Computer Science Department

TECHNICAL REPORT

Complexity of Term Rewriting

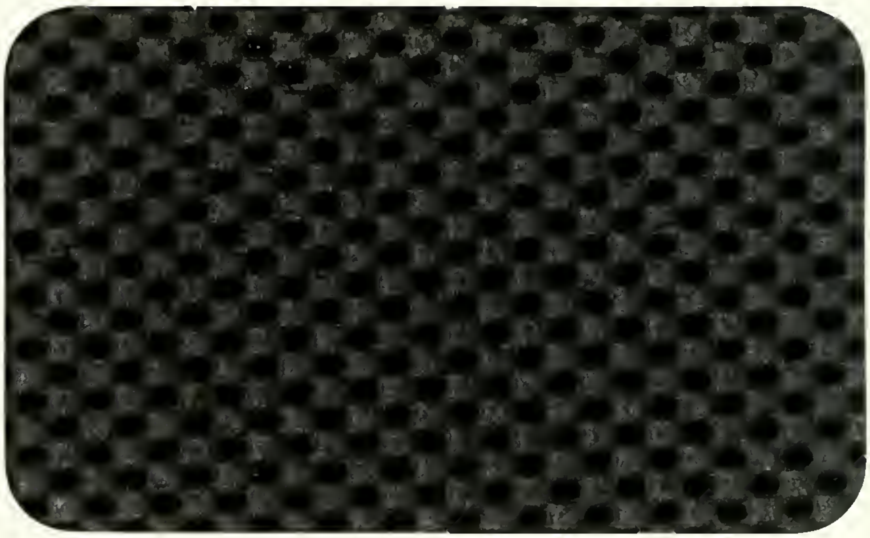
Ke Li

Technical Report 474

November 1989

NEW YORK UNIVERSITY

Department of Computer Science
Courant Institute of Mathematical Sciences
251 MERCER STREET, NEW YORK, N.Y. 10012



Complexity of Term Rewriting

Ke Li

Technical Report 474

November 1989

Complexity of Term Rewriting [1]

Ke Li
Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
251 Mercer Street
New York, NY 10012-1185
(212)998-3061
like@csd2.nyu.edu

Abstract

Term rewriting systems (TRSs) are efficient reduction systems. They find many applications in logic, languages, specifications, proof systems, etc. Rewriting a term to its normal form is a basic procedure. We show that the optimal normal form rewriting problem for terminating and confluent TRSs are NP-complete. If commutative and associative functions are allowed for a terminating and confluent TRS, the optimal normal form rewriting problem is also NP-complete.

[1]This research was supported in part by the Office of Naval Research under contract number N00014-85-K-0046 and by the National Science Foundation under grant number CCR-89-6949

Complexity of Term Rewriting

K. Li
Department of Comput[illegible]
Courant Institute of Math[illegible] Sciences
New York University
251 Mercer [illegible]
New York, NY 1001[illegible]
(212)998-3061
li[illegible]

Abstract

Term rewriting systems (TRSs) are [illegible] systems. They [illegible] many [illegible] and the theorem proof [illegible] rewriting a term to its normal form [illegible]. We show that the [illegible] normal form rewriting problem for terminating and confluent TRSs [illegible] commutative and associative functions [illegible] terminating and confluent TRSs the optimal [illegible] form rewriting problem is [illegible].

This research was supported [illegible]
N00014-85-K-0[illegible] and by the [illegible]

1 Introduction

The termination and confluence properties of term rewriting systems (TRS for short) have been studied thoroughly. To prove termination, many orderings, such as recursive path ordering, path orderings, etc., and polynomial interpretations can be used[Der82][Der87][KNS85]. For concrete applications of TRS, any simple and effective proof method for termination may be invented. Confluence was studied in deep in [Huet80] and the complete procedure to obtain a confluent system are introduced in [KB70][HO80][DJ88]. So far, we are able to design terminating and confluent TRS. A term rewriting system that has both termination and confluence properties is called a canonical TRS.

In a canonical TRS, any term has a unique normal form. This advantage of canonical TRS is used to solve the word problem in equational logic and other problems in reduction systems. Computing normal forms of terms is a basic procedure in any application of TRS. If we have some way to improve efficiency for normal form rewriting, efficiency for entire application will be improved.

Normal form rewriting have been studied by many researchers. Choppy et al. gave quantitative evaluation of normal form rewriting for a term by an algebraic analysis method[CKS87]. They defined the cost of terms and studied rewriting systems satisfying a special condition. Kapur et al. [KN87][BKN87] proved NP-hardness for many AC-matching problems, which are basic in application of TRS. Klop [Klop87] discussed strategies for regular TRS which guarantee that a term can be rewritten to its normal form. He said "for general TRSs there does not seem to be any result about the existence of 'good' reduction strategies".

In this report, we will answer why we may not find good strategies for general TRS by showing that the optimal normal form rewriting problem is NP-complete. The optimal normal form rewriting problem is formalized in this way: Given a canonical TRS, a term, and a derivation which rewrites the term to its normal form, we ask if there is a derivation which rewrites the term to its normal form and which has shorter length. With this result, the strategies which can find optimal derivations for term's normal form rewriting will not exist unless $P = NP$. The reason is that if a strategy can find optimal derivation during normal form rewriting for a term, it can answer the question that given a derivation for a term, if there exists a derivation with shorter length for that term.

