ON PROVING UNIFORM TERMINATION AND RESTRICTED TERMINATION OF REWRITING SYSTEMS

J.V. GUTTAG,† D. KAPUR,‡ AND D.R. MUSSER‡

Abstract. In mechanical theorem proving, particularly in proving properties of algebraically specified data types, we frequently need a decision procedure for the theory of a given finite set of equations (axioms). A general approach to this problem is to try to derive from the axioms a set of rewrite rules that are "canonical," i.e., they rewrite to a canonical form all terms that are equal (according the axioms and the equivalence and substitution properties of equality). Rewrite rules are canonical if and only if they determine a relation that is both confluent and uniformly terminating. The difficulty of proving uniform termination has been the major drawback of the rewrite rule approach to deciding equations.

A new method of proving uniform termination is proposed. Assuming that the rewriting relation is globally finite (for any term there are only finitely many terms to which it can be rewritten), nontermination can occur only if there are cycles. Uniform termination is proved by showing that no cycles can occur. A method related to the Knuth and Bendix method of proving confluence is developed and used as the basis of such proof. In most cases, the proposed method will only prove termination for terms up to a certain size; this kind of "restricted termination" has a number of applications.

Key words. uniform termination, restricted termination, global finiteness, rewrite rules, confluence, Knuth-Bendix algorithm, overlap closure, canonical, rewrite dominoes, equational axioms, theorem proving

1. Introduction. Term rewriting systems, also called (sets of) rewrite rules, are a model of computation that has the interesting and useful property of being directly applicable to obtaining decision procedures for equational theories. A term rewriting system is said to be uniformly terminating if for every term, every sequence of rewrites starting from that term is of finite length. This property corresponds to the uniform halting property of Turing machines, a fact which Huet and Lankford (1977) used to demonstrate the undecidability of uniform termination.

In one of the main applications of term rewriting systems, the Knuth-Bendix (1970) approach to obtaining equational decision procedures, we start with a set of equations and attempt to derive from them a set of rewrite rules with the uniform termination property, as well as the property of confluence (for any term all sequences of rewrites emanating from it are extendable to a common term). If this can be done, the rules compute a canonical form for each class of terms that are equivalent under the original equations. Having such a canonical form yields an efficient decision procedure for the theory of the original set of equations.

Another application of rewrite rules is to produce "direct implementations" of abstract data types (Guttag, Horowitz, and Musser, 1978)). Such implementations are generally not efficient enough to be used in production programs, but can be helpful during the design of new data types and of programs that use the data types. Direct implementations are guaranteed to terminate if and only if the rewrite rules have the uniform termination property.

*Received by the editors December 18, 1981, and in revised form June 11, 1982. This paper was typeset at the General Electric Research and Development Center using the Text software developed for the Unix operating system.
†Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139.
1More commonly called "finitely terminating" or "Noetherian."
Although undecidable in general, the uniform termination property can be proved for particular term rewriting systems in a variety of ways, mostly based on the mapping of terms into a well founded partially ordered set (see Huet and Oppen (1980) for a survey). An unfortunate problem with attempting to prove uniform termination this way is that, for rule sets of any significant size, an appropriate mapping is often very difficult to construct — it may require a great deal of ingenuity. In this paper, we present a new approach to proving termination, which, though not as general as the previous methods, is more algorithmic. This method will usually not yield a complete proof of uniform termination by itself. However, it can be used for proving restricted termination (i.e., termination of a finite set of terms), which has applications as discussed later in the paper. The method can possibly be used to simplify the application of other methods of proving uniform termination.

When the original rules are not uniformly terminating, one would often like to be able to detect this situation quickly, e.g., in order to avoid wasting time attempting to construct a proof of uniform termination. Under some reasonable restrictions on the form of rewrite rules, our approach provides such a test. That is, we show that if the rules are globally finite (that is to say, the number of different terms to which any term can be rewritten is finite) and every rule is right-linear or every rule is left-linear, our method can be used to effectively search for cycles in the rewriting relation.

In an effort to make this paper self-contained, we devote § 2 to a short tutorial on term rewriting systems. The reader familiar with the literature in this area will need only to skim this section. We describe the basic mechanism of term rewriting systems, and state and prove a well known result relative canonicity to uniform termination and confluence. We also relate the property of global finiteness to uniform termination; in particular, we show that if a reduction relation is globally finite and acyclic, it is uniformly terminating.

In § 3 we address the problem of showing that a rewriting relation is globally finite. We first prove that global finiteness is undecidable, and then develop some syntactic conditions sufficient to ensure global finiteness.

In §§ 4 and 5, we address the problem of showing that a rewriting relation is acyclic. The main problem is the size of the space of terms that must be traversed in searching for cycles. Our main results show that the search space can be cut down significantly. We show how to construct, from a set of rules $R$, another set of rules we call the overlap closure of $R$, with the property that if a reflexive rule is contained in it, then the rewriting relation of $R$ has a cycle. The overlap closure corresponds to a subset of the transitive closure of the rewriting relation.

Our main theorem here states that a partial converse holds, so that generation of the overlap closure and looking for reflexive rules is sufficient to detect a cycle if one exists. In proving this theorem in § 5, we develop several lemmas of independent interest and a new way representing rewrite rules and sequences of rewrites using what we call rewrite dominoes and "rewrite domino layouts." We will introduce this representation and use it in presenting the proofs of our main results about the overlap closure. We believe that this representation also will be useful in the study of other areas of rewrite rule theory.

The generation of the overlap closure is very similar to the way the Knuth-Bendix process generates rules in attempting to produce a confluent rewriting relation. Its construction is based on the use of derived pairs of terms obtained from superpositions of the right hand side of one rule with the left hand side of another.
This is in contrast to the Knuth-Bendix process, which uses critical pairs obtained from superpositions of the left hand sides of the rules.

Like the Knuth-Bendix process, the overlap closure process may fail to terminate (that is, it may continue to generate new rules indefinitely). In fact, when the original rules are uniformly terminating, it will usually happen that overlap closure generation is nonterminating. In this case, the overlap closure process does not by itself yield a proof of uniform termination (see Huet and Oppen (1980)). As we show in § 6, apart from showing nontermination, partial generation of the overlap closure is useful in some applications where it suffices to have a proof of "restricted termination" — in this case termination for all "small" terms.

The overlap closure construction is more general than the forward chain construction discussed by Dershowitz (1981). As discussed in § 5, the overlap closure can be used in proofs of termination of both right-linear and left-linear rewriting systems, whereas forward chains require an additional strong assumption in the case of left-linear rewriting systems.

2. Definitions, notation, and basic theory.

2.1 Term rewriting systems. We begin with a definition of "terms." We assume a denumerably infinite set of distinguishable symbols called variable symbols, and a disjoint finite set of distinguishable symbols called function symbols. A term is defined inductively as either (1) a variable symbol, or (2) a function symbol followed by a finite sequence of terms. In the latter case, if \( f \) is the function symbol and \( t_1, \ldots, t_n \) is the sequence of terms, the term is denoted \( f(t_1, \ldots, t_n) \) and the \( t_i \) are called the arguments of the term. The number of arguments, \( n \), is called the arity (nullary, unary, binary, etc.) of the function symbol. A constant term is written as \( f() \) and binary terms will sometimes be written in infix notation, e.g., \((x+y) \cdot z\) for \(*((x,y),z)\).

The subterms of a term are the term itself and the subterms of its arguments. Like a variable, a term of the form \( f() \) has no subterms other than itself. A subterm position and corresponding subterm within a term is a finite sequence of nonnegative integers separated by "\( \)" and a related term determined as follows: to the null sequence (denoted \( <> \)) corresponds the entire term. If \( f(t_1, \ldots, t_n) \) is the subterm at position \( i \) then the subterm at position \( i,j \) is \( t_j \).

