IOSR Journal of Mathematics (IOSRJM)
ISSN: 2278-5728 Volume 1, Issue 5 (July-Aug 2012), PP 25-30
www.iosrjournals.org

Transformation Method: Making Termination Easier

D. Singh[1], A. M. Shuaibu[2] and A. M. Ibrahim[1]
[1]*Department of Mathematics, Ahmadu Bello University, Zaria-Nigeria*
[2]*Department of Mathematics, Statistics and Computer science, Kaduna Polytechnic, Kaduna-Nigeria*

Abstract: *We have critically described the various techniques for proving termination of term rewriting systems and have shown that the best of these methods is the transformation method. Transformation method implies that the termination of a given term rewriting system can be concluded from the termination of the transformed one, and proving termination of the transformed term rewriting system is often easier than proving termination of the given term rewriting system directly. The transformation method may be applied to prove termination of a term rewriting system where standard methods fail.*
1991 Mathematics Subject Classification: 68Q42, 68-02
Keywords: *Contractum, Non-erasing, Rewriting, Terms, Termination.*

I. Introduction

Term rewriting is a very powerful method for dealing with equations. Reasoning with equations involves deriving consequences of given equations and finding values for variables that satisfies such equations. In *term rewriting system (TRS), rewrite rules*, which are directed equations, are used to replace equals by equals, but only in the indicated direction until a *normal form* (an irreducible term) is obtained.

Two of the most important properties of term rewriting systems (TRSs) are *confluence (the Church-Rosser property)*, which implies that there can be at most one normal form for any term, and *termination (strong normalization)*, which implies the existence of at least one normal form. A confluent and terminating system is called *convergent (complete or canonical)* and defines exactly one normal form for each input term. Termination ensures that all computation paths end.

Developing techniques for proving termination of TRSs is a very challenging research area for a long time. Overviews of existing techniques for detecting termination of TRSs can be found in [1], [2] and [3].
In this work, we critically discussed the various techniques of termination proofs and then considered the transformation method with its merits over the standard methods. The transformation technique is easy to implement, and has the nice property that no explosion of the search space will be caused and therefore no heuristics is required. Furthermore, the technique is more powerful to be used in tools for proving termination of TRSs automatically.

In the next section, we will discuss the fundamentals of TRSs such as relations, terms, occurrences, contexts, substitutions and the concept of a rewrite rule. Whereas, in the remaining section, termination and techniques for proving termination of TRSs are presented; in particular, we explicate the transformation method and its superiority over the other methods of termination proofs.

II. Preliminaries

A TRS is a pair (Σ, R) of an alphabet (or signature) Σ and a set of reduction rules (rewrite rules) R. The alphabet Σ consists of:
(i) A countably infinite set Var of variables $x_1, x_2, x_3, \ldots$, also denoted $x, y, z, x', y', \ldots$
(ii) A non-empty set of function symbols or operators symbols $F, G, \ldots,$ each equipped with an *arity* $n \geq 0$, i.e., the number of *arguments* it is supposed to have. Thus, the function symbols may be *nullary* $(0-\text{ary})$, *unary, binary,* etc. $0-\text{ary}$ function symbols are sometimes called *constant symbols (or constants)*. A function symbol having an arbitrary number of arguments is called *varyadic*.

Terms are strings of symbols from a signature. The set of variables is assumed to be disjoint from the function symbols in the signature Σ i.e., $Var \cap \Sigma = \emptyset$.
The set of terms (or expressions) over Σ, denoted Ter (Σ), is defined recursively as follows:
(i) $x \in Ter(\Sigma)$ for every $x \in Var$. Variable terms may be called *atomic* terms.
(ii) If F is an $n-$ary function symbol $(n \geq 0)$ and $t_1, t_2, \ldots, t_n \in Ter(\Sigma)$, then $F(t_1, t_2, \ldots, t_n) \in Ter(\Sigma)$; these are called *compound* terms.
It follows from above that the constant symbols of Σ belongs to $Ter(\Sigma)$.We write c instead of $c(\,)$ for the case $n = 0$, for a constant symbol. In general, elements of $Ter(\Sigma)$ may be called *wellformed* terms.