If we allow some functions have commutative and associative properties in a TRS, we call the TRS an AC-TRS. In many applications of TRSs, the functions have commutative and associative properties, such as "+" (addition) and "−" (subtraction) in arithmetics, "$\wedge$" (and) and "$\vee$" (or) in logic. We prove that the optimal normal form rewriting problem for AC-TRS is NP-complete.

With these results, we unlikely find any very efficient strategies for TRSs and AC-TRSs. But this does not mean we could not do any thing in efficient normal form rewriting. We may find 'good' strategies for subclasses of TRSs. In another report, we propose several efficient strategies for subclasses of TRSs which can obtain optimal or approximate derivations during normal form rewriting for terms.

2 Preliminaries

Each function symbol f has a fixed arity which is the number of arguments of f. The functions with zero arity are called constants, denoted by $a, b, c, \ldots$. Variables are denoted by $x, y, z, \ldots$ or $\alpha, \beta, \gamma, \ldots$. Function symbols and variables are disjoint. A *term* is a constant or a variable or $f(t_1, t_2, \ldots, t_n)$ where f is a function symbol, the arity of f is n, and $t_1, t_2, \ldots, t_n$ are terms. Terms are denoted by $t, s, t', s', t_1, \ldots$. Variable-free terms are called *ground terms*. A *term rewriting system* is a set of rules and each rule is of form $t \rightarrow s$ where t and s are terms. As a convention, if a variable occurs at the right side of a rule, it must be occur at the left side of the rule. Generally, a term rewriting system is denoted by R and is abbreviated by TRS.

A *position* within a term is a sequence of positive integers, describing the path from the root function symbol to the head of the subterm at that position. For example, 2.2 is the position of y in term $g(a, f(x, y), z)$. By t/p, we denote the *subterm* of t at position p. If the subterm t/p of term t is replaced by term s, we denote the new term by $t[s]_p$. A *substitution* is a mapping from variables to terms. If σ is a substitution, σ can be extended to a function from terms to terms in such a way that $f(t_1, \ldots, t_n)\sigma = f(t_1\sigma, \ldots, t_n\sigma)$. Term t *matching* with term s means there is a substitution σ such that $t\sigma = s$. t *unifiable* with s means there is a substitution σ such that $t\sigma = s\sigma$. Term t *rewrites* to term s, denoted by $t \rightarrow s$, if there are position p in t, a substitution σ, and a rule $l \rightarrow r$ such that $t/p = l\sigma$ and $s = t[r\sigma]_p$. $t \xrightarrow{*} s$ means t rewrites to s by a number of steps. We say t *root-rewrites* to s if the left side of the rule $l \rightarrow r$ being applied to t matches with t itself other than a proper subterm of t. Given a TRS R, if term $t \xrightarrow{*} t'$ and t' cannot be rewritten further by rules in R, we say t' is *irreducible* and t' is a *normal form* of t.

A TRS is *terminating* if for any term t there is no infinite chain $t \rightarrow t_1 \rightarrow t_2 \rightarrow \cdots$. A TRS is *confluent*, if we have $t \xrightarrow{*} s_1$ and $t \xrightarrow{*} s_2$, then there exists a term u such that $s_1 \stackrel{*}{\to} u$ and $s_2 \stackrel{*}{\to} u$. It is easy to show that for any terminating and confluent TRS, a term has a unique normal form.

We say function f is *associative* if f satisfies $f(t_1, f(t_2, t_3)) = f(f(t_1, t_2), t_3)$ for any terms t_1, t_2, t_3, and function f is *commutative* if f satisfies $f(t_1, t_2) = f(t_2, t_1)$ for any terms t_1 and t_2. If f is both associative and commutative, we say f is AC or f has AC properties. In this report, we address only terminating and confluent TRSs, so we assume any TRS to be discussed is both terminating and confluent.

A sequence of rewritings which reduce a term to its normal form is called a *derivation* (for that term). $|D|$ is the length of D that is the number of rewritings in D.

3 NP-Completeness of Term Rewriting

Optimal Normal Form Rewriting Problem
INSTANCE: A terminating and confluent term rewriting system R, a term t, and a derivation D rewriting t to its normal form.
QUESTION: Is there any derivation D' such that D' rewrites t to its normal form and $|D'| < |D|$?

The size of the input in the optimal normal form rewriting problem is the number of symbols in the TRS, the term, and the derivation. Suppose we simultaneously try all derivations with lengths $< |D|$ to check if there exists the D' required in the problem. Since D is considered as one of inputs, each check is done in polynomial of the input size, so the normal form rewriting problem is in NP.

3-SAT Problem
INSTANCE: Collection $C = \{c_1, c_2, ..., c_m\}$ of classes on a finite set of variables $\{x_1, x_2, ..., x_n\}$ such that $|c_i| = 3$ for $1 \leq i \leq m$.
QUESTION: Is there a truth assignment for the variables that satisfies all the clauses in C?