For example, the subterm positions and corresponding subterms within \( f(x,g(y,k(z)),h()) \) are:

- \( <> \)  \( f(x,g(y,k(z)),h()) \)
- \( 1 \)  \( x \)
- \( 2 \)  \( g(y,k(z)) \)
- \( 2.1 \)  \( y \)
- \( 2.2 \)  \( k(z) \)
- \( 2.2.1 \)  \( z \)
- \( 3 \)  \( h() \)

We write \( t[i] \) for the subterm at position \( i \) within term \( t \).

A rewrite rule is an ordered pair of terms \((l,r)\) such that every variable that occurs in \( r \) occurs also in \( l \). We usually denote the rule as \( l \rightarrow r \). A term rewriting system is a set (usually finite) of such rules. This is all there is to the syntax of term rewriting systems; they also have a very simple semantics, to which we now turn.

Two terms are identical if (1) they are identical variables, or (2) they have identical function symbols and their sequences of arguments are identical. We write \( t_1 \equiv t_2 \) for this relation.
A substitution is a mapping $\theta$ from variable names to terms such that $\theta(v) = v$ for all but a finite number of variable symbols. It is denoted by an expression of the form $[t_1/v_1, \ldots, t_n/v_k]$, where the $k \geq 0$ variable symbols $v_1, \ldots, v_k$ are distinct. (The case $k = 0$ is the identity substitution.) The domain of a substitution $\theta$ is extended to the set of all terms by inductively defining $\theta(f(t_1, \ldots, t_n))$ to be $f(\theta(t_1), \ldots, \theta(t_n))$.

A set of rewrite rules, $R$, generates a binary relation $\rightarrow$ on the set of all terms, called the rewriting relation, as follows:
1. For every rule $(l, r)$ in $R$, and every substitution $\theta$, $\theta(l) \rightarrow \theta(r)$ holds.
2. If $t \rightarrow u$, then for every function symbol $f$ and sequence of terms $t_1, \ldots, t_i, \ldots, t_n$ with $t_i = t$ for some $i$, $f(t_1, \ldots, t_i, \ldots, t_n) \rightarrow f(t_1, \ldots, u, \ldots, t_n)$ also holds.

Whenever $t \rightarrow u$, we say that $t$ rewrites directly to $u$ (using $R$). An equivalent way of defining the direct rewriting relation is as follows:

We say that $t_2$ has the form of $t_1$ if there is a substitution $\theta$ such that $\theta(t_1) = t_2$; also we say that $t_2$ is matched by $t_1$. Then $t$ rewrites directly to $u$ (using $R$) if and only if there is a subterm position $i$ such that $t[i]$ is matched by the left-hand side $l$ of some rule $l \rightarrow r$ of $R$, say $\theta(l) = t[i]$, and $u$ can be obtained from $t$ by replacing $t[i]$ by $\theta(r)$ at position $i$. We write this as $[t$ with $\theta(r)$ at $i].$

2.2. Reduction relations. As in Huet (1980), we now develop a number of concepts at a more abstract level than term rewriting systems: we assume a set $E$ and a binary relation $\rightarrow$ on $E$, called "reduction." When definitions or lemmas depend on the set $E$ being a set of terms and the reduction relation being the rewriting relation generated by a set of rewrite rules, we make explicit note of this fact.

The reflexive and irreflexive transitive closures of $\rightarrow$ are denoted $\rightarrow^*$ and $\rightarrow^+$, respectively. Thus $A \rightarrow^* B$ if and only if there are elements $A_0, \ldots, A_n$, $n \geq 0$, such that $A = A_0$, $A_n = B$ and $A_0 \rightarrow A_1 \rightarrow \cdots \rightarrow A_n$; for $A \rightarrow^+ B$ we require $n \geq 1$. The sequence $A_0 \rightarrow \cdots \rightarrow A_n$ is called a reduction sequence from $A_0$ to $A_n$.

Let $\equiv$ be the relation defined by $A \equiv B$ if and only if $A \rightarrow B$ or $B \rightarrow A$. The reflexive transitive closure of $\equiv$, denoted $\equiv^*$, is the abstract version of "equality" (it is an equivalence relation on $E$; and in the case of a term rewriting system, it is moreover an equality relation since the substitution property holds). In order to deal with $\equiv^*$ in terms of $\rightarrow^*$, we will define and use canonical forms in $E$.

First, we define an element $P$ of $E$ to be terminal if there is no element $Q$ such that $P \rightarrow Q$. If $A \rightarrow^* B$ and $B$ is terminal, then we call $B$ a terminal form of $A$. An element can have many distinct terminal forms. A nonterminating reduction sequence from $A$ is an infinite sequence $A_0 \rightarrow A_1 \rightarrow \cdots$ with $A_0 = A$. The reduction relation $\rightarrow$ is said to be terminating for $A$ if there is no nonterminating reduction sequence from $A$, and $\rightarrow$ is said to be uniformly terminating if it is terminating for every element of $E$.

We say any $B$ and $C$ are joinable if there exists a $D$ such that $B \rightarrow^* D$ and $C \rightarrow^* D$. The relation $\rightarrow$ is confluent from $A$ if for every $B$ and $C$ such that $A \rightarrow^* B$ and $A \rightarrow^* C$, $B$ and $C$ are joinable. We say $\rightarrow$ is uniformly confluent if it is confluent from every element of $E$.

**Lemma 2.2.1** (well known). If $\rightarrow$ is uniformly confluent, the terminal form of any element, if it exists, is unique.

**Proof.** Let $A \rightarrow^* B$ and $A \rightarrow^* C$, with both $B$ and $C$ terminal. By confluence from $A$, there must be a $D$ such that $B \rightarrow^* D$ and $C \rightarrow^* D$, but since $B$ and $C$ are terminal, we must have $B = D = C$. ☐
Let $\equiv$ be an equivalence relation on $E$. We say that $\rightarrow$ is canonical with respect to $\equiv$ if it is uniformly terminating and any two elements have the same terminal form if and only if they belong to the same $\equiv$ equivalence class. If $\rightarrow$ is canonical with respect to $\rightarrow^*$, we simply say that it is canonical.

**Theorem 2.2.2** (well known). A reduction relation $\rightarrow$ is canonical if and only if it is both uniformly terminating and uniformly confluent.

**Proof.** First, suppose $\rightarrow$ is canonical; then by definition it is uniformly terminating; we have to show that it is uniformly confluent. Let $A$ be any element and suppose $A \rightarrow^* B$ and $A \rightarrow^* C$; then $B \rightarrow^* C$ and, since $\rightarrow$ is canonical, $B$ and $C$ have the same terminal form. Thus $\rightarrow$ is confluent from $A$, for all $A$.

In the other direction, suppose $\rightarrow$ is uniformly terminating and uniformly confluent. Let $A$ and $B$ be any elements such that $A \rightarrow^* B$; we have to show that $A$ and $B$ have the same terminal form. By definition of $\rightarrow^*$, there are elements $A_0, \ldots, A_n, n \geq 0$, such that $A = A_0, A_n = B$ and for $0 \leq i \leq n-1$ either $A_i \rightarrow A_{i+1}$ or $A_{i+1} \rightarrow A_i$. The proof is by induction on $n$. If $n = 0$, then $A = B$ and, by confluence and Lemma 2.2.1, they have the same terminal form. If $n > 0$, then by the induction hypothesis, $A_0$ and $A_{n-1}$ have the same terminal form, $T$, say. If $A_{n-1} \rightarrow A_n$, then by confluence from $A_{n-1}$, $T$ is also the terminal form of $A_n$. The other case is $A_n \rightarrow A_{n-1}$; by confluence from $A_n$ and Lemma 2.2.1, $T$ is also the terminal form of $A_n$. □

In the case of a term rewriting relation, Knuth and Bendix (1970) showed how to perform the test for uniform confluence by examining how the left-hand sides of the rules “overlap” each other, producing “critical pairs” of terms to be checked for confluence. Out approach to proving uniform termination relies on a generalized notion of overlapping, yielding “derived pairs.” This will be discussed in § 4.1.