The terms t_i are called the arguments of the term $F(t_1, t_2, . . . , t_n)$, and the symbol F, the *head* symbol, the *outermost* or *topmost* function symbol or *root*, denoted $F \equiv root(t)$.
For example, $f(a, g(x, y))$ is a term with the outermost function symbol f.

Terms which do not contain any variable are called *closed* (or *ground*) terms. The set of all closed terms is denoted $Ter_0(\Sigma)$.

Terms in which no variable occurs more than once are called *linear*; *non-linear* otherwise. A rule is *left- linear* when its left- hand side is linear and a TRS is left- linear if no rule contains repeated variables in its left-hand side. *Right-linear* rule and TRS are analogously defined.
The length of a term t, denoted $|t|$, is defined as the number of occurrences of function symbols and variables it contains. Thus, we have the following:
$|x| = |c| = 1$, for a variable x and a constant c,
$|F(t_1, t_2, . . . , t_n)| = |t_1| + |t_2| + . . . + |t_n| + 1$, for a function symbol F with arity $n \geq 1$.
Identity of terms is called *syntactic* identity and is denoted $\equiv$. Terese [2], noted that *terms* are the *objects* of TRSs.

2.1. Contexts, Occurrences and Substitutions

The notion of *contexts* and *occurrences* are introduced in order to abstract some further properties related to terms and operations on terms.
Informally, a *context* can be considered as an *incomplete* term which may contain *empty* places or *holes*. In other words, a term, with some of its subterms replaced by holes is called a *context*.
Formally, a context is a term containing zero, one or more occurrences of a special constant symbol □ , denoting an empty place (or hole). That is, a context is a term over the extended signature $\Sigma \cup \{ □ \}$.
If C is a context containing exactly n holes, and $t_1, t_2, . . . , t_n$ are terms, then $C[t_1, t_2, . . . , t_n]$ denotes the result of replacing the holes of C from left to right by $t_1, t_2, . . . , t_n$. If $t \in Ter(\Sigma)$ can be written as $t \equiv C[t_2, . . . , t_n]$, then the context C is also called a *prefix* of t. If $t \equiv D[C[t_2, . . . , t_n]]$ for some prefix D, then C is a subcontext of t, sometimes also called a *slice*.
In case, if there is exactly one occurrence of □ in C, then it is called a *one-hole context* denoted $C[\ \ ]$.
If $t \in Ter(\Sigma)$ can be written as $t \equiv C(s)$, then the term s is said to be a *subterm* of t denoted $s \leq t$.Since □ is itself a context, the trivial context, we also have $t \leq t$. Also, s is called a *proper* subterm of t, denoted $s < t$, if $s \leq t$ and $s \neq t$. In other words, the subterms s of t, other than t itself, are called *proper* subterms of t.
Two occurrences of subterms are called *disjoint* if neither is a subterm of the other.
Substitution is the operation of filling in terms for variables. That is, a substitution is a partial mapping from variables to terms (contexts) denoted $\{x_1 \longmapsto s_1, x_2 \longmapsto s_2, . . . , x_n \longmapsto s_n\}$, indicating that the variables x_i map to the terms s_i, $i = 1,2, \ldots, n$.
Following the definition in [2], a substitution is also a map $\sigma : Ter(\Sigma) \longrightarrow Ter(\Sigma)$ which satisfies