3-SAT Problem is NP-complete, refer [GJ79]. In the following we construct a polynomial transformation from 3-SAT to the optimal normal form rewriting problem.

Given:
$V = \{x_1, x_2, ..., x_n\}$
$C = \{c_1, c_2, ..., c_m\}, \quad c_i = \{y_{i_1}, y_{i_2}, y_{i_3}\}, \quad y_{i_j} = x_k$ or $\bar{x}_k$

Construct:

Variables: $\alpha, \alpha_1, \alpha_2, \alpha_3, \beta, \beta_1, \beta_2, ..., \beta_{m+2}$

Function symbols: f of arity $m+2$ and g of arity 3

Constants:

$a_1^T, a_2^T, ..., a_n^T$	[true values]
$a_1^F, a_2^F, ..., a_n^F$	[false values]
$b_1^{pos}, b_2^{pos}, ..., b_n^{pos}$	[positive literal]
$b_1^{neg}, b_2^{neg}, ..., b_n^{neg}$	[negative literal]
S	start rewriting
S'	start ending part
T	true clause
F	false value
E	end of clause sequence
N	normal form
N_1,...,N_J	lead to N (J defined below)

Initial term t_{ini}: $f(S\ g(b\ b\ b)\ g(b\ b\ b)\ ...\ g(b\ b\ b)\ E)$
Each clause c_i corresponds to a subterm $g(b\ b\ b)$.
If x_j is in c_i, b_j^{pos} is an argument of the subterm.
If $\bar{x}_j$ is in c_i, b_j^{neg} is an argument of the subterm.

Positive integer: $J = n(m+1) + 2$

Derivation D: $t_{ini} \xrightarrow{r_6} N_1 \xrightarrow{r_7} N_2 \xrightarrow{r_7} \cdots \xrightarrow{r_7} N_J \xrightarrow{r_7} N$
(rules r_6, r_7 defined below)

Rules:

Form r_1:
$f(S\beta_1\beta_2...\beta_m E) \rightarrow f(a_1^T\beta_1\beta_2...\beta_m E)$
$f(S\beta_1\beta_2...\beta_m E) \rightarrow f(a_1^F\beta_1\beta_2...\beta_m E)$

Form r_2:
For i from 1 to n
For j from 2 to $m+1$ (for convenience, assume β_0 and β_{m+1} do not exist)
$f(\beta_1...\beta_{j-2}a_i^T g(b_i^{pos}\alpha_1\alpha_2)\beta_j...\beta_m E) \rightarrow f(\beta_1...\beta_{j-2}Ta_i^T\beta_j...\beta_m E)$
$f(\beta_1...\beta_{j-2}a_i^T g(b_i^{neg}\alpha_1\alpha_2)\beta_j...\beta_m E) \rightarrow f(\beta_1...\beta_{j-2}g(F\alpha_1\alpha_2)a_i^T\beta_j...\beta_m E)$
$f(\beta_1...\beta_{j-2}a_i^T g(\alpha_1 b_i^{pos}\alpha_2)\beta_j...\beta_m E) \rightarrow f(\beta_1...\beta_{j-2}Ta_i^T\beta_j...\beta_m E)$
$f(\beta_1...\beta_{j-2}a_i^T g(\alpha_1 b_i^{neg}\alpha_2)\beta_j...\beta_m E) \rightarrow f(\beta_1...\beta_{j-2}g(\alpha_1 F\alpha_2)a_i^T\beta_j...\beta_m E)$

$f(\beta_1 ... \beta_{j-2} a_i^T g(\alpha_1 \alpha_2 b_i^{pos}) \beta_j ... \beta_m E) \rightarrow f(\beta_1 ... \beta_{j-2} T a_i^T \beta_j ... \beta_m E)$
$f(\beta_1 ... \beta_{j-2} a_i^T g(\alpha_1 \alpha_2 b_i^{neg}) \beta_j ... \beta_m E) \rightarrow f(\beta_1 ... \beta_{j-2} g(\alpha_1 \alpha_2 F) a_i^T \beta_j ... \beta_m E)$
$f(\beta_1 ... \beta_{j-2} a_i^T g(\alpha_1 \alpha_2 \alpha_3) \beta_j ... \beta_m E) \rightarrow f(\beta_1 ... \beta_{j-2} g(\alpha_1 \alpha_2 \alpha_3) a_i^T \beta_j ... \beta_m E)$
$f(\beta_1 ... \beta_{j-2} a_i^T T \beta_j ... \beta_m E) \rightarrow f(\beta_1 ... \beta_{j-2} T a_i^T \beta_j ... \beta_m E)$

