_{m-1}}{ma_m} \geqslant \frac{b_0}{b_1-1}$
$\succ_{polC}$	$m > n = 1, \quad b_1 = 1, \quad b_0 > 0$	$\langle \rhd^{\mathsf{rpL}} \rangle$	ω^ω	–	$f \rhd g$
$\succ_{polD}$	$m = n = 1, \quad a_1 \geqslant b_1 > 1$	$\langle \succsim_\tau ; \rhd^{\mathsf{lexR}} \rangle$ †	ω	$\tau = \frac{\ln b_1}{\ln a_1}$	$f \trianglerighteq g$ if $\frac{a_0}{a_1-1} \geqslant \frac{b_0}{b_1-1}$
$\succ_{polE}$	$m = n = 1, \quad a_1 > b_1 = 1, \quad b_0 > 0$	$\langle \succsim_\tau ; \succsim_\lambda \rangle$	ω^2	$\lambda = a_1$	–
$\succ_{polF}$	$m = n = 1, \quad a_1 = b_1 = 1, \quad a_0 \geqslant b_0 > 0$	$\langle \succsim_\tau \rangle$	ω	$\tau = \frac{b_0}{a_0}$	–

Table 4.0: Polynomial orderings on monadic terms over 2-letter alphabet.

†Subject to conditions detailed in Lemmas 24, 25, and 26.

4.3.0 Order Types

We begin the analysis of monadic polynomial orderings by determining the order type of each monadic polynomial ordering.

Let the total ordering $\succ$ on monadic terms $\mathcal{T}(\{f,g\},\{v\})$ be a total extension of a polynomial ordering $\succ_{pol}$, with $f \succ_{pol} g \succ_{pol} v$. For each of the six cases (as partitioned below) the order type of $\succ$ is deduced by comparing expressions according to Theorem 6 on page 77.

12 LEMMA (ORDER TYPE OF $\succ_{polA}$)

If $m \geqslant n > 1$ then $\succ$ has order type ω.

13 LEMMA (ORDER TYPE OF $\succ_{polB}$)

If $m > n = 1$ and $b_1 > 1$ then $\succ$ has order type ω^2.

14 LEMMA (ORDER TYPE OF $\succ_{polC}$)

If $m > n = 1$ and $b_1 = 1$ and $b_0 > 0$ then $\succ$ has order type ω^ω.

15 LEMMA (ORDER TYPE OF $\succ_{polD}$)

If $m = n = 1$ and $a_1 \geqslant b_1 > 1$ then $\succ$ has order type ω.

16 LEMMA (ORDER TYPE OF $\succ_{polE}$)

If $m = n = 1$ and $a_1 > b_1 = 1$ and $b_0 > 0$ then $\succ$ has order type ω^2.

17 LEMMA (ORDER TYPE OF $\succ_{polF}$)

If $m = n = 1$ and $a_1 = b_1 = 1$ and $a_0 \geqslant b_0 > 0$ then $\succ$ has order type ω.

The six lemmata above give the order types of *total* extensions of polynomial orderings. However, they show that *all* total extensions of a given polynomial ordering have the same order type, and we are therefore justified in associating that order type (an ordinal) with the original (not necessarily total) polynomial ordering.

The proofs of Lemma 12, Lemma 15, and Lemma 17 are similar, as are the proofs of Lemma 13 and Lemma 16:

PROOF (of Lemma 12) In this case

$$[\![\mathsf{f}]\!](x) = a_m x^m + \overset{m}{\blacktriangledown}, \qquad [\![\mathsf{g}^k]\!](x) = b_n^{\frac{n^k-1}{n-1}} x^{n^k} + \overset{n^k}{\blacktriangledown} \quad \forall k \in \mathbb{N}_+,$$

giving $\mathsf{g}^k \succ_{polA} \mathsf{f}$ if $b_n^{\frac{n^k-1}{n-1}} x^{n^k} >_p a_m x^m$,

if $n^k > m$,

if $k > \frac{\ln m}{\ln n}$.

Thus in $\succ_{polA}$, f is bounded above by g^k for all $k > \frac{\ln m}{\ln n}$, and so $\succ$ has order type ω. □

PROOF (of Lemma 13) In this case

$$[\![\mathsf{f}]\!](x) = a_m x^m + \overset{m}{\blacktriangledown}, \qquad [\![\mathsf{g}^k]\!](x) = b_1^k x + \tfrac{b_1^j - 1}{b_1 - 1} b_0 \quad \forall k \in \mathbb{N}_+,$$

$$[\![\mathsf{fg}]\!](x) = a_m b_1^m x^m + \overset{m}{\blacktriangledown}, \qquad [\![\mathsf{gf}]\!](x) = a_m b_1 x^m + \overset{m}{\blacktriangledown},$$

$$[\![\mathsf{g}^i\mathsf{f}]\!](x) = a_m b_1^i x^m + \overset{m}{\blacktriangledown} \ \forall i \in \mathbb{N}, \qquad [\![\mathsf{fg}^j]\!](x) = a_m b_1^{mj} x^m + \overset{m}{\blacktriangledown} \quad \forall j \in \mathbb{N},$$

giving $\mathsf{f} \succ_{polB} \mathsf{g}^k$ if $a_m x^m >_p b_1 x$,

if $m > 1$,

and $\mathsf{g}^i\mathsf{f} \succ_{polB} \mathsf{fg}$ if $a_m b_1^i x^m >_p a_m b_1^m x^m$,

if $b_1^i > b_1^m$,

if $i > m$,

and $\mathsf{fg}^j \succ_{polB} \mathsf{gf}$ if $a_m b_1^{mj} x^m >_p a_m b_1 x^m$,

if $b_1^{mj} > b_1$,

if $mj > 1$,

if $j > 0$.

Thus in $\succ_{polB}$, f is not bounded above by g^k for any k, but fg is bounded above by $g^i f$ for all $i > m$ and gf is bounded above by fg^j for all $j > 0$, and so $\succ$ has order type ω^2. □

PROOF (of Lemma 14) In this case

$$[\![fg]\!](x) = a_m x^m + (ma_m b_0 + a_{m-1})x^{m-1} + \overset{m-1}{\blacktriangledown},$$
$$[\![g^i f]\!](x) = a_m x^m + a_{m-1}x^{m-1} + \overset{m}{\blacktriangledown} \qquad \forall i \in \mathbb{N},$$

giving $fg \succ_{polC} g^i f$ if $(ma_m b_0 + a_{m-1})x^{m-1} >_p a_{m-1}x^{m-1}$,

if $ma_m b_0 + a_{m-1} > a_{m-1}$,

if $ma_m b_0 > 0$.

Thus in $\succ_{polC}$, fg is not bounded above by $g^i f$ for any $i \in \mathbb{N}$, and so $\succ$ has order type ω^ω. □

PROOF (of Lemma 15) In this case

$$[\![f]\!](x) = a_1 x + a_0, \qquad [\![g^j]\!](x) = b_1^j x + \tfrac{b_1^j - 1}{b_1 - 1} b_0 \quad \forall j \in \mathbb{N}_+,$$

giving $g^j \succ_{polD} f$ if $b_1^j x > a_1 x$,

if $\ln b_1^j > \ln a_1$,

if $j > \frac{\ln a_1}{\ln b_1}$.

Thus in $\succ_{polD}$, f is bounded above by g^j for any $j > \frac{\ln a_1}{\ln b_1}$, and so $\succ$ has order type ω. □

PROOF (of Lemma 16) In this case

$$[\![f]\!](x) = a_1 x + a_0, \qquad [\![g^k]\!](x) = x + kb_0 \qquad \forall k \in \mathbb{N}_+,$$
$$[\![fg]\!](x) = a_1 x + a_1 b_0 + a_0, \qquad [\![gf]\!](x) = a_1 x + a_0 + b_0,$$
$$[\![g^i f]\!](x) = a_1 x + a_0 + ib_0 \quad \forall i \in \mathbb{N}, \qquad [\![fg^j]\!](x) = a_1 x + ja_1 b_0 + a_0 \quad \forall j \in \mathbb{N}.$$

giving $\mathsf{f} \succ_{polE} \mathsf{g}^k$ if $a_1 x > x$,

if $a_1 > 1$.

and $\mathsf{g}^i\mathsf{f} \succ_{polE} \mathsf{fg}$ if $a_0 + ib_0 > a_1 b_0 + a_0$,

if $ib_0 > a_1 b_0$,

if $i > a_1$.

and $\mathsf{fg}^j \succ_{polE} \mathsf{gf}$ if $ja_1 b_0 + a_0 > a_0 + b_0$,

if $ja_1 b_0 > b_0$,

if $ja_1 > 1$,

if $j > 1$.