$$\sigma(F(t_1, . . . , t_n)) \equiv F(\sigma(t_1), . . . , \sigma(t_n))$$

for every $n-$ ary function symbol F $(n \geq 0)$.In particular, $\sigma(F) \equiv F$ if F is a constant symbol. We sometimes write t^σ (or $t\sigma$) instead of $\sigma(t)$.
The substitution σ is determined by its restriction to the set of variables. Therefore, it can equivalently be defined as a map $\sigma : Var \longrightarrow Ter(\Sigma)$.This approach is quite instructive and common (see [2], [3] and [4] for details).
Relatedly, a substitution σ that replaces distinct variables by distinct variables (i.e., σ is injective and $x\sigma$ is a variable for every x) is called a *renaming*. If $\sigma|_{Var(t)}$ is a renaming, then σ is said to be a renaming for t.
The two definitions of substitution given above are essentially equivalent and is sometimes indiscriminately denoted σ. Note that the term $t\sigma$, as a matter of fact, only depend on the values of σ for variables that actually occur in t.
It may be recalled that contexts are regarded as terms over an extended signature; the operation of substitution is likewise defined for contexts, by implication.
It is understood that a term t is called an *instance* of a term s iff there exists a substitution σ such that $\sigma(s) = t$. Note that $t[u]\sigma \equiv (t\sigma)[u\sigma]$. A term r is a (renamed) *variant* of s if they are instances of each other. If $t\sigma$ is a ground term, we call σ a *ground substitution* for t and $t\sigma$, a *ground instance* of t.

2.2. Rewrite Rules

A rewrite *(or reduction*) rule for a signature Σ is a pair (l, r) of terms of Ter (Σ), usually represented $l \longrightarrow r$. In fact, for convenience, a rewrite rule is often given a name using symbols $\rho_1, \rho_2, \rho_3, \ldots$. For example, we write $\rho : l \longrightarrow r$. The following two restrictions on rewrite rules are required to be observed:
(i) l, appearing on the left-hand side of $\longrightarrow$ is not a variable; and
(ii) every variable, occurring in the right-hand side (r) of $\longrightarrow$, must also occur in l.

A rewrite rule $\rho: l \longrightarrow r$ can be viewed as a *scheme*. An instance of ρ is obtained by applying a substitution σ . The result is an *atomic reduction step* $l\sigma \longrightarrow_\rho r\sigma$. The left-hand side $l\sigma$ is called a *redex* (from reducible expression), more precisely a ρ*-redex*. The right-hand side $r\sigma$ is called its *contractum.*
The rewrite rules can be applied in the direction of the arrow only in contrast with equations which can be applied in whatever direction they are able to reduce the terms in the given ordering.
A rewrite rule is called *non-erasing* if each variable that occurs in the left-hand side also occurs (at least once) in the right-hand side. Otherwise, it is called *erasing.* A *rewrite relation* is a binary relation over a set of terms Ter (Σ) that is closed under contexts and substitutions. A rewrite relation that is transitive and irreflexive is called a *reduction order* (sometimes, a *rewrite order*).

2.3. Well-founded monotone algebras

A considerable amount of efforts have also gone into the study of Well-founded monotone algebras and its applications in TRSs (see [2] and [5], in particular).
Definition

Let Σ be a signature, being a set of operation symbols each having a fixed arity. A $\Sigma-algebra\ (A, \Sigma_A)$ is defined to consist of a non-empty set A, and for every $f \in \Sigma$, a function $f_A: A^n \longrightarrow A$, where n is the arity of f. This function f_A is called the *interpretation of* f; we write $\Sigma_A = \{f_A | f \in \Sigma\}$.

A *well-founded monotone* $\Sigma-algebra$ $(A, \Sigma_A, <)$ is a $\Sigma-$algebra (A, Σ_A) equipped with a well- founded order $<$ on A such that each algebra operation is strictly monotone in every argument. More precisely, for every $f \in \Sigma$ and all $a_1, \ldots, a_n, b_1, \ldots, b_n \in A$ with some $a_i < b_i$ for some i and $a_j = b_j$ for all $j \neq i$ we have $f_A(a_1, \ldots, a_n) < f_A(b_1, \ldots, b_n)$.
Let $(A, \Sigma_A, <)$ be a well-founded monotone $\Sigma-$algebra and $\alpha: var \longrightarrow A$, then the term evaluation $[[.]]^\alpha: Ter(\Sigma) \longrightarrow A$ is defined recursively by

(i) $$[[x]]^\alpha = \alpha(x)$$
(ii) $\alpha\big(f(t_1, \ldots, t_n)\big) = f_A(\alpha(t_1), \ldots, \alpha(t_n))$ for $x \in var, f \in \Sigma,\ t_1, \ldots, t_n \in Ter(\Sigma)$. This function induces a strict partial order $<_A$ on Ter (Σ) as follows:

$$t <_A t' \Leftrightarrow \forall \alpha: var \longrightarrow A\ \alpha(t) < \alpha(t')$$

i.e., $t <_A t'$ means that for each interpretation of the variables in A, the interpreted value of t is smaller than that of t'.