$f(\beta_1 ... \beta_{j-2} a_i^F g(b_i^{pos} \alpha_1 \alpha_2) \beta_j ... \beta_m E) \rightarrow f(\beta_1 ... \beta_{j-2} g(F \alpha_1 \alpha_2) a_i^F \beta_j ... \beta_m E)$
$f(\beta_1 ... \beta_{j-2} a_i^F g(b_i^{neg} \alpha_1 \alpha_2) \beta_j ... \beta_m E) \rightarrow f(\beta_1 ... \beta_{j-2} T a_i^F \beta_j ... \beta_m E)$
$f(\beta_1 ... \beta_{j-2} a_i^F g(\alpha_1 b_i^{pos} \alpha_2) \beta_j ... \beta_m E) \rightarrow f(\beta_1 ... \beta_{j-2} g(\alpha_1 F \alpha_2) a_i^F \beta_j ... \beta_m E)$
$f(\beta_1 ... \beta_{j-2} a_i^F g(\alpha_1 b_i^{neg} \alpha_2) \beta_j ... \beta_m E) \rightarrow f(\beta_1 ... \beta_{j-2} T a_i^F \beta_j ... \beta_m E)$
$f(\beta_1 ... \beta_{j-2} a_i^F g(\alpha_1 \alpha_2 b_i^{pos}) \beta_j ... \beta_m E) \rightarrow f(\beta_1 ... \beta_{j-2} g(\alpha_1 \alpha_2 F) a_i^F \beta_j ... \beta_m E)$
$f(\beta_1 ... \beta_{j-2} a_i^F g(\alpha_1 \alpha_2 b_i^{neg}) \beta_j ... \beta_m E) \rightarrow f(\beta_1 ... \beta_{j-2} T a_i^F \beta_j ... \beta_m E)$
$f(\beta_1 ... \beta_{j-2} a_i^F g(\alpha_1 \alpha_2 \alpha_3) \beta_j ... \beta_m E) \rightarrow f(\beta_1 ... \beta_{j-2} g(\alpha_1 \alpha_2 \alpha_3) a_i^F \beta_j ... \beta_m E)$
$f(\beta_1 ... \beta_{j-2} a_i^F T \beta_j ... \beta_m E) \rightarrow f(\beta_1 ... \beta_{j-2} T a_i^F \beta_j ... \beta_m E)$

Form r_3:
For i from 1 to $n-1$
$f(\beta_1 \beta_2 ... \beta_m a_i^T E) \rightarrow f(a_{i+1}^T \beta_1 \beta_2 ... \beta_m E)$
$f(\beta_1 \beta_2 ... \beta_m a_i^T E) \rightarrow f(a_{i+1}^F \beta_1 \beta_2 ... \beta_m E)$
$f(\beta_1 \beta_2 ... \beta_m a_i^F E) \rightarrow f(a_{i+1}^T \beta_1 \beta_2 ... \beta_m E)$
$f(\beta_1 \beta_2 ... \beta_m a_i^F E) \rightarrow f(a_{i+1}^F \beta_1 \beta_2 ... \beta_m E)$

Form r_4:
$f(\beta_1 \beta_2 ... \beta_m a_n^T E) \rightarrow f(S' \beta_1 \beta_2 ... \beta_m E)$
$f(\beta_1 \beta_2 ... \beta_m a_n^F E) \rightarrow f(S' \beta_1 \beta_2 ... \beta_m E)$

Form r_5:
$f(S' TT ... TE) \rightarrow N$

Form r_6:
$f(\beta_1 \beta_2 ... \beta_{m+2}) \rightarrow N_1$

Form r_7:
$N_1 \rightarrow N_2,\ N_2 \rightarrow N_3,\ \ldots\ ,\ N_{J-1} \rightarrow N_J,\ N_J \rightarrow N$

Intuition for the construction:

- a_i^T is the true value of variable x_i and a_i^F is the false value of variable x_i. b_i^{pos}

means positive literal x_i is in a clause and b_i^{neg} means negative literal $\bar{x}_i$ is in a clause.

- At the beginning, arbitrarily set a truth value of variable x_1, say a_1^T.
- Pass the value from the left of the sequence of the clauses (expressed as arguments of the term) to the right. For each clause, simulate the assignment of values in the clause. T means the clause is true, while F means one of values in the clause is false. If we pass value a_i^T or a_i^F but there is no literal x_i or $\bar{x}_i$ in the clause, just pass the clause.
- After all clauses have tried the truth value of x_i, arbitrarily set the truth value of x_{i+1}. And repeat passing the value.
- After all truth values of x_1, ..., x_n have been tried, we get $f(s_1...s_m a_n^T E)$ or $f(s_1...s_m a_n^F E)$ which will be rewritten to $f(S' s_1...s_m E)$. If all $s_i (1 \le i \le m)$ is T, that means we get an assignment which satisfies all clauses, then we obtain the normal form N immediately. Otherwise, the term will be rewritten to N via N_1, N_2,...,N_J.

For example, $(x_1 x_2 \bar{x}_3)(\bar{x}_2 x_3 \bar{x}_4)(x_4 x_5 x_6)$ is expressed by term:

$$f(Sg(b_1^{pos} b_2^{pos} b_3^{neg}) g(b_2^{neg} b_3^{pos} b_4^{neg}) g(b_4^{pos} b_5^{pos} b_6^{pos}) E)$$

Suppose x_2=true. We pass the value from left to right:

$f(a_2^T g(b_1^{pos} b_2^{pos} b_3^{neg}) g(b_2^{neg} b_3^{pos} b_4^{neg}) g(b_4^{pos} b_5^{pos} b_6^{pos}) E) \rightarrow$
$f(T a_2^T g(b_2^{neg} b_3^{pos} b_4^{neg}) g(b_4^{pos} b_5^{pos} b_6^{pos}) E) \rightarrow$
$f(Tg(F b_3^{pos} b_4^{neg}) a_2^T g(b_4^{pos} b_5^{pos} b_6^{pos}) E) \rightarrow$
$f(Tg(F b_3^{pos} b_4^{neg}) g(b_4^{pos} b_5^{pos} b_6^{pos}) a_2^T E)$

Denote the constructed term rewriting system by $R_{construct}$.

