LECTURE NOTES ON TERM REWRITING AND COMPUTATIONAL COMPLEXITY

by
Harvey M. Friedman
Ohio State University
friedman@math.ohio-state.edu
http://www.math.ohio-state.edu/~friedman/

November 7, 2001

Abstract. The main powerful method for establishing termination of term rewriting systems was discovered by Nachum Dershowitz through the introduction of certain natural well founded orderings (lexicographic path orderings). This leads to natural decision problems which may be of the highest computational complexity of any decidable problems appearing in a natural established computer science context.

1. TERM REWRITING.

A signature □ is a finite set of function symbols (arities ≥ 0). V is the set of variables $x_1,x_2,\ldots$. T(□,V) is the set of all terms using elements of □ □ V.

T(□) is the restriction to closed terms (i.e., with no variables).

A rewrite rule in T(□,V) is an expression

l □ r

where l,r □ T(□,V), l is not a variable, and every variable in r is a variable in l.

These two restrictions are from [BN], p. 61. Only the second restriction is important for us. We write

s □ t by l □ r

iff s,t □ T(□,V) and there is a substitution of variables in l by terms in T(□,V) which converts l to s and r to t.

A term rewriting system (trs) is a pair R = (R,□), where R is a finite set of rewrite rules in T(□,V). Term rewriting systems are implemented as follows.

We write s □ t by R iff s □ t by some l □ r in R.

An R-derivation is a nonempty sequence $t_1,t_2,\ldots$ of length 1 □ n □ such that for all 1 □ i □ n-1, t_i □ t_{i+1} by R.

THEOREM 1.1. Let R be a trs. All variables occurring in any R-derivation already occur in its first term. For all n ≥ 1 and s □ T(□,V), there are finitely many R-derivations from s of length □ n.

This depends heavily on the convention that, in each rewrite rule, all right variables are left variables.

2. ORDERED TERM REWRITING.

Ordered term rewriting is discussed in [BN], 267-268.

An ordered term rewriting system (otrs) is a triple (R,□,<), where (R,□) is a term rewriting system and < is a strict ordering on T(□,V).

An (R,□,<)-derivation is an R-derivation which is strictly decreasing under <.

Note that the presence of < only affects the allowable derivations. It does not have any affect on R, which can be any finite set of rewrite rules.

A well founded term rewriting system (wftrs) is an otrs whose < is well founded (no infinite strictly decreasing sequences).

As a consequence, every (R,<)-derivation is finite. In fact, more is true.

THEOREM 2.1. In any wftrs there are finitely many derivations starting with any given term.

Proof: Apply the fundamental fact that an infinite finitely branching tree has an infinite path. QED

3. TERMINATION FUNCTIONS. DERIVATION PROBLEMS.

N = the set of all nonnegative integers. Z^+ = the set of positive integers. The size of a term, #(t), is the total number of occurrences of functions and variables.

Let (R,□,<) be a wftrs. The termination function of (R,□,<) is TF(R,□,<):Z^+ □ Z^+ where TF(R,□,<)(n) is the longest length of an (R,□,<)-derivation that starts with a term of size at most n.

The derivation problem for (R,□,<) is to decide if there exists an (R,□,<)-derivation from a one given term to another.

THEOREM 3.1. Let (R,□,<) be a wftrs, where < is recursive. Then the termination function and the derivation problem are recursive.

Let < be a strict ordering on T(□,V). The termination function of < is TF(<):Z^+ □ Z^+ where TF(<)(n) is the longest length of a sequence t_0 > ... > t_r such that for all 0 □ i □ r, #(t_i) □ n+i.

THEOREM 3.2. Let < be a well founded strict ordering on T(□,V), where s > t implies every variable in t is a variable in s. Then TF(<) is everywhere defined.

NOTE: We have both termination functions of well founded term rewriting systems, and termination functions of well founded orderings of terms.

4. LEXICOGRAPHIC PATH ORDERINGS.

LPOs are discussed in [BN], 118-122.