### 2.3. Relating uniform termination to global finiteness and acyclicity.

A reduction relation $\rightarrow$ is locally finite if for every $A$ in $E$, the set of elements $B$ such that $A \rightarrow B$ is finite; and $\rightarrow$ is globally finite if the set of $B$ such that $A \rightarrow^* B$ is finite.

**Lemma 2.3.1.** If a reduction relation is globally finite and acyclic, it is uniformly terminating.

**Proof.** Any nonterminating reduction sequence would either have to repeat some element (hence be a cycle) or contain infinitely many distinct elements (hence be globally infinite). Thus all reduction sequences have to terminate. □

The converse does not hold in general but does hold for locally finite reduction relations.

**Theorem 2.3.2.** A locally finite reduction relation is uniformly terminating if and only if it is both globally finite and acyclic.

**Proof.** The if part is immediate from the lemma. Suppose the relation is not globally finite or is cyclic. In the latter case, it is obviously not uniformly terminating. In the former case, since the relation is assumed to be locally finite, we can apply Koenig's lemma to some element which has infinitely many descendents, concluding that there must be an infinite path of reductions from that element; thus the relation is not uniformly terminating. □

Since it is easily shown that the rewriting relation of a finite set of rules is locally finite, we have the following.

**Corollary 2.3.3.** The rewriting relation of a finite set of rules is uniformly terminating if and only if it is both globally finite and acyclic. □

In the next section we investigate methods of proving global finiteness, and in §§ 4 and 5 methods for proving that a rewriting relation is acyclic.
3. Proving global finiteness. In general, we cannot decide uniform termination algorithmically, as the following result of Huet and Lankford (Huet and Lankford (1977, Thm. 1)) shows.

**Theorem 3.1.** The uniform termination problem for rewrite rule systems is undecidable, even for terms restricted to unary and nullary function symbols.

From this result we can show that the question of global finiteness for rewrite rule systems is also undecidable.

**Theorem 3.2.** There is no decision procedure for global finiteness of rewrite rule systems.

**Proof.** Let \( R = \{ l_i \rightarrow r_i \} \) be a finite set of rewrite rules in which all terms \( l_i \) and \( r_i \) are built from a finite set \( \{ f_1, \ldots, f_k \} \cup \{ g_1, \ldots, g_m \} \) of function symbols, where each \( f_i \) is unary and each \( g_i \) is nullary, and a variable symbol \( x \). Let \( h \) be a function symbol distinct from any of the \( f_i \) or \( g_i \).

Construct a new set of rules

\[
R' = \{ l_i \rightarrow h(r_i) \} \cup \{ f_i(h(x)) \rightarrow h(f_i(x)): 1 \leq i \leq k \}.
\]

Then for any \( t_0, t_1, \ldots, t_n \),

\[
t_0 \rightarrow t_1 \rightarrow \cdots \rightarrow t_n \text{ (using } R),
\]

if and only if

\[
t_0 \rightarrow h(t_1) \rightarrow \cdots \rightarrow h^n(t_n) \text{ (using } R').
\]

where \( h^n \) denotes the \( n \)-fold composition of \( h \). The set \( R \) is uniformly terminating if and only if \( R' \) is globally finite (because if \( R \) is not globally finite, then \( R \) is not uniformly terminating). By the Huet and Lankford result, this means there can be no decision procedure for global finiteness. \( \square \)

Thus we must be satisfied with finding conditions that are sufficient to guarantee global finiteness. We begin with a simple sufficient condition that we can show is syntactically checkable.

Define the size of a term \( t \) to be the number of function and variable symbols it contains. Denoting this \( \text{Size}(t) \), we have \( \text{Size}(f(g(x,f(x)))) = 5 \), for example.

A rewrite rule, \( t \rightarrow u \), is nonexpanding if for every substitution \( \theta \), \( \text{Size}(\theta(t)) \geq \text{Size}(\theta(u)) \). Otherwise, a rewrite rule is called expanding.

A rewrite rule system \( R \) is said to be nonexpanding if and only if every rule in \( R \) is nonexpanding; we also say that the rewriting relation of \( R \) is nonexpanding.

**Lemma 3.3.** If a rewriting relation \( \rightarrow \) is nonexpanding, it is globally finite.

**Proof.** Since the relation is nonexpanding, if \( t \rightarrow^* u \) then \( \text{Size}(u) \leq \text{Size}(t) \). Also the only variables that can occur in \( u \) are those in \( t \). Thus there are only finitely many possibilities for \( u \), and \( \rightarrow \) must be globally finite. \( \square \)

The next lemma and theorem justify our claim that the nonexpanding condition is syntactically checkable. We define \( \text{Num}(v,t) \) to be the number of occurrences of the variable \( v \) in the term \( t \).

**Lemma 3.4.** Let \( \theta \) be the substitution \( [t_1/ v_1, \ldots, t_k/ v_k] \). Then for any term \( t \),

\[
\text{Size}(\theta(t)) = \text{Size}(t) + \sum_{j=1}^{k} \text{Num}(v_j,t) (\text{Size}(t_j) - 1).
\]

**Proof.** With suitable inductive definitions of \( \text{Size} \) and \( \text{Num} \), the lemma follows easily by induction on the structure of terms. \( \square \)

**Theorem 3.5.** A rewrite rule, \( t \rightarrow u \), is nonexpanding if and only if:
1) \( \text{Size}(t) \geq \text{Size}(u), \) and

2) for every variable \( v \) in \( t \), \( \text{Num}(v, t) \geq \text{Num}(v, u) \).

Proof. Let \( \theta \) be any substitution \([t_1/v_1, \cdots, t_k/v_k]\). By Lemma 3.3,

\[
(*) \quad \text{Size}(\theta(t)) - \text{Size}(\theta(u)) = \text{Size}(t) - \text{Size}(u) + \sum_{j=1}^{k} (\text{Num}(v_j, t) - \text{Num}(v_j, u))(\text{Size}(t_j) - 1),
\]

from which it is obvious that if 1) and 2) hold, then (*) is always nonnegative, implying that \( t \rightarrow u \) is nonexpanding.

Now suppose 1) or 2) fails to hold. If 1) fails, the identity substitution would make (*) negative. Suppose 2) fails, i.e., there is a \( v \) for which \( \text{Num}(v, t) < \text{Num}(v, u) \). Then again (*) can be made negative: taking \( \theta = [s/v] \) for some term \( s \), (*) simplifies to

\[
\text{Size}(t) - \text{Size}(u) + (\text{Num}(v, t) - \text{Num}(v, u))(\text{Size}(s) - 1),
\]

which becomes negative when \( \text{Size}(s) \) is sufficiently large. Thus \( t \rightarrow u \) fails to be nonexpanding. \( \square \)

Requiring a rewriting relation to be nonexpanding is restrictive since there are useful expanding but globally finite relations. For example, a set of rules for canonicalizing propositional formulas to their disjunctive normal form would include the following distributive law which is expanding though the rewriting relation is globally finite.

\[ x \cdot (y + z) \rightarrow x \cdot y + x \cdot z. \]

This restriction does have the advantage of requiring only strictly local, rule at a time, syntactic analysis. We have looked at relaxations of this restriction that preserve this locality property. An approach to relaxing this restriction so that rewrite rules using the if-then-else operator, which often arise in equational specifications of abstract data types, can be handled is discussed in Appendix A. The following two equations taken from a specification of the data type set specify one of its operations, \( \text{has} \), which tests for set membership.

\[
\text{has}(i, \text{null}) = \text{false},
\]
\[
\text{has}(i, \text{insert}(i', s)) \rightarrow \text{if } i = i' \text{ then true else has}(i, s).
\]

The rule, expressed using the if-then-else operator, corresponding to the second equation is

\[
\text{has}(i, \text{insert}(i', s)) \rightarrow \text{if-then-else}(i = i', \text{true, has}(i, s)).
\]

This approach can handle most rewriting systems that we have come across in specifying abstract data types (Musser (1980)).