Theorem 1. *Let R be any term rewriting system which contains no rule $x \rightarrow t$ where x is a variable and t is a term. If any term can be root-rewritten only finite number of times and each application of any rule does not increase the size of the term being rewritten, R is terminating.*

Proof. Induction on the size of terms. Let t be any term.
Basis. $|t| = 1$. t must be a variable or a constant. Because no rule $x \rightarrow \cdots$ exists, a single variable is irreducible. Rewriting on a single constant is a special case of root-rewriting , so any constant can be rewritten only finite number of times.
Induction. $t = f(s_1...s_p)$, $|t| = k(> 1)$.
Induction hypothesis: Any term of size less than k can be rewritten only finite number of times.
Suppose there is an infinite reduction chain starting with t. The sizes of s_1, ..., s_p are

all less than k. By the induction hypothesis, they are rewritten only finite number of times, so there must be a root-rewriting in the infinite reduction chain. Suppose t is rewritten to t' just before the first root-rewriting and the first root-rewriting is $t' \rightarrow t'' = f(t_1 ... t_p)$. Because any rewriting does not increase the size of the term being rewritten, the sizes of $t_1, ..., t_p$ are all less than k. Hence, they can be rewritten only finite number of times. In the infinite reduction chain, we can find the second root-rewriting. Along this way, infinite number of root-rewritings will be found. A contradiction. $\square$

Lemma 1. *Only finite number of rules can be applied to a term of form $f(...)$ by root-rewriting.*

Proof. In this proof, a_i denotes either a_i^T or a_i^F. Suppose $t = f(s_1 ... s_{m+2})$.

r_7 cannot be applied to t by root-rewriting and application of r_5 and r_6 by root-rewriting will lead to N, so all we need to do is to prove that if we use r_1,r_2,r_3 and r_4 to root-rewrite t, they can be applied only finite times.

Two mappings h_1 : Term $\rightarrow$ Integer and h_2 : Term $\rightarrow$ Integer are defined below:
$h_1(a_i) = 1 \quad (1 \leq i \leq n)$
$h_1(s) = 0 \quad s$ is $f(...)$, $g(...)$, variable, constant other than $a_1, ..., a_n$
$h_2(S) = 0$
$h_2(a_i) = i \quad (1 \leq i \leq n)$
$h_2(S') = n + 1$
$h_2(s) = 0 \quad s$ is $f(...)$, $g(...)$, variable, constant other than $S, a_1, ..., a_n, S'$

Here are three mappings defined on $t = f(s_1, ..., s_{m+2})$:
$H_{binary}(t) = h_1(s_1)h_1(s_2)...h_1(s_{m+2})$ This is a binary number
$H_{sum}(t) = (n + 1)(m + 2) - (h_2(s_1) + h_2(s_2) + ... + h_2(s_{m+2}))$
$H(t) = 2^{m+2} \times H_{sum}(t) + H_{binary}(t)$

Suppose s is any term and s is rewritten to s'. It is easy to verify that $h_1(s) = h_1(s')$ and $h_2(s) = h_2(s')$, because only terms with root $f(...)$ and N_i $(1 \leq i \leq J)$ can be rewritten, those terms are rewritten to terms with root $f(...)$ and N_i $(1 \leq i \leq J)$, and function values of h_1 and h_2 for those terms are always zero. Therefore, if rewriting occurs in the strict subterms of $t = f(s_1 ... s_{m+2})$ and s_i is rewritten to s_i', we have

$$H(f(s_1 ... s_{m+2})) = H(f(s_1' ... s_{m+2}'))$$

That means, any rewriting of strict subterms of t does not change the H value of t. But we will show that any root-rewriting of t by r_1, r_2, r_3, or r_4 will decrease the H value.

Let $t = f(s_1 ... s_{m+2})$ be rewritten to $t' = f(s_1' ... s_{m+2}')$ by rule r by root-rewriting.

Case 1. r is r_2
$H_{sum}(t) = H_{sum}(t')$
$H_{binary}(t) > H_{binary}(t')$
Hence, $H(t) > H(t')$

Case 2. r is r_1, r_3 or r_4
$H_{sum}(t) = H_{sum}(t') + 1$
$-2^{m+2} < H_{binary}(t) - H_{binary}(t') < 2^{m+2}$
$H(t) - H(t') = 2^{m+2} \times 1 + (H_{binary}(t) - H_{binary}(t') > 0$
Hence, $H(t) > H(t')$.