Let □ be given, and let < be a strict ordering on □. We define <(□,V) as the unique strict ordering on T(□,V) satisfying the following condition.

Let s,t □ T(□,V). s >(□,V) t if and only if t □ V appears in s and s ≠ t, or s,t are not variables and the following holds.

Let s = f(s_1,...,s_m), t = g(t_1,...,t_n).

i) There exists i such that s_i ≥(□,V) t; or
ii) f > g and for all j, s ≥(□,V) t_j; or

iii) f = g and for all j, s >(□,V) t_j, and (s_1,…,s_m) >(□,V) (t_1,...,t_n) under the lexicographic extension of >(□,V) to T(□,V)^m = T(□,V)^n.

This is a recursive definition of a strict order. It has low computational complexity.

We write <(□) for the restriction of <(□,V) to T(□); i.e., to the closed terms.

These important orderings were introduced by Nachum Dershowitz in order to give a unified treatment of termination in term rewriting systems.

THEOREM 4.1. For all strict well founded orderings < on □, <(□,V) is a strict well founded ordering. <(□) is a well ordering iff < on □ is a linear ordering.

Dershowitz proved this using an important combinatorial theorem of J.B. Kruskal in [Kr60]. By far the simplest proof of this theorem is in [NW63]. Here is one of its many variants.

KRUSKAL’S TREE THEOREM. Let T_1,T_2,... be an infinite sequence of finite trees with left/right structure, where the vertices are labeled from a finite set. Then there is a label, inf, and structure preserving embedding from some tree into a later one.

The idea of the proof of Theorem 4.1 is this. If s >(□,V) t then every variable in t is present in s. So to prove well foundedness of <(□,V), we can pretend that V is finite.

Thus we can view the terms in T(□,V) as finite trees with left/right structure where the vertices are labeled from the finite set □ □ V. Suppose h is a label, inf, structure preserving embedding from term s into term t. Then s □(□,V) t, and hence □s >(□,V) t.

Kruskal’s tree theorem had earlier been proof theoretically analyzed in the 1980’s from the point of view of logic, and shown to be deeply connected with proof theory and fast growing functions. See [Fr01].

In fact, the ordinals of the lpo’s correspond exactly to the ordinals associated with Kruskal’s theorem.

Our work extends this correspondence to lpo term rewriting. Specifically, we establish the correspondence between lpo term rewriting and recursion on proof theoretic ordinals associated with Kruskal’s theorem.

Since s >(□,V) t implies every variable in t is a variable in s, we see that <(□,V) has an everywhere defined termination function.

The idea is that

i) the termination functions of lpo’s grow extremely fast, as fast as the growth rates for recursion on large proof theoretic ordinals;

ii) the termination functions of lpo term rewriting systems grow equally fast as termination functions of lpo’s;

iii) this fast growth makes its way into the computational complexity of derivation problems for the lpo term rewriting systems.

5. LPO TERM REWRITING THEOREMS.

In lpo term rewriting, we work with (R,□,<(□,V)) or (R,□,<(□)), where □ is a (finite) signature, R is a finite set of rewrite rules for T(□,V), < is a strict ordering on □, <(□,V) is the lpo on T(□,V) induced by <, and <(□) is the lpo on T(□) induced by <.

THEOREM 5.1. For all otrs (R,□,<(□,V)) there exists <(□*) such that □n ≥ 1, TF(R, <(□,V))(n) < TF(<(□*))(n).

THEOREM 5.2. □(□,<) there exists an otrs (R,□*, <*(□*)) such that □n ≥ 1, TF(<(□))(n) < TF(R,□*,<(□*))(n).

THEOREM 5.3. For all (□,<) there exists <(□*) such that the derivation problem for any otrs (R,□,<(□,V)) is in time complexity O(TF(<(□*)).

THEOREM 5.4. □(□,<) □ an otrs (R,□*,<(□*)) such that the derivation problem for (R,□*,<(□*)) is not in time complexity O(TF(<(□)).