4. Searching for cycles. We remind the reader of our basic approach to proving uniform termination: proving global finiteness, and proving there are no cycles in the rewriting relation. We have dealt with global finiteness in the previous section, and we now turn to the question of cycles. We assume we are given a term rewriting system \( R = \{l_i \rightarrow r_i\} \) whose rewriting relation is globally finite, and we wish to determine whether or not there is any cycle of terms

\[
(*) \quad t_0 \rightarrow t_1 \rightarrow \cdots \rightarrow t_n \rightarrow t_0
\]
The following development of a method of searching for cycles is based on some generalizations of notions that Knuth and Bendix (1970) used in testing for uniform confluence.

4.1. Superpositions and derived pairs. The definitions of superpositions and derived pairs depends on the important concept of "unification."

Two terms \( t \) and \( u \) are said to be unifiable if there is a substitution \( \theta \) such that \( \theta(t) \equiv \theta(u) \). \( \theta \) is called a unifier of \( t \) and \( u \), and whenever \( \theta \) is chosen so that it is a factor of any other unifier \( \theta_1 \) (i.e., \( \theta_1 \) can be written as a composition \( \theta_2 \circ \theta \) for some \( \theta_2 \)), then it is called a most general unifier (m.g.u.) of \( t \) and \( u \). It can be shown that the m.g.u. of two terms, if it exists, is unique up to variable renaming.

Consider, for example, the terms: \( t = f(x,g(y)) \) and \( u = f(h(z),w) \) and the substitutions:

\[
\begin{align*}
\theta_1 &= [h(g(u))/x,g(u)/z,g(y)/w] \\
\theta_2 &= [h(z)/x,g(y)/w].
\end{align*}
\]

\( \theta_1 \) unifies \( t \) and \( u \) to the term \( f(h(g(u)),g(y)) \), and

\( \theta_2 \) unifies \( t \) and \( u \) to the term \( f(h(z),g(y)) \).

\( \theta_1 \) is clearly not the most general unifier of \( t \) and \( u \) since it is not a factor of \( \theta_2 \). \( \theta_2 \), on the other hand, is the most general unifier of \( t \) and \( u \).

Two terms are said to overlap if one is unifiable with a nonvariable subterm of the other. In determining whether an overlap exists, the variables of one term are renamed, if necessary, so as not to conflict with those of the other term. Two terms could overlap in many ways resulting in many superpositions as discussed below.

Let \( s \) and \( t \) overlap. Their superposition is defined as either

a) \( s \) unifies with a nonvariable subterm \( t' \) of \( t \), by m.g.u. \( \theta \), in which case \( \theta(t) \) is called a superposition of \( s \) and \( t \); or

b) \( t \) unifies with a nonvariable subterm \( s' \) of \( s \), by m.g.u. \( \theta \), in which case \( \theta(s) \) is called a superposition of \( s \) and \( t \).

Consider, for example, the terms \( s = x^{-1} \circ x \) and \( t = (x' \circ y') \circ z' \). The substitution \( \theta = [x^{-1}/x',x/y'] \) unifies \( x^{-1} \circ x \) with \( x' \circ y' \). The resultant superposition of \( s \) and \( t \) is \( (x^{-1} \circ x) \circ z' \).

Now consider ordered pairs of terms \((r,s)\) and \((t,u)\) such that \( s \) and \( t \) overlap, as above. (If the variables of \( t \) must be renamed, the same renaming must be applied to \( u \).) Then along with the superposition \( \theta(t) \) or \( \theta(s) \) we obtain the derived pair of terms \( <p,q> \), where

a) if \( s \) unifies with a nonvariable subterm \( t[i] \) by m.g.u. \( \theta \),

\[
\begin{align*}
p &= \theta(t) \text{ with } \theta(r) \text{ at } i, \\
q &= \theta(u);
\end{align*}
\]

b) if \( t \) unifies with a nonvariable subterm \( s[i] \) by m.g.u. \( \theta \),

\[
\begin{align*}
p &= \theta(r), \\
q &= \theta(s) \text{ with } \theta(u) \text{ at } i.
\end{align*}
\]

In the case of a rewriting system \( R = \{(l_1,r_1),\ldots\} \), the derived pairs obtained from the pairs \((r_1,l_1)\) and \((l_1,r_1)\) are called critical pairs.

Consider, for example, obtaining a critical pair from the rewrite rules:

\[
\begin{align*}
x^{-1} \circ x &\Rightarrow e, \\
(x' \circ y') \circ z' &\Rightarrow x' \circ (y' \circ z').
\end{align*}
\]
We begin by constructing the ordered pairs \((e, x^{-1} \cdot x)\) and \(((x' \cdot y') \cdot z', x' \cdot (y' \cdot z'))\). As we saw earlier \(x^{-1} \cdot x\) can be unified with \(x' \cdot y'\) using the substitution \(\theta = [x^{-1}/x, x/y']\). This leads to the derived pair \(<e \cdot z', x^{-1} \cdot (x \cdot z')>\) which is a critical pair of the rules.

The computation of critical pairs is central to the Knuth-Bendix test for confluence (Knuth and Bendix (1970)). In this application, it is always the left-hand sides that are superposed with each other. In the method of searching for cycles to be described, we consider superpositions of the right- and left-hand sides as well.

4.2. Overlap closure. For a term rewriting system \(R\), the overlap closure of \(R\), written \(OC(R)\), is the term rewriting system defined inductively as follows:

a. Every rule \(r \rightarrow s\) in \(R\) is also in \(OC(R)\).

b. Whenever \(r \rightarrow s\) and \(t \rightarrow u\) are in \(OC(R)\), every derived pair \(<p, q>\) of \((r, s)\) and \((t, u)\) is in \(OC(R)\) (as \(p \rightarrow q\)).

c. No other rules are in \(OC(R)\).

That each derived pair is in fact a rewrite rule is shown by the following:

Lemma 4.2.1. If \(r, s, t, u\) are terms such that \((r, s)\) and \((t, u)\) are rewrite rules, then every derived pair \(<p, q>\) of \((r, s)\) and \((t, u)\) is also a rewrite rule.

Proof. One just has to verify that for each case in the definition of derived pair that every variable that occurs in \(q\) occurs also in \(p\). □

Examples of overlap closures:

i. Let \(R = \{f(x) \rightarrow g(x)\}\), then \(OC(R) = R\).

ii. Let \(R = \{f(x) \rightarrow g(h(x)), h(x) \rightarrow k(x)\}\), then \(OC(R) = R \cup \{f(x) \rightarrow g(k(x))\}\).

iii. Let \(R = \{x \cdot (y \cdot z) \rightarrow (x \cdot y) \cdot z\}\), then from the superposition \((x' \cdot y') \cdot z'\) we obtain the rule \(x \cdot (x' \cdot y') \cdot z' \rightarrow ((x \cdot x') \cdot y' \cdot z')\) and from the superposition \((x \cdot ((x' \cdot y') \cdot z')\) we obtain \(x \cdot (x' \cdot (y' \cdot z')) \rightarrow (x \cdot (x' \cdot y')) \cdot z'.\)

These rules then lead to further rules, and \(OC(R)\) is infinite.

iv. Let \(R = \{f(x) \rightarrow g(x), g(h(x)) \rightarrow f(h(x))\}\). Then \(OC(R)\) consists of \(R\) and the reflexive rules \(f(h(x)) \rightarrow f(h(x))\) and \(g(h(x)) \rightarrow g(h(x))\).

The name "overlap closure" comes from the fact that the rules of \(OC(R)\) are a subset of the transitive closure of the rewriting relation of \(R\):

Lemma 4.2.2. If \(p \rightarrow q\) is in \(OC(R)\) then \(p \rightarrow^+ q\) (using \(R\)).