Thus in $\succ_{polE}$, f is not bounded above by g^k for any k, but fg is bounded above by $\mathsf{g}^i\mathsf{f}$ for any $i > a_1$ and gf is bounded above by fg^j for any $j > 1$, and so $\succ$ has order type ω^2. □

PROOF (of Lemma 17) In this case

$$[\![\mathsf{f}]\!](x) = x + a_0, \qquad [\![\mathsf{g}^k]\!](x) = x + kb_0 \quad \forall k \in \mathbb{N}_+,$$

giving $\mathsf{g}^k \succ_{polF} \mathsf{f}$ if $kb_0 > a_0$,

if $k > \frac{a_0}{b_0}$.

Thus in $\succ_{polF}$, f is bounded above by g^k for any $k > \frac{a_0}{b_0}$, and so $\succ$ has order type ω. □

Now we know the order type of all total extensions for each monadic polynomial ordering. In the sequel we will see exactly when a polynomial ordering is total on monadic terms $\mathcal{T}(\{\mathsf{f}, \mathsf{g}\}, \{v\})$, and when two polynomial orderings are distinct.

4.3.1 Numeric Invariants

Knowing the order type of a polynomial ordering tells us about the ordering's structure. With this information we can proceed to identify the ordering that results from any given choice of parameters (the polynomial interpretations assigned to function symbols). The next step towards this goal is to determine the numeric invariants of the orderings within each class.

For the three cases having order type ω, the numeric invariant (τ) is deduced by comparing expressions according to Definition 18 and Lemma 8 on page 79. Similarly for the two cases having order type ω^2, the numeric invariant (λ) is deduced by comparing expressions according to Definition 19 and Lemma 10 on page 80. As before, $(a_m, \ldots, a_0)$ are the coefficients of $[\![\mathrm{f}]\!](x)$ and $(b_n, \ldots, b_0)$ are the coefficients of $[\![\mathrm{g}]\!](x)$, and $\succ_{pol}$ is the resulting (not necessarily total) polynomial ordering.

18 LEMMA (TAU OF $\succ_{polA}$)

If $m \geqslant n > 1$ *then* $\succ_{pol}$ *has* $\tau = \frac{\ln n}{\ln m}$.

19 LEMMA (LAMBDA OF $\succ_{polB}$)

If $m > n = 1$ *and* $b_1 > 1$ *then* $\succ_{pol}$ *has* $\lambda = m$.

20 LEMMA (TAU OF $\succ_{polD}$)

If $m = n = 1$ *and* $a_1 \geqslant b_1 > 1$ *then* $\succ_{pol}$ *has* $\tau = \frac{\ln b_1}{\ln a_1}$.

21 LEMMA (LAMBDA OF $\succ_{polE}$)

If $m = n = 1$ *and* $a_1 > b_1 = 1$ *(and* $b_0 > 0$*) then* $\succ_{pol}$ *has* $\lambda = a_1$.

22 LEMMA (TAU OF $\succ_{polF}$)

If $m = n = 1$ *and* $a_1 = b_1 = 1$ *and* $a_0 \geqslant b_0 > 0$ *then* $\succ_{pol}$ *has* $\tau = \frac{b_0}{a_0}$.

The proofs of Lemma 18, Lemma 20, and Lemma 22 are similar, as are the proofs of Lemma 19 and Lemma 21:

PROOF (of Lemma 18) In this case

$$[\![t]\!](x) \;=\; a_m^A b_n^B x^C + \overset{C}{\blacktriangledown},$$

for some $A, B \in \mathbb{N}$ and $C = m^{\#(\mathsf{f},t)} n^{\#(\mathsf{g},t)}$,

giving

$$\begin{aligned} t \succ_{pol} u \quad &\text{if} \quad m^{\#(\mathsf{f},t)} n^{\#(\mathsf{g},t)} > m^{\#(\mathsf{f},u)} n^{\#(\mathsf{g},u)}, \\ &\text{if} \quad \#(\mathsf{f},t)\ln m + \#(\mathsf{g},t)\ln n > \#(\mathsf{f},u)\ln m + \#(\mathsf{g},u)\ln n, \\ &\text{if} \quad \#(\mathsf{f},t) + \#(\mathsf{g},t)\tfrac{\ln n}{\ln m} > \#(\mathsf{f},u) + \#(\mathsf{g},u)\tfrac{\ln n}{\ln m}. \end{aligned}$$

Thus $t \succ_\tau u$ implies $t \succ_{pol} u$ for $\tau = \frac{\ln n}{\ln m}$. □

PROOF (of Lemma 19) In this case

$$[\![\mathsf{g}^i\mathsf{f}]\!](x) \;=\; a_m b_1^i x^m + \overset{m}{\blacktriangledown} \quad \forall i \in \mathbb{N}, \qquad [\![\mathsf{f}\mathsf{g}^j]\!](x) \;=\; a_m b_1^{mj} x^m + \overset{m}{\blacktriangledown} \quad \forall j \in \mathbb{N}.$$

giving

$$\begin{aligned} \mathsf{g}^i\mathsf{f} \succsim \mathsf{f}\mathsf{g}^j \quad &\text{if} \quad a_m b_1^i x^m \gtrsim a_m b_1^{mj} x^m, \\ &\text{if} \quad b_1^i \gtrsim b_1^{mj}, \\ &\text{if} \quad i \gtrsim mj. \end{aligned}$$

Thus $i \gtrsim \lambda j$ implies $\mathsf{g}^i\mathsf{f} \succsim \mathsf{f}\mathsf{g}^j$ for $\lambda = m$. □

PROOF (of Lemma 20) In this case

$$[\![t]\!](x) \;=\; a_1^{\#(\mathsf{f},t)} b_1^{\#(\mathsf{g},t)} x + \overset{1}{\blacktriangledown}.$$

giving

$$\begin{aligned} t \succ_{pol} u \quad &\text{if} \quad a_1^{\#(\mathsf{f},t)} b_1^{\#(\mathsf{g},t)} > a_1^{\#(\mathsf{f},u)} b_1^{\#(\mathsf{g},u)}, \\ &\text{if} \quad \#(\mathsf{f},t)\ln a_1 + \#(\mathsf{g},t)\ln b_1 > \#(\mathsf{f},u)\ln a_1 + \#(\mathsf{g},u)\ln b_1, \\ &\text{if} \quad \#(\mathsf{f},t) + \#(\mathsf{g},t)\tfrac{\ln b_1}{\ln a_1} > \#(\mathsf{f},u) + \#(\mathsf{g},u)\tfrac{\ln b_1}{\ln a_1}. \end{aligned}$$

Thus $t \succ_\tau u$ implies $t \succ_{pol} u$ for $\tau = \frac{\ln b_1}{\ln a_1}$. □

PROOF (of Lemma 21) In this case

$$[\![\mathsf{g}^i\mathsf{f}]\!](x) = a_1 x + a_0 + i b_0, \qquad [\![\mathsf{f}\mathsf{g}^j]\!](x) = a_1 x + j a_1 b_0 + a_0 \quad \forall i, j \in \mathbb{N}.$$

$$\begin{aligned} \text{giving } \mathsf{g}^i\mathsf{f} \precsim \mathsf{f}\mathsf{g}^j \quad &\text{if} \quad a_0 + i b_0 \geqslant j a_1 b_0 + a_0, \\ &\text{if} \quad i b_0 \geqslant j a_1 b_0, \\ &\text{if} \quad i \geqslant a_1 j. \end{aligned}$$