Therefore, each application of r_1, r_2, r_3, and r_4 will decrease the function value of H. But the value of H is non-negative. We conclude that the term t can only be root-rewritten by r_1, r_2,r_3 and r_4 finite times. □

Lemma 2. *The term rewriting system $R_{construct}$ is terminating and confluent.*

Proof.
Terminating. In $R_{construct}$, there is no rule $x \rightarrow \cdots$. Only constant N_i $(1 \leq i \leq J)$ are reducible and they are all rewritten to normal form N. Any term of form $g(...)$ is not root-reducible. Any term of $f(...)$ can be root-rewritten only finite times by lemma 1. Hence, any term can be root-rewritten only finite times in $R_{construct}$. Application of rules in $R_{construct}$ does not increase the size of the term being rewritten. By theorem 1, $R_{construct}$ is terminating.

Confluent. Induction on the sizes of terms.
Basis. $|t| = 1$. t must be a variable or constant, which is either irreducible or has unique normal form N.
Induction. Suppose $|t| = k(> 1)$ and any term of size less than k has unique normal form.
Case 1. $t = f(s_1 ... s_{m+2})$. Let t be rewritten to normal form t'. If t' is not N, it must be of form $f(...)$ because any term of form $f(...)$ is rewritten to a term of form $f(...)$. t' is reducible by r_6. This is a contradiction, since t' is in normal form. Thus, t must be rewritten to N.
Case 2. $t = g(s_1 s_2 s_3)$. t cannot be root-rewritten and by induction hypothesis, s_1, s_2 and s_3 have unique normal forms, so t has unique normal form. □

Lemma 3. *There is an assignment of values to $\{x_1, x_2, ..., x_n\}$ which satisfies all the clauses if and only if there is a derivation D' such that D' rewrites the initial term t_{ini} to normal form N and $|D'| < |D|$.*

Proof.
Only if. There is a desirable variable assignment. Note that $|D| = J + 1$. Suppose the assignment is $x_i \rightarrow a_i (1 \leq i \leq n)$ where a_i denotes either a_i^T or a_i^F. Construct

D' as follows: By rule r_1, t_{ini} is rewritten to $f(a_1 g(bbb)...g(bbb)E)$. Then a_1 passes from the left to the right and simulates the values of the m clauses. By rule r_3, we get $f(a_2 g(bbb)...g(bbb)E)$. Repeat the passing process. At last we get $f(S'TT...TE)$ which will be rewritten to N. The total number of steps is $1 + n(m+1) + 1 = n(m+1) + 2 = J$. Hence, $|D'| < |D|$.

If. Note that in order to rewrite to N in $< |D| (= J+1)$ steps, rule r_6 cannot be used; otherwise, at least $J+1$ steps are needed. First the initial term t_{ini} must go through rule r_1 and then goes through r_2 and r_3. By r_1 and r_3, an assignment is selected, while by r_2, the values are passed to all clauses. To rewrite to N in $< J+1$ steps, the rewriting process must go through r_5. The left side is $f(S'TT...TE)$, that means all clauses are satisfied. □

Theorem 2. *The Optimal Normal Form Rewriting Problem is NP-complete.*

Proof. We have already known that the optimal normal form rewriting problem is in NP. By lemma 2, we have constructed a terminating and confluent term rewriting system from an instance of 3-SAT Problem. The theorem immediately follows lemma 3. □

4 Complexity of AC-Term Rewriting

AC Optimal Normal Form Rewriting Problem
INSTANCE: A terminating and confluent term rewriting system R in which some functions are AC, a term t, and a derivation D rewriting t to its normal form.
QUESTION: Is there any derivation D' such that D' rewrites t to its normal form and $|D'| < |D|$?

As in the last section, if we simultaneously try all derivations with lengths $< |D|$ to check if there exists the D' required in the problem, it is easily seen that this problem is in NP.

Ensemble Computation Problem (from [GJ79])
INSTANCE: A collection C of subsets of a finite set A and a positive integer J.
QUESTION: Is there a sequence

$$< z_1 = x_1 \cup y_1, z_2 = x_2 \cup y_2, ..., z_j = x_j \cup y_j >$$

of $j \leq J$ union operations, where each x_i and y_i is either $\{a\}$ for some $a \in A$ or z_k for some $k < i$, such that x_i and y_i are disjoint for $1 \leq i \leq j$ and such that for every subset $c \in C$ there is some z_i, $1 \leq i \leq j$, that is identical to c?

We can always find the desirable sequence in polynomial steps in the problem above, so we assume J is polynomial of the input size. By the proof on P.67 of Garey and Johnson's book, the subset can be restricted to exact three elements, that means, the following subproblem of Ensemble Computation Problem is also NP-complete.

3-Element Ensemble Computation Problem
INSTANCE: A collection C of three element subsets of a finite set A and a positive J.
QUESTION: the same as QUESTION of Ensemble Computation Problem.

Reduce 3-Element Ensemble Computation Problem to AC Optimal Normal Form Problem by polynomial transformation.

Note that if function f is associative, we can "flatten" the arguments of f. For example, $f(f(a,b),f(c,d)) = f(a,f(b,f(c,d)))$, the latter denoted by $f(a,b,c,d)$. Thus, we consider that the arities of associative functions are not fixed.