Proof. By induction on the construction of \(p \rightarrow q\) in \(OC(R)\). The basis of the induction is the case that \(p \rightarrow q\) is included in \(OC(R)\) by virtue of being a rule of \(R\). Then obviously \(p \rightarrow^+ q\) holds. If \((p \rightarrow q)\) is included in \(OC(R)\) by being a derived pair of \((r, s)\) and \((t, u)\) then by the induction hypothesis for the two rules \((r, s)\) and \((t, u)\) we have \(r \rightarrow^+ s\) and \(t \rightarrow^+ u\). By the definition of derived pair and the transitivity of \(\rightarrow^+\), we then have \(p \rightarrow^+ q\). □

Corollary 4.2.3. If \(OC(R)\) contains a reflexive rule, \(t \rightarrow t\), then the rewriting relation of \(R\) has a cycle.

Proof. Immediate from the Lemma. □

We would like to have the converse of this corollary, that if the rewriting relation of \(R\) has a cycle, then \(OC(R)\) contains a reflexive rule. This would permit searching for cycles by incrementally computing \(OC(R)\), looking for a reflexive rule.
While we have not been able to prove this in full generality, we will present in the next section a restricted version and its proof. The proof is not easy, because the overlap closure of $R$ is in general much smaller than the full transitive closure of $R$. It is this small size, relative to the transitive closure, however, that makes it feasible to use the overlap closure as the basis of an approach to proving uniform termination or at least, a useful notion of “restricted termination.” These ideas will be discussed in § 6.

5. **Rewrite dominoes and overlap closure theorems.** In order to be able to develop and prove useful results about the overlap closure, we need to be able to deal precisely with the various cases of overlap between successive applications of rewrite rules in a rewrite sequence. We have found it useful to introduce a new representation of rewriting that helps to make such cases clear.

The **domino representation** (or rewrite domino) of a rewrite rule is a rectangle divided into left and right halves in which are inscribed tree representations of the left and right terms of the rule. Function symbols in the terms are represented by labelled circles in the trees. Variable symbols are represented by labeled rectangles, called “variable boxes.” For examples of some rules and their corresponding rewrite dominoes, see Fig. 1.

---

**Fig. 1. A set of rewrite rules and their corresponding rewrite dominoes.**
For each kind of domino (that is, each domino corresponding to a specific rule), we assume there is an infinite stock of dominoes of that kind with their variable rectangles filled in with all possible terms. For each such domino, we also assume an infinite number of copies are available in the stock.

A sequence of rewrites can be represented by a domino layout, which is a two-dimensional arrangement of dominoes that obeys the rules of matching corresponding to those of term rewriting (§ 2). Before giving the formal definition of a layout, we refer the reader to an example of a rewrite sequence using the rules given in Fig. 1 and its corresponding domino layout as shown in Fig. 2. Another example is in Fig. 3, and the two layouts in Figs. 2 and 3 could be concatenated to give a single longer layout.

We draw trees oriented sideways with the root at the left, and we will use nested triangles to represent trees schematically. We will give a simultaneous inductive definition of the terms layout, position of a domino in a layout, adjacency of dominoes in a layout, and right-end dominoes of a layout. (These definitions may seem tedious, but are necessary to avoid dependence in proofs on intuitive geometric notions.)

The basis of the definitions is a unit layout from t to w: a horizontal arrangement of a tree t, a domino, D, with trees u and v, and another tree w,

![Diagram of a unit layout from t to w]

in which at some position, i, in t (as defined in § 2) there is a subtree t' that is identical to u, ignoring the variable boxes that appear in u; and w is the tree [t with v at i].

The position of D in this layout is i, and D is a right-end domino. A layout is defined as follows: a unit layout is a layout; and if L is a layout from t₀ to t₁ and L₁ is a unit layout from t₁ to t₂, then the concatenation $L^+$ of L and L₁, with t₁ deleted, is a layout from t₀ to t₂. The position of the domino $D₁$ of L₁ in the new layout $L^+$ is the position i of $D₁$ in L₁. In $L^+$, $D₁$ is said to be adjacent to just those dominoes of D and L that satisfy: D is a right-end domino of L and the position j of D is a prefix of i or i is a prefix of j (recall that positions are sequences of natural numbers). These dominoes are also said to be adjacent to $D₁$, so that adjacency is a symmetric relation. Finally, we say that $D₁$ is a right-end domino of $L^+$, and each domino of L is a right-end domino of $L^+$ if and only if it is a right-end domino of L and it is not adjacent to $D₁$.

The examples in Figs. 2 and 3 illustrate a number of observations we can make about this representation of rewriting:

1. Two dominoes that are not in the transitive closure of the adjacency relation represent rewrites of disjoint subterms. Thus in drawing layouts we allow one of such a pair of dominoes to be placed above the other. In other words, in a domino layout there is no distinction between different orders of rewriting when the rules are being applied to disjoint subterms; e.g., the layout in Fig. 3 would not be different if rule 5 had been applied before rule 4 or before rule 3. One can think of these rules being applied in parallel, since the order of application is always immaterial in this case. The layout representation just makes this property especially evident.
Fig. 2. A rewrite domino layout and the corresponding rewriting sequence (using dominoes of Fig. 1).

Fig. 3. Another layout (a continuation of the layout in Fig. 2).
2. To the property that "the rightmost term of a rewrite sequence is terminal" corresponds the property that "there is no way to play a domino on the layout" (formally, there is no way to concatenate a unit layout onto the layout). The layout is said to be *blocked*. (The layout in Fig. 3 is blocked.)

3. Thus the rules have the uniform termination property if and only if every possible layout eventually is blocked. Equivalently, there are no infinite layouts.

Our purpose with this representation of rewriting is to provide a conceptual tool for finding and presenting proofs of new results about term rewriting systems. The first result we will prove with the aid of rewrite dominoes is one that will allow us to speed up the search for cycles by considering only those sequences of rewrites in which a "major rewrite" occurs.

A rewrite \( t_0 \rightarrow t_1 \) is called a *major rewrite* if it is by application of a rule, \( t \rightarrow u \), to the entire term \( t_0 \); i.e., for some substitution \( \theta \), \( \theta(t) = t_0 \) and \( \theta(u) = t_1 \). When only a proper subterm of \( t_0 \) is matched, \( t_0 \rightarrow t_1 \) is called a *minor rewrite*.

In a layout, a domino is called a *major domino* (of the layout) if it represents a major rewrite, and a *minor domino* otherwise.

A *major cycle* is a cycle in which at least one of the rewrites is major.

**Theorem 5.1.** If a rewriting relation has a cycle, it has a major cycle.

**Proof.** Let us define the *corridor* of a domino \( D \) in a layout to be the horizontal rectangle containing all dominoes whose positions in the layout are equal to or extensions of the position of \( D \), using the definition of position as a sequence of natural numbers. (For example, in Fig. 3, the central domino has position 1.1 and its corridor contains it and the top leftmost domino, which has the same position, but does not contain the bottom leftmost domino, which has position 1.2, or the rightmost domino, which has position 1. The corridor of the rightmost domino contains all of the dominoes of the layout.) If we add to a corridor of a domino \( D \) the appropriate trees at either end, it forms a sublayout of the original layout, and \( D \) is a major domino of this sublayout.

Any two corridors in a layout are either disjoint or one is contained in the other. Therefore, we can determine a sublayout which has a domino that is major in it, as follows: start with any leftmost domino and follow its corridor to the right; whenever a domino is encountered that does not lie in the corridor, adopt its corridor. When we reach the right end, we have a corridor containing a layout. If the whole layout is cyclic, the sublayout corresponding to this corridor will be also, and will represent a major cycle. \( \square \)

"Major cycle" is a weaker notion than "prime cycle" introduced by Klop (1980), i.e., every prime cycle is a major cycle but not vice versa. Consider for example, the rewriting system, \( \{ f(x,y) \rightarrow f(y,x), a() \rightarrow b(), b() \rightarrow a() \} \). The cycle

\[
\begin{align*}
f(a(),b()) & \rightarrow f(b(),a()) \rightarrow f(b(),b()) \\
& \rightarrow f(b(),a()) \rightarrow f(a(),a()) \rightarrow f(a(),b())
\end{align*}
\]

is a major cycle but is not a prime cycle because another cycle

\[
a() \rightarrow b() \rightarrow a()
\]

is contained in it. Klop also proves a result about prime cycles similar to Theorem 5.1.
We now want to define some terminology and some manipulations of layouts that will be useful in proving theorems about the overlap closure of a set of rules. Consider an adjacent pair of dominoes in a layout. Let \( t \) and \( u \) be the trees on the adjacent halves, where a subtree \( t' \) of \( t \) is identical to \( u \) (possibly \( t' = t \)):

If either of \( t' \) or \( u \) is contained entirely within a variable box, i.e., the match is not between two nonvariable subterms, we say that the pair of dominoes is \textit{weakly matched}, and otherwise that it is \textit{strongly matched}.