Thus $i \geqslant \lambda j$ implies $\mathsf{g}^i\mathsf{f} \precsim \mathsf{f}\mathsf{g}^j$ for any $\lambda = a_1$. □

PROOF (of Lemma 22) In this case

$$[\![t]\!](x) = x + \#(\mathsf{f},t)a_0 + \#(\mathsf{g},t)b_0.$$

$$\begin{aligned} \text{giving } t \succ_{pol} u \quad &\text{if} \quad \#(\mathsf{f},t)a_0 + \#(\mathsf{g},t)b_0 > \#(\mathsf{f},u)a_0 + \#(\mathsf{g},u)b_0, \\ &\text{if} \quad \#(\mathsf{f},t) + \#(\mathsf{g},t)\tfrac{b_0}{a_0} > \#(\mathsf{f},u) + \#(\mathsf{g},u)\tfrac{b_0}{a_0}. \end{aligned}$$

Thus $t \succ_\tau u$ implies $t \succ_{pol} u$ for any $\tau = \frac{b_0}{a_0}$. □

We now know the invariant pre-order, $\precsim_\tau$ or $\precsim_\lambda$, contained by each of the ω and ω^2 orderings respectively. Therefore the orientation of two terms by a polynomial ordering can be predicted up to $\sim_\tau$ or $\sim_\lambda$ without any calculation of polynomials, simply by examination of the defining interpretations. In the sequel we will complete the analysis by determining the orientation of two

terms by a polynomial ordering when they are equivalent under $\succsim_\tau$ or $\succsim_\lambda$.

4.3.2 Lexicographic Extensions

The logical and numeric invariants do not tell the whole story as far as polynomial orderings are concerned. For example, we now know that every weakly linear polynomial ordering (on $\mathcal{T}(\{f, g\}, \{v\})$) is a (not necessarily proper) extension of a tau pre-order,

$$\succsim_\tau \subseteq \succ_{polD},$$

(and for each given $\succ_{polD}$ we know the value of τ) but so far we have not determined if or when the extension is proper. In other words, we know

$$\succ_{polD} = \langle \succsim_\tau ; \sqsupset \rangle$$

for some (possibly trivial) relation $\sqsupset$, and we would like to be able to say more about such extensions. We will see that for $\succ_{polA}$, $\succ_{polB}$, and $\succ_{polD}$ the extensions are lexicographic orderings, and (with respect to conditions below) we can specify the precedences of the lexicographic orderings.

The orderings $\succ_{polE}$ and $\succ_{polF}$ do not determine extensions; they are in fact equal to the pre-orders already deduced. Since those pre-orders have finite equivalence classes[3], any total extension will result in a total well-founded ordering.

In the proofs of Lemma 24 and Lemma 26 we make use of the following lemma.

23 LEMMA

If $a, b \geqslant 2$ *and* $A, B \in \mathbb{N}_+$ *then*

[3] A simple property of τ and λ pre-orders.

$a^{A-1}(a-1)b^{B-1}(b-1) - (a^{A-1}-1)(b^{B-1}-1) > 0.$

PROOF If $a \geqslant 2$ then $a^{A-1}(a-1) > a^{A-1}-1$ since $a^{A-1}(a-1)-(a^{A-1}-1) = a^{A-1}(a-2)+1$. Similarly $b^{B-1}(b-1) > b^{B-1}-1$, and so $a^{A-1}(a-1)b^{B-1}(b-1) > (a^{A-1}-1)(b^{B-1}-1)$. □

The lexicographic extensions are determined as follows.

24 LEMMA (PRECEDENCE IN $\succ_{polA}$)

If $m \geqslant n > 1$ and also $m^p \neq n^q$ for all $p, q \in \mathbb{N}_+$ then

$$\succ_{polA} = \langle \succsim_\tau ; \rhd^{\mathsf{lexR}} \rangle,$$

where $\tau = \frac{\ln n}{\ln m}$, and the precedence $\rhd$ is defined by $\mathsf{f} \bowtie \mathsf{g}$ if $\frac{\ln a_m}{m-1} \gtrless \frac{\ln b_n}{n-1}$.

25 LEMMA (PRECEDENCE IN $\succ_{polB}$)

If $m > n = 1$ and $b_1 > 1$ then

$$\succ_{polB} = \langle \succsim_\tau ; \succsim_\lambda ; \rhd^{\mathsf{lexR}} \rangle,$$

where $\tau = 0$, $\lambda = m$ and the precedence $\rhd$ is defined by $\mathsf{f} \bowtie \mathsf{g}$ if $\frac{a_{m-1}}{ma_m} \gtrless \frac{b_0}{b_1-1}$.

26 LEMMA (PRECEDENCE IN $\succ_{polD}$)

If $m = n = 1$ and $a_1 \geqslant b_1 > 1$ (and $a_1 \geqslant 2$) and also $a_1^p \neq b_1^q$ for all $p, q \in \mathbb{N}_+$, then

$$\succ_{polD} = \langle \succsim_\tau ; \rhd^{\mathsf{lexR}} \rangle,$$

where $\tau = \frac{\ln b_1}{\ln a_1}$ and the precedence $\rhd$ is defined by $\mathsf{f} \bowtie \mathsf{g}$ iff $\frac{a_0}{a_1-1} \gtrless \frac{b_0}{b_1-1}$.

The proofs of Lemma 24 and Lemma 26 are similar.

PROOF (of Lemma 24) The proof is by Lemma 11, showing that $\langle \succ_\tau ; \rhd^{\mathsf{min}}_{\mathsf{max}} \rangle \subseteq \succ_{polA}$. From Lemma 12 and Lemma 18, $\succ_{polA}$ has order type ω and $\succ_\tau \subseteq \succ_{polA}$, and $\succ_\tau$ is closed under subterms (since $\tau \neq 0$). Therefore it remains to show that $\langle \sim_\tau ; \rhd^{\mathsf{min}}_{\mathsf{max}} \rangle \subseteq \succ_{polA}$.

Suppose $t \sim_\tau u$, for $\tau = \frac{\ln n}{\ln m}$. Now τ is irrational if (and only if) $m^p \neq n^q$ for all $p, q \in \mathbb{N}_+$, so by Lemma 7, t is a permutation of u, and we write $A = \#(\mathsf{f}, t) = \#(\mathsf{f}, u)$ and $B = \#(\mathsf{g}, t) = \#(\mathsf{g}, u)$.

We have

$$\begin{aligned}
[\![\mathsf{fg}]\!](x) &= a_m b_n^m x^{mn} + m a_m b_n^{m-1} b_{n-1} x^{mn-1} + \overset{mn-1}{\blacktriangledown}, \\
[\![\mathsf{gf}]\!](x) &= a_m^n b_n x^{mn} + n a_m^{n-1} a_{m-1} b_n x^{mn-1} + \overset{mn-1}{\blacktriangledown},
\end{aligned}$$

giving

$$\begin{aligned}
\mathsf{gf} \precsim \mathsf{fg} \quad &\text{if} \quad a_m^n b_n \geqslant a_m b_n^m, \\
&\text{if} \quad \tfrac{\ln a_m}{m-1} \geqslant \tfrac{\ln b_n}{n-1}, \\
&\text{if} \quad \mathsf{f} \not\rhd \mathsf{g}.
\end{aligned}$$

Since $\succ$ is a reduction ordering, this means that for any $t_1, t_2 \in \mathcal{T}$, $t_1{:}\mathsf{g}{:}\mathsf{f}{:}t_2 \precsim t_1{:}\mathsf{f}{:}\mathsf{g}{:}t_2$ if $\mathsf{f} \not\rhd \mathsf{g}$. Assume wlog. that $\frac{\ln a_m}{m-1} > \frac{\ln b_n}{n-1}$. Then $\mathsf{f} \rhd \mathsf{g}$, $\mathsf{gf} \succ_{polA} \mathsf{fg}$, and so the equivalence class $\widetilde{t}$ is totally ordered under $\succ_{polA}$ with maximum $max_{\succ_{polA}} \widetilde{t} = \mathsf{g}^B \mathsf{f}^A$ and minimum $min_{\succ_{polA}} \widetilde{t} = \mathsf{f}^A \mathsf{g}^B$.