In summary, a TRS is a binary relation over the set of terms of a given signature. The pairs of the relation are used for computing by replacements until an irreducible term is eventually reached. Hence, the absence of infinite sequences of replacements is called *termination.* A TRS is terminating if all rewrite sequences are finite. Rules of a terminating system are called *reduction or rewrite rules.*

III. Termination

Most properties of TRSs are undecidable. Given any TRS with finite signature and finitely many *reduction rules*, then it is undecidable whether confluence holds, and also whether termination holds (see [6] and [7]). Even for TRSs with only one rule, termination is undecidable [1]. However, it is observed in [4] that for *ground TRSs* (where all rules are between ground terms) confluence is decidable. Also termination is decidable for ground TRSs. For a particular TRS it may also be undecidable whether two terms are *convertible*, whether a term has a *normal form*, whether a term has an *infinite reduction.* Barendregt [8] observed that a TRS where all these properties of terms are undecidable is the *combinatory logic* (CL).

The key observation is that confluence and uniqueness of normal forms are the same property in the presence of termination, but confluence and uniqueness of infinite normal forms are not. Confluence implies uniqueness of infinite normal forms, but the other way round. Thus, we searched for and found a new property that resembles confluence and that is equivalent to uniqueness of infinite normal form. This property is called *skew confluence* [9].

Developing techniques for proving termination of TRSs is a challenging research area already for a long time. In recent years the emphasis is this area has shifted towards automation. It is no longer sufficient for new techniques to prove termination of particular TRSs in theory, but also tools should be able to use these techniques to prove termination automatically. Several tools have been developed for this goal, and there is a yearly competition in which all of these tools are applied to an extensive set of examples (see [10], for details).
To this end, we critically explicate the various methods of proving termination of TRSs and show the efficacy of the method of transformation over the basic techniques.

3.1. Methods of Termination

Termination is an important property of TRSs and some standard methods have been devised to prove termination. A standard method to prove the termination of a particular TRS consists in finding a well-founded order $>$ such that $l > r$ for each rewrite step $l \longrightarrow r$. Hence, by contraposition, as any infinite derivation would correspond to an infinite descending chain in the well-founded order, termination is reached. It turns out that it suffices to consider only those orders which are closed under contexts and substitutions. This eliminates the need to check reduction for all possible rewrite steps, but only for rewrite rules and is often used to prove termination taking the advantage of structure of terms [11].

Several methods for proving termination of term rewriting systems have been developed. Most of these methods are based on reduction orderings which are *well-founded*, *compatibl*e with the structure of terms and *stable* with respect to substitutions. All of these orderings are called *simplification* orderings, i.e., *a term is always greater than its proper subterms*.

Examples of these methods include Knuth-Bendix order, polynomial interpretations, multiset order, lexicographic path order, recursive decomposition order, semantic path order, transformation order, forward closures, semantic interpretations, dummy elimination and distribution elimination [12].

Techniques for proving termination of TRSs are generally classified into three classes; semantical methods, syntactical methods and transformation methods. We briefly discuss these three methods indicating the merits and shortcomings, in each case in the remaining parts of this section.

Method 1. Semantical Method

Proving termination by semantical method means that a *weight* function has to be defined in such a way that at every reduction step, the weight of a term strictly decreases. If the weight is a natural number or, more generally, a value in a set equipped with a well-founded order, then this cannot go on forever and thus termination is ensured. As a framework, this has been found very fundamental for proving termination. In fact, it has come to serve as the basis for a hierarchy of various kinds of termination.

Consider the following two simple examples of TRSs [2].

$$R_1\begin{cases} f(f(x)) \longrightarrow g(x) \\ g(g(x) \longrightarrow f(x) \end{cases}, \qquad R_2\begin{cases} f(g(x)) \longrightarrow f(h(f(x))) \\ g(g(x)) \longrightarrow f(g(h(x))) \end{cases}$$

It is obvious that R_1 is terminating, since at each reduction step the length of the term strictly decreases, and R_2 is also terminating since at each reduction step the number of gs strictly decreases.