\textit{Examples.} In Fig. 3, the domino pair

is weakly matched. Similarly the pair

that appears in the concatenation of the layouts of Figs. 2 and 3 is weakly matched, while all the other adjacent pairs are strongly matched.

A term is said to be \textit{linear} if no variable occurs in it more than once. A rewrite rule is \textit{left-linear} if its left term is linear and \textit{right-linear} if its right term is linear.

Now suppose we have two weakly matched dominoes, as in Fig. 4a, where \( t' \) is contained in the \( x \) variable box. If the \((s,t)\) domino is right-linear (i.e., \( t \) is linear), then the pair of dominoes can be \textit{transposed} as follows: remove the \((u,v)\) domino from the layout and move the \((s,t)\) domino to the right, so that copies of the \((u,v)\) domino can be inserted to the left of the \((s,t)\) domino, one adjacent to each \( x \) box in \( s \) (see Fig. 4b). Then the resulting configuration is still a layout, (the dominoes all match, using the same set of rules) with the same end trees. This is the case also when a symmetric kind of transposition is performed on a layout in Fig. 5a, producing the layout in Fig. 5b, where we assume that the \((u,v)\) domino is left-linear.

Such transpositions cannot necessarily be performed on strongly matched dominoes, but we will define a different kind of manipulation for this case. Strong matching corresponds to the concept of overlapping in the definition of derived pairs: if
Fig. 4. Transposition of weakly matched dominoes, where left domino is right-linear.

Fig. 5. Transposition of weakly matched dominoes, where right domino is left-linear.
(r, s) and (t, u) are rules that have a derived pair <p, q>, then the dominoes corresponding to (r, s) and (t, u) can be placed in a layout so that they are strongly matched. The layout configuration shows just where the strong match occurs and identifies a potential derived pair.

Suppose now that instead of our stock of dominoes corresponding to a given rule set R, we have a stock corresponding to OC(R), the overlap closure of R. Then for any strongly matched pair of dominoes in a layout there is a domino in our stock which corresponds to a derived pair generated by the matching pair. By a technical lemma proved in Appendix B, we can replace the strongly matched pair in the layout by the “derived pair domino” thus identified, and the result will still be a layout with the same end trees.

We are now in a position to prove:

**Theorem 5.2.** Suppose the rewriting relation of R is globally finite and every rule in R is right-linear. If the rewriting relation of R has a cycle, OC(R) contains a reflexive rule.

**Proof.** (By construction.) Let

\[ t_0 \rightarrow t_1 \rightarrow \cdots \rightarrow t_n \rightarrow t_0 \]

be a given cycle. Corresponding to (*) is a cyclic domino layout

![Diagram of domino layout](image)

where the dominoes correspond to rules of R. In fact since each of these rules is also in OC(R), we may take this layout as a layout of dominoes corresponding to rules of OC(R). We will show how to manipulate this layout to a form that shows there is a reflexive rule \( t \rightarrow t \) in OC(R).

We describe the manipulations as an algorithm operating on the cyclic layout (**).

Step 1. [Extract major cycle.] As in the proof of Theorem 5.1, extract from (**) a sublayout representing a major cycle, making it the layout subject to the following steps. Also replace \( t_0 \) with its subterm matched by the layout.

Step 2. [Push major dominoes to right end.] Manipulate the layout to a form in which all of the major dominoes are together at the right end, by means of transpositions or replacements by derived pair dominoes; whenever D is a major domino and E is a minor domino adjacent to D on the right

![Diagram of dominoes](image)

either D and E are weakly matched, in which case they can be transposed, or they are strongly matched, in which case they can be replaced by the derived pair domino they define — which is a major domino. This derived pair domino is also right linear, as the lemma in Appendix C shows.
Step 3. [Look for cycle among major dominoes.] There is now a nonempty sequence of major dominoes $D_1, \ldots, D_m$ at the right end of the layout:

```
< a b D_1 \cdots D_m <
```

These dominoes can only be strongly matched — except for the case where the right-hand side of $D_i$ is just a variable, but shortly we will show that such a possibility can be ruled out. If there is some contiguous subsequence $D_i, \ldots, D_j$ that forms a cyclic layout

```
< D_i \cdots D_j <
```

then, since there can only be strong matches, these dominoes can be combined by $j - i + 1$ replacements into a single domino $D$ that forms a cyclic layout:

```
< a b D <
```

Let $D$ represent $(p, q)$. Then there is a substitution $\theta$ such that $u_0 = \theta(p)$ and $\theta(q) = u_0$, i.e., $\theta$ unifies $p$ and $q$. Furthermore, a derived pair of $(p, q)$ and $(p, q)$ is the reflexive rule $(\theta(p), \theta(q))$. Since this is in $\text{OC}(R)$, we terminate the algorithm.

Step 4. [Duplicate.] If no such subsequence exists, concatenate to the layout a copy of the layout. Return to Step 2 with the resulting layout:

```
< a b D \cdots D_m <
```

That concludes the statement of the algorithm. Before considering the question of termination of the algorithm, we dispense with the detail mentioned in Step 3: the case of adjacent major dominoes $D$ and $E$ where the right term $u$ of $D$ is just a variable. We can assume the left term $t$ of $D$ is not just a variable (if it were then it would have to be the same variable as $u$ and we would already have a reflexive rule). Since the layout is cyclic, if we drop $D$ from the layout, we obtain a layout that has as its right end term a proper subterm identical to the left end term. From this we conclude that the term rewriting relation is not globally finite, contrary to assumption. This contradiction rules out the case under discussion.

It is obvious that each step of this algorithm is effective and terminating. Overall termination is guaranteed by the following facts:

a. At the $k$th execution of Step 2, the number of major dominoes, $m$, at the right end is at least $2^k$.

b. Let $t_0^{(k)}$ denote the term to the left of $D_1$ in the layout at the $k$th execution of Step 3. Since each $t_0^{(k)}$ is derived from $t_0$ and the rewriting relation is globally finite, there are only finitely many distinct possibilities for $t_0^{(k)}$. By a), then, there is one such term for which arbitrarily long layouts of major
dominoes exist. Again by global finiteness, these layouts cannot all continue without producing a term, \( u_0 \), that is a duplicate of some term previously obtained in the layout.

Since the algorithm always terminates, and does so with a reflexive rule in \( \text{OC}(\mathcal{R}) \), this proves the theorem. \( \square \)

The corresponding theorem obtained by replacing "right-linear" by "left-linear" can also be proved in a similar manner. Combining these theorems with Corollary 4.3, we have:

**Theorem 5.3.** Suppose the rewriting relation of \( \mathcal{R} \) is globally finite and every rule in \( \mathcal{R} \) is right-linear or every rule in \( \mathcal{R} \) is left-linear. Then the rewriting relation of \( \mathcal{R} \) is uniformly terminating if and only if \( \text{OC}(\mathcal{R}) \) contains no reflexive rule.

In the next section we explore some applications of this theorem.