We give an idea of the proof of Theorem 5.2, which is the heart of the matter. This shows how enormous integers come up in lpo term rewriting.

Let $t_0,\ldots,t_r$ be strictly descending in <(□), where each #(t_i) □ n+i and n ≥ 1. We want to find a derivation in an lpo trs that is just as long. The lpo trs must depend only on (□,<) and start with a term of size □ n.

It will be convenient to assume n ≥ 2, the case n = 1 being handled by a simple modification (in fact, simplification). Consider the following long sequence of closed terms.

□$_1$(n-2)*
□$_2$((n-1)*)
...
□$_{13}$((n+10)*)

J(□,G(n*,0,0,0,0),0,0)
...
J(□,G(0,0,0,0,0),t_0,n*)

J(t_0,G(n*,0,0,0,0),0,n*)
...
J(t_0,G(0,0,0,0,0),t_1,n*)

J(t_1,G((n+1)*,0,0,0,0),0,(n+1)*)
...
J(t_1,G(0,0,0,0,0),t_2,(n+1)*)

J(t_2,G((n+2)*,0,0,0,0),0,(n+2)*)
...
J(t_2,G(0,0,0,0,0),t_3,(n+2)*)

J(t_2,G((n+3)*,0,0,0,0),0,(n+3)*)

...

J(t_r,G((n+r)*,0,0,0,0),0,(n+r)*)
...
J(t_r,G(0,0,0,0,0),t_r,(n+r)*)

Here J,G,S,0,□$_1$,...,□$_{13}$,□ are new, and p* = S...S0, where there are p occurrences of S. These are, respectively, of arities 4,5,1,0,1,...,1. Take 0 < S and f < □ < □$_{13}$ < □$_{12}$ < ... < □$_1$ where f □ □ □ {0,S,G,J}.

Consider the following segment.

$\square_1$(n-2)*
$\square_2$((n-1)*)
...
$\square_{13}$((n+10)*)

Clearly this is strictly descending and starts with a term of complexity n. It is supported by the rules

$\square_i$(x) □ $\square_j$(S(x)).

Consider the step

$\square_{13}$((n+10)*)
J(□,G(n*,0,0,0,0),0,0)

This step is supported by the rule

$\square_{13}$(SSSSSSSSSS(x)) □ J(□,G(x,0,0,0),0,0).

Consider the segment

J(□,G(n*,0,0,0,0),0,0)
...
J(□,G(0,0,0,0,0),t_0,n*)

In the third argument, we go from 0 to t_0 by a buildup of terms procedure that is only exponential in #(t_0)= n. This raises the third argument, and so has to be compensated by lowering the second argument. However, the obvious lowering of the second argument takes only n steps, which is too quick. But it is easy to use the other arguments of G to greatly slow this down, as in primitive recursion.

Consider the segments

J(t_i,G((n+i)*,0,0,0,0),0,(n+i)*)
...
J(t_i,G(0,0,0,0,0),t_{i+1},(n+i)*)

This is handled in the same way. The rules used to support these segments have the form

J(x,G(_,_,_,_,_),_,y) □ J(x,G(_,_,_,_,_),_,y), where x,y are variables, the blanks inside G are appropriate terms driving a simple primitive recursion of suitably exponential length, and the blanks outside G are appropriate terms supporting the buildup of closed terms in original signature □.

Consider the following steps.

J(t_i,G(0,0,0,0,0),t_{i+1},(n+i)*)
J(t_{i+1},G((n+i+1)*,0,0,0,0),0,(n+i+1)*)

The step is supported by the rule

J(x,G(y,0,0,0,0),z,w) □ J(z,G(S(w),0,0,0,0),0,S(w)).

Finally, consider the following step.

J(□,G(0,0,0,0,0),t_0,n*)

J(t_0,G(n*,0,0,0,0),0,n*)

This step is supported by the rule

J(□,G(0,0,0,0,0),y,z) □ J(□,G(z,0,0,0,0),0,z).