Given
$A = \{e_1, e_2, ..., e_n\}$
$C = \{c_1, c_2, ..., c_m\}, \quad c_i = \{e_{i,1}, e_{i,2}, e_{i,3}\}, \; e_{i,j} \in A \;\; (1 \le i \le m, \; 1 \le j \le 3)$
$J \ge 0$

Construct

AC function symbols:	f and g
Constants:	$e_1, e_2, ..., e_n, N, N_1, N_2, ..., N_K$ (K defined below)
Variable:	$\alpha, \alpha_1, \alpha_2, ..., \alpha_m, \beta, \beta_1, \beta_2, ..., \beta_m$
Initial term t_{ini}:	$t = f(g(e_{1,1}e_{1,2}e_{1,3})g(e_{2,1}e_{2,2}e_{2,3})...g(e_{m,1}e_{m,2}e_{m,3}))$
Positive integer:	$K = J - m + 1$
Derivation D:	$t_{ini} \overset{r_{m+2}}{\longrightarrow} N_1 \overset{r_{m+3}}{\longrightarrow} \cdots \overset{r_{m+3}}{\longrightarrow} N_K \overset{r_{m+3}}{\longrightarrow} N$

Rules:
$r_1 \quad f(g(e_ie_j\alpha)\beta_1\beta_2...\beta_{m-1}) \to f(N\beta_1\beta_2...\beta_{m-1}) \; (1 \le i < j \le n)$
$r_2 \quad f(g(e_ie_j\alpha_1)g(e_ie_j\alpha_2)\beta_1...\beta_{m-2}) \to f(NN\beta_1...\beta_{m-2}) \; (1 \le i < j \le n)$
...
$r_{m-1} \quad f(g(e_ie_j\alpha_1)...g(e_ie_j\alpha_{m-1})\beta) \to f(NN...N\beta) \; (1 \le i < j \le n)$
$r_m \quad f(g(e_ie_j\alpha_1)...g(e_ie_j\alpha_m)) \to f(NN...N) \; (1 \le i < j \le n)$
$r_{m+1} \quad f(NN...N) \to N$
$r_{m+2} \quad f(\beta_1\beta_2) \to N_1$
$r_{m+3} \quad N_1 \to N_2, N_2 \to N_3, ..., N_K \to N$

Explanation: (1) The construction of rules forces the function g to be associative and commutative, since without the properties $f(g(e_3e_2e_1)g(e_4e_2e_1))$ cannot be rewritten. (2) Because f is associative, any term of form $f(s_1...s_p)$ $(p \geq 2)$ can be rewritten to N_1 by r_{m+2}.

If a subset $c_i = \{e_{i_1}, e_{i_2}, e_{i_3}\}$, then c_i can be computed by two operations: $z_{j_1} = \{e_{i_1}\} \cup \{e_{i_2}\}$ and $z_{j_2} = z_{j_1} \cup \{e_{i_3}\}$. Therefore, $J \leq 2m$. There are polynomial number of rules. Obviously, each rule has polynomial number of symbols, so the reduction is a polynomial transformation. Define the size of a term to be the number of function symbols, variables and constants used in the term. The size of term t is denoted by $|t|$.

Lemma 4. *The term rewriting system constructed is terminating and confluent (in terms of AC).*

Proof.
Terminating. Application of $r_1, ..., r_m, r_{m+1}$, and r_{m+2} decreases the size of the term being rewritten, so these rules can be used only finite times. r_{m+3} reduces any N_i $(1 \leq i \leq K)$ to N and N is irreducible, so r_{m+3} can be used only finite times. Therefore, for any term, only finite rules can be used. That means, the system is terminating.

Confluent. Induction on the size of terms.
Basis. Terms of size 1 are variables or constants, which are either irreducible or written to normal form N by r_{m+3}. Hence, any term of size 1 has unique normal form.

Induction. Let the size of t be $k(> 1)$ and suppose any term of size less than k has unique normal form.
Case 1. $t = g(s_1...s_p)$. Because t cannot be root-written, the normal form of t is $g(s_1'...s_p')$, where s_i' $(1 \leq i \leq p)$ is a normal form of s_i. The sizes of $s_1, ..., s_p$ are less than k. By induction hypothesis, the normal form of s_i $(1 \leq i \leq p)$ is unique. Hence, t has unique normal form.
Case 2. $t = f(s_1...s_p)$. Let t be rewritten to normal form t'. If t' is not N, it must be a term of form $f(...)$ because any term of form $f(...)$ is rewritten to a term of form $f(...)$. t' is reducible by r_{m+2}. This is a contradiction, since t' is in normal form. Thus, t must be rewritten to N. □

Lemma 5. *There is a required sequence of length $j \leq J$*

$$< z_1 = x_1 \cup y_1, z_2 = x_2 \cup y_2, ..., z_j = x_j \cup y_j >$$

if and only if there is a derivation D' such that D' rewrites the initial term t_{ini} to normal form N and $|D'| < |D|$.

Proof.