Checking that indeed this kind of weight decreases at every reduction step; it suffices to check that for every rule the weight of the left-hand side is strictly greater than the weight of the right-hand side.

To this end, we will formalize this kind of termination proof and illustrate it with more examples.

A well-founded monotone algebra $(\mathrm{A},\ \Sigma_{\mathrm{A}}, <)$ is *compatible* with a TRS if $\mathrm{l} >_{\mathrm{A}} \mathrm{r}$ for every rule $\mathrm{l} \longrightarrow \mathrm{r}$ in the TRS. This leads us to a fundamental result due to Lankford of 1975 [13].

Theorem 1

A TRS is terminating if and only if it admits a compatible well-founded monotone algebra.

A way of proving termination of a TRS using semantical method is now as follows: (i) choose a set A equipped with a well-founded order $<$, (ii) define for each operation symbol f a corresponding f_A that is strictly monotone in all its arguments, and for which $\tau(l) > \tau(r)$ for all rewrite rules $l \longrightarrow r$ and all $\tau{:}\, var \longrightarrow A$. Then according to theorem 1, the TRS is terminating.

Example

The two simple examples shown earlier can be seen as applications of theorem 1 with the choice for $(A, <)$ as follows. The length of terms over f, g is modeled by choosing $f_A(x) = g_A(x) = x + 1$ for all $x \in A$. Indeed, f_A and g_A are both strictly monotone, and

$f_A(f_A(x)) = x + 2 > x + 1 = g_A(x)$ and $g_A(g_A(x)) = x + 2 > x + 1 = f_A(x)$, for all $x \in A$. Hence $f(f(x)) >_A g(x)$ and $g(g(x)) >_A f(x)$, proving termination of R_1 by theorem 1.

The number of gs in terms over f, g, h is modeled by choosing $f_A(x) = h_A(x) = x$ and $g_A(x) = x + 1$ for all $x \in A$. Clearly f_A, g_A and h_A are all strictly monotone, and

$$f_A(g_A(x)) = x + 1 > x = f_A(h_A(f_A(x)))$$

and $g_A(g_A(x)) = x + 2 > x + 1 = f_A(g_A(h_A(x)))$

for all $x \in A$, proving termination of R_2 by theorem 1.

This technique has some attendant difficulties in its application for how to choose suitable weight functions. For example, only some rough heuristics are available. The simplest useful choice for $(A, <)$ is $(\mathbb{N}^+, <)$, the set of strictly positive integers with the natural order. In many applications this is already a fruitful choice.

The main difficulty in showing termination by the semantical approach consists in finding an appropriate well-founded order.

This approach generalizes to termination modulo equations. However, it is observed in [1] that proving termination of rewriting modulo equations is, in practice, considerably more difficult than for plain rewrite systems. A TRS (Σ, R) is called *terminating modulo a set E of equations* over Σ if no infinite sequence of the following form exists:

$$t_1 \longrightarrow_R t_2 =_E t_3 \longrightarrow_R t_4 =_E t_5 \longrightarrow_R t_6 \ldots.$$

An important example of this approach is the method of polynomial interpretations [2].

Method 2. Syntactical Method

Syntactical methods are based upon orders on terms that are defined by induction on the structure of the terms. Given such an order, if it can be proved that it is well-founded, and if every reduction step causes a decrease with respect to the order, then termination is guaranteed. Taking the set of terms equipped with such an order makes this approach fit in the general framework of semantical methods. A well-known example of this type of order is the *recursive path order (RPO)*. To determine if *a term l is greater than a term r* in RPO, the outermost operators of the two terms are compared first. If the operators are equal, then those (immediate) subterms of r that are not also subterms of l must each be smaller (recursively in the term ordering) than some subterm of l. If the outermost operator of l is greater than that of r, then l must be greater than each subterm of r; while if the outermost operator of l is neither equal nor greater than that of r, then some subterm of l must be greater or equal to r.