6. ACKERMANN FUNCTION.

Let f:Z^+ □ Z^+ be strictly increasing. Let f':Z^+ □ Z^+ be given by f'(n) = f...f(1), where there are n f's. Define f_1(n) = 2n, f_{k+1} = f_k'. Write A(k,n) = f_k(n), and A(k) = A_k(k). A(k,n) has tremendous growth, even for small k. E.g.,

A(3,5) = $2^{65,536}$. A(4,3) = 65,536. A(4,4) = E*(65,536). And A(4,5) is E*(E*(65,536)).

Here E*(n) is an exponential stack of n 2's. A(5,5) is incomprehensibly large.

We can define A(k,n) as a double recursion, by A(k,n) = if k = 1 then 2n else if n = 1 then 2 else A(k-1,A(k,n-1)).

Note that values of A at only pairs lexicographically lower than (k,n) are called.

7. MULTIRECURSION.

We formally introduce multirecursion, which we prefer to do on □ = {0,1,...}.

A single step multirecursion is an equation

$f(x_1,\ldots,x_k) = t$,

where t is a term using the distinct variables $x_1,\ldots,x_k$, the k-ary function symbol $f_{<x1,\ldots,xk}$, the successor function S, the constant 0, and if then else.

The idea is that $f_{<x1,\ldots,xk}$ is interpreted as

$f_{<x1,\ldots,xk}(y_1,\ldots,y_k) =$ if $(y_1,\ldots,y_k) <_{lex} (x_1,\ldots,x_k)$ then $f(y_1,\ldots,y_k)$ else 0.

Obviously there is a unique solution f:$□^k$ □ □ to this equation.

A multirecursion is a nonempty finite sequence of equations

$f_1(x_1,\ldots,x_{k1}) = t_1$
$f_2(x_1,\ldots,x_{k2}) = t_2$
...
$f_r(x_1,\ldots,x_{kr}) = t_r$,

where each t_i uses at most $x_1,\ldots,x_{ki}$, $f_1,\ldots,f_{i-1}$, the successor function S, the constant 0, if then else, and $f_{i<x1,\ldots,x\_ki}$.

Here again $f_{i<x1,\ldots,x\_ki}$ is interpreted as

$f_{i<x1,\ldots,x\_ki}(y_1,\ldots,y_{ki}) =$ if $(y_1,\ldots,y_{ki}) <_{lex} (x_1,\ldots,x_{ki})$ then $f_i(y_1,\ldots,y_{ki})$ else 0.

Let (□,<) be a well ordering. We can define a multirecursion over (□,<) again as a nonempty finite sequence of equations

$f_1(x_1,\ldots,x_{k1}) = t_1$
$f_2(x_1,\ldots,x_{k2}) = t_2$
...
$f_r(x_1,\ldots,x_{kr}) = t_r$,

where each t_i uses at most $x_1,...,x_{ki},f_1,...,f_{i-1}$, the successor function S, the constant 0, if then else, <*, and $f_{i<*x1,...,x\_ki}$.

Here $f_{i<*x1,...,x\_ki}$ is interpreted as

$f_{i<x1,...,x\_ki}(y_1,...,y_{ki})$ = if $(y_1,...,y_{ki}) <*_{lex} (x_1,...,x_{ki})$ then $f_i(y_1,...,y_{ki})$ else 0,

where $<*_{lex}$ is the lexicographic product of k copies of <*.

This definition is appropriate in the case where <* is itself multirecursive.

Thus we speak of a multirecursive well ordering <*.

8. MULTIRECURSION AND TERMINATION FUNCTIONS.

Under reasonable hypotheses, multirecursion on initial segments of a well ordering and the termination function of its initial segments are intertwined.

9. UPWARDLY GENERATED ORDINALS.

Let □ be an ordinal. Let $f:\square^k$ □ □. We say that f is upwardly increasing if and only if

for all $\square_1,…,\square_k,\square_1,…,\square_k < \square$, if each $\square_i$ □ $\square_i$, then $\square_1,…,\square_k$ □ $f(\square_1,…,\square_k)$ □ $f(\square_1,…,\square_k)$.