Recently, Dershowitz (1981) has proposed a "forward chain" construction for rewriting systems and proved that a right-linear rewriting system is uniformly terminating if and only if it has no infinite forward chains. However, for left-linear systems the analogous result requires that the left-hand sides of the rules be nonoverlapping, a problem that we had independently encountered when considering the forward chain construction and a similar backward chain construction. We were thus led to invent the overlap closure construction. The following example from Dershowitz (1981) illustrates the advantage of the overlap closure construction over forward chains. Using the forward chain construction, it is not possible to determine the nontermination of this left-linear rewrite system, as pointed out by Dershowitz. The rewriting system is

\[
\begin{align*}
f(a(), b(), x) & \rightarrow f(x, x, b()), \\
b() & \rightarrow a().
\end{align*}
\]

These rules have only two forward chains, both finite:

\[
\begin{align*}
f(a(), b(), x) & \rightarrow f(x, x, b()) \rightarrow f(x, x, a()), \quad \text{and} \quad b() \rightarrow a(),
\end{align*}
\]

but we cannot conclude anything about the termination of the rules because they are not right-linear and, although they are left-linear, the left-hand sides are overlapping. But in the overlap closure construction, the rules have a derived pair rule

\[
f(b(), b(), x) \rightarrow f(x, x, b()),
\]

which, when overlapped with itself, gives the reflexive rule

\[
f(b(), b(), b()) \rightarrow f(b(), b(), b()),
\]

as a derived pair, proving that the rules are nonterminating.

An open question about the power of the overlap closure construction is whether the assumption of left-linearity or right-linearity is necessary. Although we have not been able to find proofs of our results without this assumption, we have also been unable to construct a counterexample.

**6. Using the overlap closure.** Theorem 5.3 implies that the uniform termination property for globally finite, right-linear (left-linear) rewrite systems is decidable when the overlap closure is finite. Unfortunately, \( \text{OC}(\mathcal{R}) \) will usually be infinite.

When \( \text{OC}(\mathcal{R}) \) is infinite, Theorem 5.3 implies that nontermination is semidecidable. In our experience, if \( \mathcal{R} \) is nonterminating, then there is a cycle involving terms of size comparable to the terms in the rules. Since rules in \( \text{OC}(\mathcal{R}) \) get very big soon, we are likely to generate a reflexive rule very quickly.
Theorem 5.3 has utility for proving "restricted termination," i.e., termination for all terms in some set S. We describe this as termination over S. A particularly interesting set to look at is the set of all terms of limited size.

**Lemma 6.1.** If a rewriting relation arrow is globally finite and S is finite, termination over S is decidable.

**Proof.** For each element of S, we just have to follow all possible rewriting paths until we either reach a terminal form or a cycle. By global finiteness, there can be only finitely many such paths and each is of finite length. □

Let \( OC_n(R) = \{(l, r) \in OC(R) : \text{Size}(l) \leq n\} \). If R contains only nonexpanding rules, then \( OC_n(R) \) can be computed by starting with the empty set, inserting all rules \((l, r)\) of R such that \( \text{Size}(l) \leq n \), and continuing to compute and insert derived pairs until a set S is reached such that for any rules \((r, s)\) and \((t, u)\) from S any derived pair \((p, q)\) is either in S or has \( \text{Size}(p) > n \). As for \( OC(R) \), checking whether \( OC_n(R) \) has a reflexive rule is better for deciding termination over set of terms of size \( \leq n \) than the brute force approach suggested in the proof of the above lemma, because the search space in the former case is reduced considerably.

**Lemma 6.2.** Suppose R contains only right-linear (left-linear) and nonexpanding rules. If \( OC_n(R) \) contains no reflexive rule, R is terminating over the set of all terms of size \( n \) or less.

**Proof.** By contradiction. Suppose there is a cycle with terms of size less than or equal to \( n \); from the cycle, we will construct a reflexive rule in \( OC_n(R) \) in the same way as in the proof of Theorem 5.2.

Let the cycle be

\[
t_0 \rightarrow t_1 \rightarrow \cdots \rightarrow t_m = t_0.
\]

The size of \( t_i \) is less than or equal to \( n \); note that the size of each term in the cycle is the same, as rules are restricted to be nonexpanding. Using the construction suggested in the proof of Theorem 5.2, we can extract a major cycle. In Step 2, if the rules are right-linear (left-linear), weak matches between adjacent dominoes are transposed to the left (right). At the end of Step 2, there is a nonempty sequence of major dominoes having terms of size \( k \) at the right (left) end of the domino layout. If there is some contiguous subsequence that forms a cyclic layout, then there is a reflexive rule in \( OC_k(R) \). Otherwise, we do Step 4. Since all dominoes have terms of size \( k \), their derived pairs will be of size less than or equal to \( k \). So, we will eventually get a reflexive rule in \( OC_k(R) \). □

A reduction relation arrow is said to be canonical over S if it is terminating over S and any two elements in S have the same terminal form if and only if they belong to the same \( \sim \) equivalence class. Thus for a term rewriting relation that is canonical over S, equations expressed using terms drawn from S are decidable by rewriting to terminal forms and checking for identity. We have the following partial generalization of Theorem 2.2.2.

**Theorem 6.3.** If a reduction relation arrow is both terminating over S and uniformly confluent, it is canonical over S.

**Remark.** Uniform confluence, not just "confluence over S," is necessary here.

The following rewrite system illustrates this point.

\[
a(x) \rightarrow h(x),
\]

\[
a(x) \rightarrow k(x).
\]
If \( S = \{ h(x), k(x) \} \), then clearly \( \rightarrow \) is both terminating over \( S \) and confluent over \( S \), but is not canonical over \( S \), as \( h(x) = k(x) \).

\textbf{Proof} By contradiction. Let \( t_1 \) and \( t_2 \) be in \( S \) for which arrow is not canonical; i.e., \( t_1 \) and \( t_2 \) belong to the same \( \leftrightarrow \) equivalence class but they have different terminal forms \( t_1^* \) and \( t_2^* \).

\[ t_1 \leftrightarrow t_1' \leftrightarrow \cdots \leftrightarrow t_k' \leftrightarrow t_2 \]

where \( \leftrightarrow \) stands for either \( \rightarrow \) or \( \leftarrow \). If \( t_1 \rightarrow t_1' \), then \( t_1' \) also has the terminal form \( t_1^* \). Otherwise, if \( t_1 \leftarrow t_1' \), then \( t_1' \) has the terminal form \( t_1^* \) by uniform confluence. Similarly, for any \( t_i' \leftrightarrow t_j' \), \( t_i' \) and \( t_j' \) have the same terminal form \( t_i^* \). In particular, \( t_2 \) has the terminal form \( t_1^* \), but \( t_2^* \) is also the terminal form of \( t_2 \) implying that \( t_1^* = t_2^* \). Hence the contradiction. \( \square \)

The above theorem about restricted canonicity is useful only if it is possible to prove uniform confluence (or more importantly to transform a given set of rules into a uniformly confluent set of rules having the same equational theory as the original set) in the absence of uniform termination requirement on the rule set. We have attempted to modify the Knuth-Bendix algorithm to do this, but without success. Peterson and Stickel (1981) remark that such an approach for establishing uniform confluence is not likely to be successful, as it would allow one to prove the decidability of the word equation problem over free semigroups in a simple way as follows:

Given the rule set for semigroups:

\[ x \cdot (y \cdot z) \rightarrow (x \cdot y) \cdot z, \]

\[ (x \cdot y) \cdot z \rightarrow x \cdot (y \cdot z); \]

to solve a word equation problem (i.e., for a pair of terms, \( t_1 \) and \( t_2 \), whether there exists a substitution of variables which make the resulting instances of \( t_1 \) and \( t_2 \) equal in the theory of free semigroups), we add the following rules:

\[ h(t_1) \rightarrow 0, \]

\[ h(t_2) \rightarrow 1, \]

where \( h \) is a new function symbol distinct from all other function symbols appearing in words. (0 and 1 are assumed to be distinct.) If the above set of rules is uniformly confluent, then there does not exist any substitution \( \theta \) such that \( \theta(t_1) = \theta(t_2) \) in the theory of free semigroups; otherwise, such a substitution exists. The word equation problem over free semigroups is a difficult problem which has been worked on for over 20 years and only recently solved by Makanin who gives a very lengthy and complex proof.