Special examples of RPO are the multiset path order (MPO) and the lexicographic path order (LPO). Orders of this kind are usually *precedence-based*; they depend upon an order (called *precedence*) on the function symbols. We use a RPO to prove the termination of a TRS as follows:

Example

Consider the following system (for disjunctive normal form):

$$\begin{gathered} --a \longrightarrow a \\ -(a+b) \longrightarrow -a \times -b \\ -(a \times b) \longrightarrow -a + -b \\ a \times (b+c) \longrightarrow (\mathrm{a} \times b) + (a \times c) \\ (b+c) \times a \longrightarrow (b \times a) + (c \times a) \end{gathered}$$

for any terms a, b and c.

If the operators are ordered by $- > \times > +$, then we need only to show that

$$\begin{gathered} --a >_{rpo} a \\ -(a+b) >_{rpo} -a \times -b \\ -(a \times b) >_{rpo} -a + -b \\ a \times (b+c) >_{rpo} (a \times b) + (a \times c) \\ (b+c) \times a >_{rpo} (b \times a) + (c \times a), \end{gathered}$$

By the following line of reasoning:

$--a >_{rpo} a$ since $- > +$.

$-(a+b) >_{rpo} -a \times -b$, since $-(a+b) >_{rpo} -a$ and $-(a+b) >_{rpo} -b$ because the outermost operators are the same, but $\mathrm{a} + \mathrm{b} >_{rpo} a$ and $a+b >_{rpo}$ b. This is true by subterm condition. Thus, the second inequality holds. By similar argument, the third inequality also holds.

For the fourth inequality, since $\times > +$,

$a \times (b+c) >_{rpo} a \times b$ and $a \times (b+c) >_{rpo} a \times c$.

By the definition of the RPO for the case when two terms have the same outermost operator, $\{a, b+c\} > >_{rpo} \{a, b\}$ and $\{a, b+c\} >>_{rpo} \{a, c\}$. These two inequalities between multisets hold, since the element, $b+c$ is greater than both b and c with which it is replaced. By analogous argument, the fifth inequality also holds.

Hence, the system terminates for all inputs.

The disadvantage of this method is that it covers only a very restricted class of terminating systems. It is currently understood that generalization to termination modulo equations is hardly possible (see [11], for details).

It is decidable whether the termination of a finite TRS with a finite signature can be proved with RPO. Another advantage of this method is that it provides an algorithm that checks whether termination can be proved by this method or not. This algorithm is easy to implement.

Method 3. Transformation Method

In order to explicate the structure of transformation method, we first provide the following definition:

Definition 1.

Let Ψ be a transformation generated from a given TRS (Σ, R), a new TRS $\Psi(\Sigma, R)$ with $\Psi(\Sigma, R) = (\Psi(\Sigma), \Psi(R))$. Such a Ψ is called *nontermination preserving* if termination of (Σ, R) follows from termination of $\Psi(\Sigma, R)$. A transformation Ψ falls in this category if termination of a TRS (Σ, R) follows from termination of $\Psi(\Sigma, R)$.

Transformation methods provide *nontermination preserving transformations* between rewrite systems. In this way, developing nontermination preserving transformations on TRSs gives rise to a new class of transformation methods to prove termination of TRSs. Note that iteration of the aforesaid process may be required. For example, if termination of $\Psi(R)$ cannot be proved by some basic methods, one can go on trying to find another transformation Ψ' for which termination of $\Psi'(\Psi(R))$ is easily proved. If this succeeds for nontermination preserving transformations Ψ and Ψ', then termination of R can be concluded;otherwise it asks for further iteration.

Definition 2.

A recursive program scheme (RPS) is a TRS in which all left-hand sides of the rules have distinct roots symbols, and all of these left-hand sides are of the form $f(x_1,\ldots,x_n)$ where $x_1,\ldots,x_n$ are distinct variables.

Termination of finite RPS is easy to establish. Moreover, a finite RPS is terminating if and only if it is RPO-terminating.

Theorem 2**.**

Let R be any TRS and S be a terminating non-erasing RPS. Suppose $R_1 = \{l \longrightarrow r \in R \mid S(l) = S(r)\}$, $R_2 = \{S(l) \longrightarrow S(r) \mid l \longrightarrow r \in R \wedge S(l) \neq S(r)\}$. If both R_1 and R_2 are terminating TRSs, then R is terminating.