Now suppose $t \, \langle \sim_\tau ; \rhd^{\mathsf{min}}_{\mathsf{max}} \rangle \, u$, for $\tau = \frac{\ln n}{\ln m}$ and $\mathsf{f} \rhd \mathsf{g}$. Then $t = \mathsf{f}^{A-1}\mathsf{g}^B\mathsf{f}$ and $u = \mathsf{g}^{B-1}\mathsf{f}^A\mathsf{g}$. It remains to show that $t \succ_{polA} u$.

We have

$$\begin{aligned}
[\![t]\!](x) &= a_m^{\frac{m^A-1}{m-1} + m^{A-1}(n^B-1)} b_n^{m^{A-1}\frac{n^B-1}{n-1}} x^{m^A n^B} + \overset{m^A n^B}{\blacktriangledown}, \\
[\![u]\!](x) &= a_m^{\frac{m^A-1}{m-1} n^{B-1}} b_n^{(m^A-1)n^{B-1} + \frac{n^B-1}{n-1}} x^{m^A n^B} + \overset{m^A n^B}{\blacktriangledown},
\end{aligned}$$

giving

$$
\begin{aligned}
t \gtrsim u \quad \text{if} \quad & a_m^{\frac{m^A-1}{m-1}+m^{A-1}(n^B-1)} b_n^{m^{A-1}\frac{n^B-1}{n-1}} \\
& \geqslant a_m^{\frac{m^A-1}{m-1}n^{B-1}} b_n^{(m^A-1)n^{B-1}+\frac{n^B-1}{n-1}}, \\
\text{if} \quad & \tfrac{\ln a_m}{m-1}((m^A-1)(1-n^{B-1})+m^{A-1}(n^B-1)(m-1)) \\
& \geqslant \tfrac{\ln b_n}{n-1}((n^B-1)(1-m^{A-1})+n^{B-1}(m^A-1)(n-1)), \\
\text{if} \quad & \tfrac{\ln a_m}{m-1}(m^{A-1}(m-1)n^{B-1}(n-1)-(m^{A-1}-1)(n^{B-1}-1)) \\
& \geqslant \tfrac{\ln b_n}{n-1}(m^{A-1}(m-1)n^{B-1}(n-1)-(m^{A-1}-1)(n^{B-1}-1)), \\
\text{if} \quad & \tfrac{\ln a_m}{m-1} \geqslant \tfrac{\ln b_n}{n-1}.
\end{aligned}
$$

by Lemma 23, and thus $t \succ_{polA} u$. □

PROOF (of Lemma 25) Suppose $t \langle \sim_\tau; \sim_\lambda \rangle u$. Let $k = \#(\mathsf{f}, t) = \#(\mathsf{f}, u)$, and write t and u as

$$t = \mathsf{g}^{l_0}\mathsf{f}\mathsf{g}^{l_1}\ldots\mathsf{g}^{l_{k-1}}\mathsf{f}\mathsf{g}^{l_k} \quad \text{and} \quad u = \mathsf{g}^{j_0}\mathsf{f}\mathsf{g}^{j_1}\ldots\mathsf{g}^{j_{k-1}}\mathsf{f}\mathsf{g}^{j_k},$$

with $\sum_i m^i l_i = \sum_i m^i j_i$. This gives

$$
\begin{aligned}
[\![t]\!](x) \;=\; & a_m^{\frac{m^k-1}{m-1}} b_1^{\sum m^i l_i} x^{m^k} \\
& + (m a_m \tfrac{b_1^{l_k}-1}{b_1-1} b_0 + a_{m-1}) m^{k-1} a_m^{\frac{m^{k-1}}{m-1}-1} b_1^{(\sum m^i l_i)-l_k} x^{m^k-1} + \underset{\blacktriangledown}{m^k-1},
\end{aligned}
$$

$$
\begin{aligned}
\text{giving } t \gtrsim u \quad \text{if} \quad & (m a_m \tfrac{b_1^{l_k}-1}{b_1-1} b_0 + a_{m-1}) b_1^{\sum m^i l_i - l_k} \\
& \geqslant (m a_m \tfrac{b_1^{l_k}-1}{b_1-1} b_0 + a_{m-1}) b_1^{\sum m^i j_i - j_k}, \\
\text{if} \quad & (m a_m \tfrac{b_1^{l_k}-1}{b_1-1} b_0 + a_{m-1}) b_1^{-l_k} \geqslant (m a_m \tfrac{b_1^{j_k}-1}{b_1-1} b_0 + a_{m-1}) b_1^{-j_k}, \\
\text{if} \quad & (\tfrac{a_{m-1}}{m a_m} - \tfrac{b_0}{b_1-1}) b_1^{j_k} \geqslant (\tfrac{a_{m-1}}{m a_m} - \tfrac{b_0}{b_1-1}) b_1^{l_k}, \\
\text{if} \quad & (\tfrac{a_{m-1}}{m a_m} - \tfrac{b_0}{b_1-1}) j_k \geqslant (\tfrac{a_{m-1}}{m a_m} - \tfrac{b_0}{b_1-1}) l_k.
\end{aligned}
$$

Thus $t \langle \sim_\tau; \sim_\lambda; \rhd^{\mathsf{lexR}} \rangle u$ implies $t \gtrsim u$, where $\mathsf{f} \trianglerighteq \mathsf{g}$ iff $\frac{a_{m-1}}{m a_m} \geqslant \frac{b_0}{b_1-1}$. Hence

$\succ_{polB} = \langle \succsim_\tau ; \succsim_\lambda ; \rhd^{\mathsf{lexR}} \rangle$. □

PROOF (of Lemma 26) The proof is by Lemma 11, showing that $\langle \succ_\tau ; \rhd^{\mathsf{min}}_{\mathsf{max}} \rangle \subseteq \succ_{polD}$. From Lemma 15 and Lemma 20, $\succ_{polD}$ has order type ω and $\succ_\tau \subseteq \succ_{polD}$, and $\succ_\tau$ is closed under subterms (since $\tau \neq 0$). Therefore it remains to show that $\langle \sim_\tau ; \rhd^{\mathsf{min}}_{\mathsf{max}} \rangle \subseteq \succ_{polD}$.

Suppose $t \sim_\tau u$, for $\tau = \frac{\ln b_1}{\ln a_1}$. Now τ is irrational if (and only if) $a_1^p \neq b_1^q$ for all $p, q \in \mathbb{N}_+$, so by Lemma 7, t is a permutation of u. Let $A = \#(\mathsf{f}, t) = \#(\mathsf{f}, u)$ and $B = \#(\mathsf{g}, t) = \#(\mathsf{g}, u)$.

We have

$$\begin{aligned} [\![\mathsf{fg}]\!](x) &= a_1 b_1 x + a_1 b_0 + a_0, \\ [\![\mathsf{gf}]\!](x) &= a_1 b_1 x + a_0 b_1 + b_0, \end{aligned}$$

giving

$$\begin{aligned} \mathsf{gf} \succsim \mathsf{fg} \quad &\text{iff} \quad a_0 b_1 + b_0 \gtrsim a_1 b_0 + a_0, \\ &\text{iff} \quad \tfrac{a_0}{a_1 - 1} \gtrsim \tfrac{b_0}{b_1 - 1}, \\ &\text{iff} \quad \mathsf{f} \not\rhd \mathsf{g}. \end{aligned}$$

Since $\succ$ is a reduction ordering, the above shows that for any $t_1, t_2 \in \mathcal{T}$, $t_1{:}\mathsf{g}{:}\mathsf{f}{:}t_2 \succsim t_1{:}\mathsf{f}{:}\mathsf{g}{:}t_2$ iff $\mathsf{f} \not\rhd \mathsf{g}$. Assume wlog. that $\frac{a_0}{a_1 - 1} > \frac{b_0}{b_1 - 1}$. Then $\mathsf{f} \rhd \mathsf{g}$ and $\mathsf{gf} \succ_{polD} \mathsf{fg}$, and so the equivalence class $\widetilde{t}$ is totally ordered under $\succ_{polD}$ with maximum $max_{\succ_{polD}} \widetilde{t} = \mathsf{g}^B \mathsf{f}^A$ and minimum $min_{\succ_{polD}} \widetilde{t} = \mathsf{f}^A \mathsf{g}^B$.