Only if. Given $< z_1 = x_1 \cup y_1, z_2 = x_2 \cup y_2, ..., z_j = x_j \cup y_j >$. It is always possible to construct a sequence

$$\begin{aligned} < \quad & u_1 = \{e_{i_1,1}\} \cup \{e_{i_1,2}\}, ..., u_p = \{e_{i_p,1}\} \cup \{e_{i_p,2}\}, \\ & u_{p+1} = \{e_{i_{p+1},1}\} \cup u_{k_1}, ..., u_{p+m} = \{e_{i_{p+q},1}\} \cup u_{k_m} \quad > \end{aligned}$$

where $e_{i_l,1}, e_{i_l,2} (1 \leq l \leq p+m)$ are one of constants $e_1, ..., e_n$, $u_{k_l} (1 \leq l \leq m)$ is one of $u_1, ..., u_p$ and $p + m \leq j$, such that the sequence is a subsequence of the given sequence and satisfies the requirement of 3-Element Ensemble Computation Problem. The reason $p+m$ may be less than j is that there may be some redundant operations. For example, $z_{i_1} = z_{i_2} \cup z_{i_3}$ is a redundant operation, because any subset contains only three elements. We construct D' as follows: If u_l $(1 \leq l \leq p)$ is contained in w number of subsets $u_{p+v_1}, ..., u_{p+v_w}$, rule r_w can be used to simulate this operation and sets w arguments of f to N. Because the subset c_i corresponds to argument $g(...)$ of f one by one, eventually all arguments of f will be set to N. r_{m+1} rewrites $f(NN...N)$ to N. So, by $\leq p$ applications of rules of form $r_1, ..., r_m$, we get $f(NN...N)$ and by rule r_{m+1}, we get the normal form N. The number of rewriting steps in D' is $\leq p + 1 \leq j - m + 1 \leq J - m + 1 = K$, but $|D| = K + 1$.

If. The initial term t_{ini} is rewritten to N in $k \leq |D| (= K + 1)$ steps. Only rules of form $r_1, ..., r_m, r_{m+1}$ can be used, since r_{m+3} does not match with the term and application of r_{m+2} needs more than K steps to rewrite the term to N. Any application of rules of form $r_1, ..., r_m$ can be simulated by an operation $z_i = \{e_p\} \cup \{e_q\}$. After k steps, t_{ini} is rewritten to N. The last rule applied must be $f(NN...N) \to N$, that means after $k - 1$ steps t is rewritten to $f(NN...N)$ and two of three elements of each subset have been unioned. Add m operations $z_{p_1} = \{e_{p_2}\} \cup z_{p_3}$, we obtain the required sequence of length $k - 1 + m \leq K - 1 + m = (J - m + 1) - 1 + m = J$.

□

Theorem 3. *The AC Optimal Normal Form Rewriting Problem is NP-complete.*

Proof. We have already known that the AC optimal normal form rewriting problem is in NP. By lemma 4, a terminating and confluent term rewriting system with AC functions has been constructed from an instance of 3-Element Ensemble Computation Problem. The theorem immediately follows lemma 5. □

Acknowledgements

This research was done under the supervision of Prof. Zvi Kedem and Dr. Krishna Palem. Many thanks to their inspiration, reading, correcting, and guidance. Without them this work could not be done. I am also grateful to Prof. D. Kapur, who read the early version of this report and gave very helpful comments.

References

[BKN87] D.Benanav, D.Kapur and P. Narendran, "Complexity of matching problems", J. Symbolic Computation 3 (1987), 203-216.

[CKS87] C.Choppy, S.Kaplan, M.Soria, "Algorithmic complexity of term rewriting systems", 2nd Int. Conf. on Rewriting Techniques and Applications, 256-270, 1987.

[Der82] N. Dershowitz, "Orderings for term-rewriting systems", Theoretical Computer Science 17 (1982), 279-301.

[Der87] N. Dershowitz, "Termination of rewriting", J. Symbolic Computation(1987) 3, 69-116.

[DJ88] N. Dershowitz and J.-P. Jounnaud, "Rewriting systems", Draft, 1988.

[GJ79] M.R. Garey and D.S. Johnson, "Computers and intractability: A guide to the theory of NP-completeness", W.H. Freeman and Company, New York, 1979.

[HO80] G. Huet and D.C. Oppen, "Equations and rewrite rules: A survey", in Formal Languages: Perspective and Open Problems, R. Book(ed.), Academic Press, 1980.

[Huet80] G. Huet, "Confluent reductions: abstract properties and applications to term rewriting systems", JACM 27, 4 (1980), 797-821.

[KB70] D.E. Knuth and P.B. Bendix, "Simple word problems in universal algebras", in: (J. Leech, ed.) Computational Problems in Abstract Algebra, Pergamon Press, 1970, 263-297.

[KN87] D.Kapur and P.Narendran, "Matching, unification and complexity", SIGSAM Bull. 21, 4(Nov. 1988), 6-9.

[KNS85] D. Kapur, P. Narendran, and G. Sivakumar, "A path ordering for proving termination of term rewriting systems", CAAP'85, LNCS 185.

[Klop87] J.W. Klop, "Term rewriting systems: A tutorial", Bulletin of the European Asso. for Theoretical Comp. Sci. 32, 1987.

DATE DUE	BORROWER'S NAME