We say that □ is upwardly generated iff there exist finitely many upwardly increasing functions $f_1,…,f_k$ from □ into □, of various arities ≥ 0, which generate □. Here we allow arity 0.

The sup of all upwardly generated ordinals is the important recursive ordinal □.

In fact, we can perform a more general construction. Let (D,<*) be a linear ordering. Let $f:D^k$ □ D.

We say that f is upwardly increasing iff for all $x_1,…,x_k,y_1,…,y_k$ □ D, if each x_i □ y_i, then $x_1,…,x_k$ □ $f(x_1,…,x_k)$ □ $f(y_1,…,y_k)$.

We say that (D,<*) is upwardly generated iff there exist finitely many upwardly in-creasing functions $f_1,…,f_k$ from D into D, of various arities ≥ 0, which generate D. Here we allow arity 0.

It can be proved that every upwardly generated (D,<*) is a well ordering, and their sup is the same ordinal □.

10. THE ORDINAL □.

Let n ≥ 1. We define $F_n:\square_1^n$ □ $\square_1$ as follows.

Let $\square_1,\ldots,\square_n < \square_1$.

Take $F_n(\square_1,\ldots,\square_n)$ to be the least □ such < $\square_1$ such that

i) □ is a power of □;
ii) for all $\square_1,\ldots,\square_n < \square$, if $(\square_1,\ldots,\square_n) <_{lex} (\square_1,\ldots,\square_n)$ then $F_n(\square_1,\ldots,\square_n) < \square$.

THEOREM 10.1. For n ≥ 1, let $\square_n$ be the least ordinal such that $F_n:\square_n$ □ $\square_n$. Then $\square_n$ is upwardly generated by F_n,0,+.

In fact, one can effectively compare terms in F_n,0,+.

The natural algorithm is of low computational complexity.

Any ordinal < $\square_n$ may be represented by different terms, but by exactly one normal term. The normal terms over F_n,0,+ are the terms generated by the clauses:

a) 0 is a normal term;
b) if k ≥ 2 and $t_1,\ldots,t_k$ are normal terms each of which start with F_n, and $t_1 \geq \ldots \geq t_k$, then $t_1 + \ldots + t_k$ is a normal term;
c) if $t_1,\ldots,t_n$ are normal terms then $F_n(t_1,\ldots,t_n)$ is a normal term.

One can effectively place any term in normal form (low complexity).

11. □ AND THE LEXICOGRAPHIC PATH ORDERINGS.

The lengths of the total lpo's are cofinal in □. There are nice embeddings of the total lpo's into the $\square_n$, and from the $\square_n$ into total lpo's. The embeddings are sufficiently good as to allow us to show that the termination functions of the total lpo's and the termination functions of the $\square_n$ are intertwined.

12. COMPLEXITY OF LPO TERM REWRITING.

We can now read off that lpo term rewriting corresponds quantitatively and complexity-wise to recursion on the ordinals $\square_n$.

13. PROOF THEORETIC FORMULATIONS.

$\square^1_2$ bar induction formulations. Provably recursive functions. Independence results.

REFERENCES

[BN] Franz Baader & Tobias Nipkow, Term Rewriting and All That, Cambridge Univ. Press, 1998.

[Fr01] Internal Finite Tree Embeddings, Reflection on the Foundations of Mathematics: Essays in honor of Solomon Feferman, ed. Sieg, Sommer, Talcott, Lecutre Notes in Logic, volume 15, Association for Symbolic Logic, 2001.

[Kr60] J.B. Kruskal, Well-quasi-ordering, the tree theorem, and Vazsonyi's conjecture, Trans. Amer. Math. Soc. 95 (1960), 210-225.

[NW63] C. St. J. A. Nash-Willliams, On well-quasi-ordering finite trees, Proc. Cambridge Phil. Soc. 59 (1963), 833-835.