Now suppose $t \langle \sim_\tau ; \rhd^{\mathsf{min}}_{\mathsf{max}} \rangle u$, for $\tau = \frac{\ln b_1}{\ln a_1}$ and $\mathsf{f} \rhd \mathsf{g}$. Then $t = \mathsf{f}^{A-1} \mathsf{g}^B \mathsf{f}$ and $u = \mathsf{g}^{B-1} \mathsf{f}^A \mathsf{g}$. It remains to show that $t \succ_{polD} u$.

We have

$$\begin{aligned} [\![t]\!](x) &= a_1^A b_1^B x + a_1^{A-1} b_1^B a_0 + a_1^{A-1}(b_1^B - 1)\tfrac{b_0}{b_1 - 1} + (a_1^{A-1} - 1)\tfrac{a_0}{a_1 - 1}, \\ [\![u]\!](x) &= a_1^A b_1^B x + b_1^{B-1} a_1^A b_0 + b_1^{B-1}(a_1^A - 1)\tfrac{a_0}{a_1 - 1} + (b_1^{B-1} - 1)\tfrac{b_0}{b_1 - 1}, \end{aligned}$$

giving

$$
\begin{aligned}
t \gtrsim u \quad \text{iff} \quad & a_1^{A-1}b_1^B a_0 + a_1^{A-1}(b_1^B - 1)\tfrac{b_0}{b_1-1} + (a_1^{A-1} - 1)\tfrac{a_0}{a_1-1} \\
& \geqslant b_1^{B-1}a_1^A b_0 + b_1^{B-1}(a_1^A - 1)\tfrac{a_0}{a_1-1} + (b_1^{B-1} - 1)\tfrac{b_0}{b_1-1}, \\
\text{iff} \quad & \tfrac{a_0}{a_1-1}[a_1^{A-1}(a_1 - 1)b_1^{B-1}(b_1 - 1) - (a_1^{A-1} - 1)(b_1^{B-1} - 1)] \\
& \geqslant \tfrac{b_0}{b_1-1}[a_1^{A-1}(a_1 - 1)b_1^{B-1}(b_1 - 1) - (a_1^{A-1} - 1)(b_1^{B-1} - 1)], \\
\text{iff} \quad & \tfrac{a_0}{a_1-1} \geqslant \tfrac{b_0}{b_1-1}.
\end{aligned}
$$

by Lemma 23, and thus $t \succ_{polD} u$. □

This completes the analysis of polynomial orderings on monadic terms over a two-letter alphabet. Each such ordering can be expressed as a lexicographic combination of simpler, more tangible orderings, and these orderings are determined by the leading coefficients of the polynomial interpretations.

4.3.3 Summary for Two Unary Function Symbols

The results that have been presented here for two unary function symbols can be summarised by the following theorem.

27 THEOREM

Let $\succ_{pol}$ be a polynomial ordering on monadic terms $\mathcal{T}(\{\mathsf{f},\mathsf{g}\},\{\mathsf{v}\})$ defined by the interpretations

$$
\begin{aligned}
[\![\mathsf{f}]\!](x) &= a_m x^m + \cdots + a_1 x + a_0, \qquad a_m \ldots a_0 \in \mathbb{R}, \\
[\![\mathsf{g}]\!](x) &= b_n x^n + \cdots + b_1 x + b_0, \qquad b_n \ldots b_0 \in \mathbb{R}, \\
[\![\mathsf{v}]\!](x) &= x
\end{aligned}
$$

such that $(a_m,\ldots,a_0)\ \langle >_{len}; >^{\mathsf{lexL}}\rangle\ (b_n,\ldots,b_0)\ \langle >_{len}; >^{\mathsf{lexL}}\rangle\ (1,0)$. *Then*

0. *The order type of all total extensions of* $\succ_{pol}$ *is the same* (ω, ω^2, *or* ω^ω) *and is determined by trivial inspection of the leading coefficients of the interpretations, as detailed in Table 4.0.*

1. *If* $\succ_{pol}$ *has order type* ω, *then it is an extension of a tau pre-order and the value of tau is determined by examination of a simple ratio, as detailed in Table 4.0. If* $\succ_{pol}$ *has order type* ω^2, *then it is an extension of a lambda pre-order and the value of lambda is determined by trivial inspection of the leading coefficient of* $[\![\mathsf{f}]\!]$, *as detailed in Table 4.0. If* $\succ_{pol}$ *has order type* ω^ω, *then it is the recursive path ordering from the left defined by* $\mathsf{f} \rhd \mathsf{g}$.

2. *If one of the following holds:*

   - $m \geqslant n > 1$, $\frac{\ln a_m}{m-1} \neq \frac{\ln b_n}{n-1}$, *and* $m^p \neq n^q$ *for all* $p, q \in \mathbb{N}_+$,
   - $m > n = 1$, $b_1 > 1$ *and* $\frac{a_{m-1}}{m a_m} \neq \frac{b_0}{b_1 - 1}$, *or*
   - $m = n = 1$, $a_1 \geqslant 2$, $b_1 > 1$, $\frac{a_0}{a_1 - 1} \neq \frac{b_0}{b_1 - 1}$, *and* $a_1^p \neq b_1^q$ *for all* $p, q \in \mathbb{N}_+$,

   then $\succ_{pol}$ *is a total ordering, extending its tau or lambda pre-order with a lexicographic ordering, the precedence of which is determined by examination of a ratio as detailed in Table 4.0. If* $[\![\mathsf{f}]\!]$ *is linear and* $[\![\mathsf{g}]\!]$ *is strongly linear then* $\succ_{pol}$ *is a pre-order; the pre-order defined above.*

Hence the properties of $\succ_{pol}$ *can be fully determined from inspection of its leading parameters (subject to the three side conditions for lexicographic extensions above). From a computational point of view, we can replace* $\succ_{pol}$ *by an equivalent lexicographic combination of simpler orderings.*

The work of Martin and Scott told us what the order types of total simplification orderings on $\mathcal{T}(\{f, g\}, \{v\})$ could be, that the orderings of order type ω^ω are the recursive path orderings and the others contain certain pre-orders, and what term expressions can be compared to determine the pre-orders contained.

The work presented here has first applied those techniques to polynomial orderings on $\mathcal{T}(\{f, g\}, \{v\})$ and then shown how this provides a handle to determine much more detailed knowledge of the orderings. The polynomial orderings have been fully resolved into lexicographic combinations of simpler orderings, which are determined from the leading parameters of the orderings.

In the next section we will see how these results might extend to larger alphabets.

4.4 Three Unary Function Symbols

We begin extending the analysis to alphabets with more than two function symbols by first considering the case of three unary function symbols. From this study it will become clear how to generalise the results to arbitrary (finite) alphabets of unary function symbols.

Let $\mathcal{T}(\{f, g, h\}, \{v\})$ be the set of finite monadic terms over three unary function symbols, f, g, h, and one variable, v. Let $\succ$ be a polynomial ordering on $\mathcal{T}$ such that $f \succ g \succ h \succ v$. If we classify the polynomial interpretations $[\![f]\!](x)$, $[\![g]\!](x)$, and $[\![h]\!](x)$ according to whether each is non-linear, weakly linear, or strongly linear, then the order type of the defined polynomial ordering is given by Table 4.1.