Example

Let the TRS R consist of the two rules

$$g(f(x)) \longrightarrow f(h(x)), \quad h(x) \longrightarrow g(x).$$

Termination of R can neither be proved by RPO nor Knuth-Bendix order. However, by choosing S to be the single rule $h(x) \longrightarrow g(x)$, we obtain

$R_1 = h(x) \longrightarrow g(x)$, $R_2 = \{g(f(x)) \longrightarrow f(g(x))\}$, which are both proved to be terminating by means of RPO with $g \succ f$. Termination of R follows by theorem 2.

The transformation method can be used to prove termination where basic methods such as RPO failed by nontermination preserving transformations. This technique has the nice property that no explosion of the search space will be caused and therefore no heuristics is required. Furthermore, the technique is easy to implement in proving termination automatically and easily generalizes to termination modulo equations.

IV. Conclusion

Some standard methods have been developed to prove termination for many TRSs. However, sometimes proving termination of a given TRS using these standard methods fails. In the literature many attempts have been made to strengthen these methods, mainly by refining path orderings. In this work we briefly discussed some standard methods of proving termination of TRSs with their defects and present another approach where if standard techniques like recursive path ordering fail to prove termination of a given TRS R, we do not try to find refinements of the order, but try to apply a non-termination preserving transformation ψ on R such that termination of $\psi(R)$ can be proved by means of transformation for which termination of the original TRS can be concluded from the termination of the transformed TRS.

We are not aware of TRSs for which termination of the transformed TRS is harder to prove than the termination of the original TRS, while the converse often occurs.

TRSs have various applications in some aspects of symbolic algebra, automated deduction, high-level programming languages, program verification and artificial intelligence.

References

[1] N. Dershowitz, Termination of Rewriting, *Journal of Symbolic Computation, 3(1 &2), 1987*, 69-115.

[2] Terese, *Term Rewriting Systems*, in M. Bezem, J. W. Klop and R. Vrijer (Eds.), Cambridge Tracts in Theoretical Computer Science, 55 (Cambridge University Press, 2003).

[3] J. W. Klop, *Term Rewriting Systems*, in S. Abramsky, D. Gabbay, and T. Maibaum,(Eds.), Handbook of Logic in Computer Science, 1(Oxford University Press, 1992) 1-112.

[4] G. Huet and D. S. Lankford, On the uniform halting problem for term rewriting systems. Rapport loboria 283, Institut de Recherche en Informatique et en Automatique, Le Chesnay, France, 1978.

[5] F. Baader and T. Nipkow, *Term Rewriting and All That* (Cambridge University Press, 1998).

[6] M. Dauchet, S. Tison, T. Heuillard, and P. Lescanne, Decidability of the confluence of ground term rewriting systems, *Proceedings of the 2nd Symposium on Logic in Computer Science*, New York, USA, 1987, 353-359.

[7] M. Oyamaguchi, The Church-Rosser property for ground term rewriting systems is decidable. *Theoretical Computer Science 49 (1)*,1987.

[8] H. P. Barendregt, The Lambda Calculus, its syntax and semantics, *Studies in Logic and the Foundations of Mathematics 103*, 2nd edition, Netherlands, 1984.

[9] S. Blom, *Term graph rewriting: syntax and semantics*, IPA dissertation series, no. 2001-05, 2001.

[10] C. Marché and H. Zantema, The Termination Competition,. in F. Baader (Ed.), *Proceedings of the 18th International Conference on Rewriting Techniques and Applications*, LNCS 4533, Springer Verlag, 2007, 303-313.

[11] N. Dershowitz, Ordering for Term Rewriting Systems, J. *Theoretical Computer Science, 17(3)*, 1982, 279-301.

[12] A. Koprowski, *Certification of Termination Proofs for Term Rewriting*, Radboud University Nijmegen, Foundations Group, intelligent Systems, ICIS, 2008.

[13] D. S. Lankford, *Canonical Algebraic Simplification in Computational Logic*, Memo ATP-25, Automatic Theorem Proving Project, University of Texas, USA, 1975.