These results are elaborated in the sequel by considering in turn each of the

NL	WL	SL	
		f, g, h	ω
	f	g, h	ω^2
f		g, h	ω^ω
	f, g	h	ω^2
f	g	h	ω^ω
f, g		h	ω^ω
	f, g, h		ω
f	g, h		ω^2
f, g	h		ω^2
f, g, h			ω

Table 4.1: Polynomial orderings on monadic terms over 3-letter alphabet.

three cases, ω, ω^2, and ω^ω.

4.4.0 Order Type ω

First consider the case where all function symbols have interpretations in the same class.

28 LEMMA

Let $\mathcal{T}(\mathcal{F}, \{v\})$ be the set of finite monadic terms over the finite non-empty set of unary function symbols $\mathcal{F} = \{f_1, f_2, \ldots, f_z\}$. Let $\succ$ be a total well-founded reduction ordering[4] *on $\mathcal{T}$ such that $f_1 \succ f_2 \succ \cdots \succ f_z$ and $f_z^j \succ f_1$ for some $j \in \mathbb{N}$. Then the ordering $\succ$ has order type ω.*

PROOF Since the set $\mathcal{T}$ is infinite, its order type under $\succ$ is at least ω. Assume that the conditions are satisfied but the ordering has order type greater than ω. Then there is (at least) one term, $t \in \mathcal{T}$, that has an infinite set of successors, $U = \{u \in \mathcal{T} \mid t \succ u\}$. Let $n \in \mathbb{N}$ be the number of function symbols in t, denoted by $|t|$. Since there are finitely many function symbols,

[4] i.e. a total simplification ordering

there are finitely many terms of any given length, so U has no finite bound on the length of its members, and in particular there is a term $u_0 \in U$ such that $|u_0| \geqslant jn$. Since $\succ$ is a simplification ordering, $f_1^n \succeq t$, and also $u_0 \succeq f_z^{jn}$. Therefore, $f_1^n \succeq t \succ u_0 \succeq f_z^{jn}$, which contradicts $f_z^j \succ f_1$. Hence the order type is ω. □

This lemma indicates why the polynomials have been partitioned into the non-linear, weakly linear, and strongly linear classes: these sets of polynomials are closed under composition and satisfy the above condition. It is thus clear that if a finite number of unary function symbols have interpretations in the same class then the polynomial ordering on monadic terms over these function symbols has order type ω.

4.4.1 Order Type ω^2

Next consider the case where the interpretations span two adjacent classes, first by examining the sub-case where exactly one interpretation is in the higher of the two classes.

29 LEMMA

Let $\mathcal{F} = \{f_1, f_2, \ldots, f_z\}$ be a finite set of unary function symbols such that $z \geqslant 2$. Let $\succ$ be a polynomial ordering on $\mathcal{T}$ such that the interpretation of f_1 is non-linear (resp. weakly linear) and all remaining interpretations are weakly linear (resp. strongly linear). Then the ordering $\succ$ has order type ω^2.

PROOF Partition $\mathcal{T}$ into classes $S_i = \{s \in \mathcal{T} \mid \#(f_1, s) = i\}$, $i \in \mathbb{N}$. Clearly for all $t' \in S_{i+1}$, $t \in S_i$, $t' \succ t$. We proceed by showing that each $(S_i, \succ)$ has order type ω.

Let $t = \alpha_0 f_1 \alpha_1 f_1 \ldots f_1 \alpha_k$ be an arbitrary member of S_k, where each $\alpha_i \in \mathcal{T}(\{f_2, \ldots, f_z\}, \mathcal{V})$. From Lemma 28 we know that for each $f \in \{f_2, \ldots, f_z\}$ there is an $N \in \mathbb{N}_+$ such that $f_z^N \succ f$. Let $s \in S_k$ be the term arrived at by

replacing each $f \in \{f_2, \ldots, f_z\}$ by the appropriate number (i.e. for each f the smallest sufficient N) of f_z's. Therefore, $s \succ t$ and $s \in \mathcal{T}(\{f_1, f_z\}, \mathcal{V})$.

In the proofs of Lemma 13 and Lemma 16 we saw that $(\mathcal{T}(\{f_1, f_z\}, \mathcal{V}), \succ)$ has order type ω^2 by showing that it contains a lambda pre-order; a pre-order that first orients by the number of f_1's in each term. Therefore as a corollary we know that $(S_k \cap \mathcal{T}(\{f_1, f_z\}, \mathcal{V}), \succ)$ has order type ω. From Lemma 28 we know also that $(\mathcal{T}(\{f_2, \ldots, f_z\}, \mathcal{V}), \succ)$ has order type ω. Let $U \subset S_k \cap \mathcal{T}(\{f_1, f_2, \ldots, f_z\}, \mathcal{V})$ be the set of successors of s, $\{u \mid s \succ u\}$, and assume that U is infinite. We will choose some $u \in U$, dependent on s. In u replace each $f \in \{f_2, \ldots, f_{z-1}\}$ by f_z. Since $\mathcal{F}$ is finite, there can be no upper bound on the length of members of (infinite) U, so we can choose a $u \in U$ such that after this replacement to $f_z^{u_0} f_1 f_z^{u_1} f_1 \ldots f_1 f_z^{u_k}$ its lambda expression

$$u_k \lambda^k + \cdots + u_1 \lambda + u + 0$$

is greater than the lambda expression of s. Therefore $u \succ_\lambda s$ and so $u \succ s$, which contradicts $s \succ u$, so U must be finite.

Since any term $t \in S_k \cap \mathcal{T}(\{f_1, f_2, \ldots, f_z\}, \mathcal{V})$ is smaller than some $s \in S_k \cap \mathcal{T}(\{f_1, f_z\}, \mathcal{V})$ and the term s has finitely many terms smaller than it, the class $(S_i, \succ)$ has order type ω. Hence $(\mathcal{T}, \succ)$ has order type ω^2. □

30 LEMMA

Let $\mathcal{F} = \{f_1, \ldots, f_y\} \cup \{f_{y+1}, \ldots, f_z\}$ be a finite set of unary function symbols and let $\succ$ be a polynomial ordering on $\mathcal{T}$ such that the interpretations lie in two adjacent classes of polynomials, $\{f_1, \ldots, f_y\}$ in the upper class and $\{f_{y+1}, \ldots, f_z\}$ in the lower class, with at least one interpretation in each of the two classes. Then the ordering $\succ$ has order type ω^2.

PROOF This proof is similar to the proof of Lemma 29, but here the symbols $f_1, \ldots, f_{y-1}$ are replaced by f_y. □

4.4.2 Order Type ω^ω

Finally, consider the case where the interpretations are not contained in two adjacent classes. This means that at least one interpretation is non-linear and at least one interpretation is strongly linear.

The case where all interpretations are either non-linear or strongly linear follows the pattern above.

31 LEMMA

Let $\mathcal{F}$ be a finite set of unary function symbols, and let $\succ$ be a polynomial ordering on $\mathcal{T}$ such that all interpretations are either non-linear or strongly linear, and each of these two classes contains at least one interpretation. Then the ordering $\succ$ has order type ω^ω.

PROOF The proof is by bounding substitutions, and is almost identical to the proofs of Lemmata 29 and 30. □

More interesting is the case where interpretations lie in all three classes, since this is where we might expect the highest order type. We conclude with the three-symbol case since the extension to more symbols will be obvious from the preceding lemmata.

32 CONJECTURE

Let $\mathcal{F} = \{\mathsf{f}, \mathsf{g}, \mathsf{h}\}$ be a set of unary function symbols, and let $\succ$ be a polynomial ordering on $\mathcal{T}$ such that the interpretations of $\mathsf{f}, \mathsf{g}, \mathsf{h}$ are non-linear, weakly linear, and strongly linear respectively. Then the ordering $\succ$ has order type ω^ω.

JUSTIFICATION We have

$$\begin{aligned}
[\![\mathsf{f}]\!](x) &= a_m x^m + \cdots + a_1 x + a_0, \\
[\![\mathsf{g}]\!](x) &= b_1 x + b_0, \\
[\![\mathsf{h}]\!](x) &= x + c
\end{aligned}$$

Partition $\mathcal{T}$ into classes $A_i = \{a \in \mathcal{T} \mid \#(\mathsf{f}, a) = i\}$, $i \in \mathbb{N}$. With reference to the polynomial interpretations, clearly for all $a' \in A_{i+1}$, $a \in A_i$, $a' \succ a$ and the classes A_i are ordered by an ω ordering. We would proceed by showing that each $(A_i, \succ)$ has order type ω^{i+2}.

Without loss of generality, consider a class A_i with $i \geqslant 1$. Partition A_i into classes $B_j = \{b \in A_i \mid b \sim_\lambda \mathsf{g}^j \mathsf{f} \alpha\}$, $j \in \mathbb{N}$, $\alpha \in A_{i-1}$, and λ is defined as before according to the interpretations of f and g. Again with reference to the polynomial interpretations, for all $b' \in B_{j+1}$ and all $b \in B_j$ we have $b' \succ b$, and the classes B_j are ordered by an ω ordering. A proof would proceed by showing that each $(B_j, \succ)$ has order type ω^{i+1}.

4.5 Further Developments

In this chapter we have fully analysed the polynomial orderings on a two-symbol unary alphabet, and we have analysed the order types of polynomial orderings on arbitrary unary alphabets. There are several directions in which this could develop.

- Polynomial orderings on monadic terms could be fully analysed so that the values of the invariants, and the determinants of any precedence orderings could be tabulated.
- The interpretations could be extended to include elementary interpretations.

- The analysis might be extended to include binary function symbols.

5 Conclusions

In this chapter the findings of the preceding chapters are summarised and future developments are discussed.

5.0 Summary

In this thesis we have tackled the problem of proving a term rewriting system terminating from two sides. First we studied two automatic techniques for finding semantic orderings, by linear programming and by population-based search. We saw that the method of complete description would be ideal for solving the problem of Knuth-Bendix termination if its redundant data could be precluded. A proposal for how this could be achieved was presented along with an idea of how the algorithm could be verified.

The search mechanism of genetic algorithms was applied to the search for a suitable polynomial ordering. Employing the interpretation heuristics of Steinbach and the constraint test of Giesl, the GA ordering search was found to be a promising approach to polynomial termination. The GA parameters that were found to be optimal were discussed, possibly shedding light on what makes an effective search for polynomial orderings.

Second we analysed the class of polynomial orderings defined on monadic terms. Using the order invariants of Scott and Martin, we saw how a seemingly complex class of orderings could be tamed and decomposed: by logical invariant, by numeric invariant, and finally by precedence. Once divided, the subclasses could be analysed to reveal a direct translation from defining parameters (polynomial coefficients) to resulting ordering.

5.1 Pipeline of Ordering Families

As discussed in Chapter 2, the Knuth-Bendix family of orderings is a natural choice for first strike on a termination problem. There is a full-decision procedure for KB termination, and there should be an efficient algorithm for the test. If the KB orderings fail, the user may well go to polynomial orderings to try. The question is, what useful information can be passed from the KB procedure to the polynomial procedure so that the second attempt doesn't waste time repeating the search of the first? Clearly it is a waste of time attempting all-strongly linear interpretations since those are a subfamily of the KB orderings, but there is probably more that can be passed on. If the KB orderings failed due to degeneracy in a rule inequality there is probably some way the polynomial ordering can capitalise on knowing what weights make the rule balanced. If polynomial orderings fail then there is also knowledge to be passed on, albeit more specialised. It may be possible to gain useful knowledge from a failed GA population, similar to detecting degeneracy in the KB ordering algorithm, by examining to what values the coefficients had settled and which inequalities they were failing to satisfy.

5.2 Extending the Orderings

Lescanne ([Les92]) has extended polynomial orderings to include exponential expressions. It would be interesting to see how the performance of the GA search mechanism is affected by introducing exponentials. There are already some preliminary results extending the analysis of Chapter 4 to these.

The genetic algorithm operators can be modified to accommodate permutations as individuals, so a second 'precedence search' GA could be incorporated with that of Chapter 3.

5.3 Test Bed

Steinbach has gathered and made available a large collection of termination problems. Dershowitz has also made an effort to collect examples from the literature, most of which are open problems.

When a new technique for termination is ready for testing or training, these examples are an invaluable resource. However, there is also a need for 'boring' examples. The cases that appear in the literature, being used as illustrative examples, for instance, by and large fall into two categories: those that appeared because they are interesting (and challenging) termination problems, and those that are not. It is difficult, for example, to find collections of examples that are terminating by polynomial ordering but which pose a computational challenge to the termination tool. The author would like to have a collection of examples that make an interesting domain for termination tools, even if the examples themselves are of little interest (to human readers). This would enable termination tools to be tested without having to tie them into a completion tool.

5.4 Complexity Links

There are several branches of interest in the proof-theoretic complexities associated with termination orderings and term rewriting (e.g. [Hof92, Cic90] and [FZ94]). The subject of Ferreira & Zantema's study is the order types of the underlying semantic domain, whereas here we have looked at the order types of the term algebra itself, so for example, Ferreira & Zantema classify all polynomial orderings in $\mathbb{N}$ as order-type ω. It may be fruitful to examine more closely how the two views relate, and whether one can feed the other.

Bibliography

[AHU58] Kenneth J Arrow, Leonid Hurwicz, and Hirofumi Uzawa. *Studies in Linear and Non-Linear Programming.* Stanford Univ Press, 1958.

[BBM93a] David Beasley, David R Bull, and Ralph R Martin. An overview of genetic algorithms: Part 1, fundamentals. *University Computing*, 15(2):58–69, 1993.

[BBM93b] David Beasley, David R Bull, and Ralph R Martin. An overview of genetic algorithms: Part 2, research topics. *University Computing*, 15(4):170–181, 1993.

[Bir35] G Birkhoff. On the structure of abstract algebras. *Proceedings of the Cambridge Philosophical Society*, 31:433–454, 1935.

[BL87a] Françoise Bellegarde and Pierre Lescanne. Transformation orderings. In *Proceedings 12th CAAP (TAPSOFT)*, pages 69–80, 1987.

[BL87b] Ahlem Ben Cherifa and Pierre Lescanne. Termination of rewriting systems by polynomial interpretations and its implementation. *Science of Computer Programming*, 9(2):137–159, 1987.

[BT95] Tobias Blickle and Lothar Thiele. A comparison of selection schemes used in genetic algorithms. TIK 11, Swiss Federal Institute of Technology, December 1995.

[Cic90] Adam Cichon. Bounds on derivation lengths from termination proofs. Tech report CSD-TR 622, University of London, Royal Holloway and Bedford New College, 1990.

[CJM+92] B A Carré, T J Jennings, F J Maclennan, P F Farrow, and J R Garnsworthy. SPARK – the SPADE Ada kernel. Technical report, Program Validation Ltd, 1992.

[CL92] Adam Cichon and Pierre Lescanne. Polynomial interpretations and the complexity of algorithms. In D Kapur, editor, *LNAI 607: 11th International Conference on Automated Deduction*, pages 139–147, 1992.

[Col75] George E Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *2nd GI Conference on Automata and Formal Languages*, pages 134–183, 1975.

[Cro92] Nick Cropper. Implementing the incremental Knuth-Bendix ordering algorithm. Project report, Computing Science Dept, Glasgow University, 1992.

[Cur89] I F Currie. Newspeak: A reliable programming language. In *High-Integrity Software*, Computer Systems, pages 122–158. Pitman, 1989.

[Dan63] George B Dantzig. *Linear Programming and Extensions.* Princeton Univ Press, 1963.

[Der79] Nachum Dershowitz. A note on simplification orderings. *Information Processing Letters*, 9(5):212–215, 1979.

[Der95] Nachum Dershowitz. 33 examples of termination. In *Proceedings Spring School*, volume 909 of *LNCS*, 1995.

[DJ90] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In J van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 6, pages 243–320. Elsevier Science Publisher BV, Amsterdam, 1990.

[DKM90] Jeremy Dick, John Kalmus, and Ursula Martin. Automating the knuth-bendix ordering. *Acta Informatica*, 28:95–119, December 1990.

[DM79] N Dershowitz and Z Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, August 1979.

[FZ94] Maria C F Ferreira and Hans Zantema. Syntactical analysis of total termination. Technical Report UU-CS-1994-28, Dept Computer Science, Utrecht University, 1994. Available in ftp://ftp.cs.ruu.nl/pub/RUU/CS/techreps/CS-1994/.

[Gar60] Walter W Garvin. *Introduction to Linear Programming.* McGraw-Hill, 1960.

[Gar83] Martin Gardner. Mathematical games. *Scientific American*, 249(2):8–13, August 1983.

[Gie95a] Jürgen Giesl. Generating polynomial orderings for termination proofs. In *6th International Conference on Rewriting Techniques and Applications*, Kaiserslautern, Germany, 1995.

[Gie95b] Jürgen Giesl. POLO – a system for termination proofs using polynomial orderings. Technical Report IBN 95/24, Technische Hochschule, Darmstadt, 1995.

[Gol89a] David E Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.

[Gol89b] David E Goldberg. Sizing populations for serial and parallel genetic algorithms. In *Proceedings Third International Conference on Genetic Algorithms*, 1989.

[Goo96] E Goodman. An introduction to galopps – the "genetic algorithm optimized for portability and parallelism" system. Technical Report 96-07-01, Michigan State University, September 1996. v3.2.

[Hof92] Dieter Hofbauer. *Termination Proofs and Derivation Lengths in Term Rewriting Systems.* PhD thesis, Technische Universität Berlin, 1992. Bericht 92–46.

[Hol75] John H Holland. *Adaptation in Natural and Artificial Systems.* University Michigan Press, 1975.

[JL82] Jean-Pierre Jouannaud and Pierre Lescanne. On multiset orderings. *Inf Proc Letters*, 15(2):57–63, September 1982.

[JL87] Jean-Pierre Jouannaud and Pierre Lescanne. Rewriting systems. *Technology and Science of Informatics*, 6(3):181–199, 1987.

[Kar84] N Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[KB67] Donald E Knuth and Peter B Bendix. Simple word problems in universal algebras. In J Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297, Oxford, England, August/September 1967. Pergamon Press.

[KL80] S Kamin and J-J Lévy. Attempts for generalising the recursive path orderings. Technical report, Dept Computer Science, Univ of Illinois, Urbana, Illinois, February 1980.

[Klo87] Jan Willem Klop. Term rewriting systems – a tutorial. *Bulletin of the European Association for Theoretical Computer Science*, 32:143–182, 1987.

[Kre68] Béla Krekó. *Linear Programming*, chapter 15, pages 263–267. Sir Isaac Pitman & Sons, 1968.

[Lan75] Dallas S Lankford. Canonical algebraic simplification in computational logic. Technical Report ATP-25, University of Austin, Texas, 1975.

[Lan79] Dallas S Lankford. On proving term rewriting systems are noetherian. Memo MTP-3, Louisiana Technical U., Dept. of Mathematics, Ruston (Louisiana), May 1979.

[Les81] Pierre Lescanne. Two implementations of the recursive path ordering on monadic terms. In *19th Allerton House Conference on Communications, Control and Computing*, pages 634–643, Urbana, 1981. University of Illinois Press.

[Les84] Pierre Lescanne. Uniform termination of term rewriting systems - the recursive decomposition ordering with status. In *Proceedings 9th CAAP*, pages 182–194. Cambridge University Press, 1984.

[Les86] Pierre Lescanne. Divergence of the Knuth-Bendix completion procedure and termination orderings. *EATCS*, 30:80–83, 1986.

[Les92] Pierre Lescanne. Termination of rewrite systems by elementary interpretations. In Hélène Kirchner and G Levi, editors,

Proceedings 3rd International Conference on Algebraic and Logic Programming, volume 632 of *LNCS*, pages 21–36, Volterra, Italy, September 1992.

[Mar87] Ursula Martin. How to choose the weights in the Knuth-Bendix ordering. In *Second International Conference on Rewriting Techniques and Applications*, volume 256 of *LNCS*, pages 42–53, May 1987.

[Mar93] Ursula Martin. On the diversity of orderings on strings. Technical report CS/93/13, St Andrews University, 1993. To appear in Fundamentae Informaticae.

[Mat93] Brian Matthews. MERILL: An equational reasoning system in Standard ML. In Claude Kirchner, editor, *Rewriting Techniques and Applications, 5th International Conference, RTA-93*, volume 690 of *LNCS*, pages 441–445, Montreal, Canada, 1993. Springer-Verlag.

[MN70] Zohar Manna and Stephen Ness. On the termination of Markov algorithms. In *Proceedings 3rd International Conference System Science*, pages 789–792, 1970.

[MRTT53] T S Motzkin, H Raiffa, G L Thompson, and R M Thrall. The double description method. In H W Kuhn and A W Tucker, editors, *Contributions to Theory of Games*, volume 2, pages 51–73. Princeton Univ Press, 1953.

[MS93] Ursula Martin and Elizabeth Scott. The order types of termination orderings on monadic terms, strings and multisets. In *Proceedings Eighth Annual IEEE Symposium on Logic in Computer*

Science, pages 356–363, Montreal, Canada, 1993. IEEE Computer Society Press.

[MZ94] Aart Middeldorp and Hans Zantema. Simple termination revisited. In Alan Bundy, editor, *Automated Deduction – CADE-12*, volume 814 of *LNAI*, pages 451–465, 1994.

[Pla78] David A Plaisted. A recursively defined ordering for proving termination of term rewriting systems. Technical report R-78-943, Dept of Computer Science, Univ of Illinois, 1978.

[Pla93] David A Plaisted. Term rewriting systems. In D M Gabbay, C J Hogger, and J A Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 4, chapter 2. Oxford Univ Press, 1993.

[SK93] Joachim Steinbach and Joachim Kühler. Check your ordering – termination proofs and open problems. SEKI SR-90-25, Universität Kaiserslautern, May 1993.

[Ste88] Joachim Steinbach. Comparison of simplification orderings. In U Martin et al, editor, *BCS-FACS Term Rewriting Workshop and Tutorial*, Bristol, England, September 1988.

[Ste91] Joachim Steinbach. Termination proofs of rewriting systems: Heuristics for generating polynomial orderings. SEKI SR-91-14, Universität Kaiserslautern, Germany, 1991.

[Ste92] Joachim Steinbach. Proving polynomials positive. In *Proceedings 12th Conference on Foundations of Software Technology and Theoretical Computer Science*, New Delhi, India, 1992.

[Ste94] Joachim Steinbach. *Termination of Rewriting.* PhD thesis, Fachbereich Informatik, Universität Kaiserslautern, 1994.

[Tar51] Alfred Tarski. *A Decision Method for Elementary Algebra and Geometry.* University of California Press, Berkeley, California, 1951.

[Tur95] D A Turner. Elementary strong functional programming. In *First International Symposium on Functional Programming Languages in Education*, volume 1022 of *LNCS*, pages 1–13, Nijmegen, Netherlands, December 1995.

[Uza58a] Hirofumi Uzawa. An elementary method for linear programming. In *Studies in Linear and Non-Linear Programming*, chapter 12, pages 179–188. Stanford Univ Press, 1958.

[Uza58b] Hirofumi Uzawa. A theorem on convex polyhedral cones. In *Studies in Linear and Non-Linear Programming*, chapter 2, pages 23–31. Stanford Univ Press, 1958.

[Wai93] Stan S Wainer. The logical complexity of proof transformations and program transformations. In Bauer et al, editor, *Proof and Computation*, NATO Summer School, Marktoberdorf, 1993.

[Whi93] L Darrell Whitley. *Foundations of Genetic Algorithms*, volume 2. Morgan Kaufmann, 1993.

[Wil65] R L Wilder. *Introduction to the Foundations of Mathematics.* J Wiley & Sons, New York, 1965.