Proving restricted termination is useful in analyzing automatic implementation of a data type generated from its algebraic specification as discussed in Guttag, Horowitz, and Musser (1978). An implementation of an operation is essentially taking the expression corresponding to the operation invocation and simplifying it using the rewrite rules specifying the operation behavior. It is possible to run such an implementation and evaluate expressions whose termination can be proved a priori. Analysis of the operation behavior is helpful in designing abstract data types.

Proving restricted termination is also useful in an OBJ-like system (Goguen and Tardo (1979)) for designing specification of abstract data types and algorithms using equational axioms. Axioms are viewed as unidirectional rewrite rules and specifications are analyzed by interpreting expressions. OBJ has a memory mode in
which during the simplification of expressions, rewriting of expressions in the presence of cycles is allowed. While rewriting a term, when a term from the set $S$ whose termination is proved is hit, there is no need to store the intermediate terms generated in the rewriting from that point onwards.

7. Conclusion. In rewrite rule theorem proving, as elsewhere, almost all of the interesting questions are undecidable. A particularly interesting undecidable question, with a number of practical ramifications, is whether or not a given set of rewrite rules has the uniform termination property. The motivation behind the work presented in this paper is the circumvention of this undecidability.

Our rather distinctive approach to this began with dividing the problem of proving uniform termination into the separable component problems of proving global finiteness and proving acyclicity. We then dealt with the former by developing sufficient conditions for establishing global finiteness. The conditions developed in § 3 and Appendix A seem reasonably general, and, moreover, are computationally easy to check. They are based on showing that every instance of each rule is nonexpanding, a property for which we presented syntactically checkable necessary and sufficient conditions.

Proving acyclicity, on the other hand, is not amenable to any kind of local, rule at a time, analysis. Our approach to dealing with this problem started with the development of a procedure that finds a cycle if one exists but may not terminate otherwise. The obvious procedure for doing this, which is tremendously inefficient, is based on the enumeration of all terms over the alphabet of the rewriting system. Our procedure, which is closely related to the Knuth-Bendix procedure for constructing confluent sets of rewrite rules, is based on the computation of what we called the overlap closure of the set of rewrite rules. In order to develop this procedure, we introduced in § 5 a new model of term rewriting called rewrite dominoes. The primary application of this model was to prove that a set of rewrite rules, provided they are all right-linear (or all left-linear) and globally finite, has a cycle if and only if its overlap closure contains a reflexive rule.

This result implies that in the cases where the overlap closure computation terminates we can decide uniform termination of the original set of rewrite rules. Unfortunately the overlap closure is rarely finite. However, our procedure generates the overlap closure in such a way that for any integer $n$ we can generate a subset of the overlap closure that is sufficient to decide whether or not there is a cycle in which every term is of size $n$ or less. This gives us a decision procedure for what we called restricted termination. We conjecture that for certain classes of term rewriting systems, it should be possible to compute a bound, $n$, such that if a cycle exists, there exists a cycle in which every term is of size $n$ or less. For such classes, the overlap closure would provide a decision procedure for uniform termination.

We have explored the utility of restricted termination in the application of term rewriting with which we are most familiar, the algebraic specification of abstract data types, and see a number of interesting ways to make use of it there. We suspect the notion of restricted termination will be useful in other application areas, but we have not investigated this. Investigations should also be made into the usefulness of the overlap closure computation and the domino model of term rewriting systems for the study of properties other than uniform or restricted termination.
Appendix A. Proving uniform termination of rules using if-then-else operator. In specifying abstract data types using equations, it is often useful to introduce an auxiliary function if-then-else with the following semantics:

\[
\begin{align*}
\text{if-then-else}(\text{true},x,y) & = x, \\
\text{if-then-else}(\text{false},x,y) & = y.
\end{align*}
\]

But the rewrite rules corresponding to the equational axioms using if-then-else are often expanding; for example, consider the following rule:

\[
\text{has?}(i,\text{insert}(s,i')) \rightarrow \text{if-then-else}(i = i',\text{true},\text{has?}(i,s)).
\]

We will only consider rewrite rules whose right-hand side may use if-then-else operator; their left-hand sides are assumed not to use if-then-else, except for the rules

\[
\begin{align*}
\text{if-then-else}(\text{true},x,y) & \rightarrow x, \\
\text{if-then-else}(\text{false},x,y) & \rightarrow y.
\end{align*}
\]

The uniform termination of such rules can be proved by considering the termination of a corresponding set of rules that do not use if-then-else operator, as the following theorem shows. For a rule \((l,r)\) in \(R\), the corresponding set \(D(\{(l,r)\})\) of rules is defined as:

\[
D(\{(l,r)\}) = \bigcup_i D(R_i) \quad \text{if } r \text{ does not use if-then-else.}
\]

A rule \((l,r)\) expressed using if-then-else is called componentwise-nonexpanding if the corresponding derived set of rules, \(D(\{(l,r)\})\), is nonexpanding. Similarly, the rule set \(R\) is componentwise-nonexpanding if \(D(R)\) is nonexpanding.

6 Theorem A.1. If \(R\) is componentwise-nonexpanding, then \(R\) is uniformly terminating if and only if \(D(R)\) is uniformly terminating.

Proof. Since \(R\) is componentwise-nonexpanding, \(D(R)\) is nonexpanding and hence globally finite.

(i) \(\implies\) (by contradiction)

Assume that \(D(R)\) is not uniformly terminating and \(R\) is. Since \(D(R)\) is globally finite, there is a cycle \(t_0 \rightarrow t_1 \rightarrow \cdots \rightarrow t_n \equiv t_0, n > 0\). This cycle must use a rule in \(D(\{(l\rightarrow r)\})\) such that \(r\) contains the if-then-else operator, otherwise \(R\) would have a cycle. From the above cycle, we can construct a sequence of rewrites in \(R\) such that wherever a rule from the set corresponding to a rule in \(R\) using if-then-else is applied in the cycle, we apply the rule in \(R\). Let the sequence of rewrites be

\[
t_0 \rightarrow t'_1 \rightarrow \cdots \rightarrow t'_n.
\]

\(t'_n\) in that case contains \(t_0\), which can be further rewritten. Thus we get an infinite sequence of rewrites using \(R\), implying \(R\) is not terminating, which is a contradiction.

(ii) \(\implies\) (by contradiction)
Assume that \( R \) is not uniformly terminating and \( D(R) \) is. There are two possibilities:

(a) \( R \) leads to an infinite sequence of rewrites.

\[
t_0 \rightarrow t_1 \rightarrow \cdots \rightarrow t_n \rightarrow \cdots
\]

Since \( R \) is componentwise-nonexpanding, there must exist \( i, j \) such that \( j > i \) and the subterm, \( t'_k \) in \( t_i \) being rewritten at the \( i+1 \)st step is a subterm in \( t_j \). From the rewriting sequence

\[
t_i \rightarrow \cdots \rightarrow t_j, (*)
\]

we can construct a finite set \( S \) of rewriting sequences in \( D(R) \) by following the rewriting sequence \((*)\) as follows:

(i) Initialize \( S \) to be \( \{ t_i \} \).

(ii) At a rewriting step \( t_k \rightarrow t_{k+1} \) in which the subterm \( t'_k \) at position \( m \) is rewritten using the rule \((l, r)\) of \( R \),

(a) if \( r \) does not use if-then-else operator, discard sequences in \( S \) whose last term does not have \( t'_k \) at position \( m \) and extend other sequences in \( S \) by rewriting \( t'_k \) at position \( m \) using \((l, r)\),

(b) otherwise, discard sequences in \( S \) whose last term does not have \( t'_k \) at position \( m \) and extend other sequences in \( S \) by rewriting \( t'_k \) at position \( m \) in all possible ways using every rule in \( D((l, r)) \).

Thus \( S \) will have a sequence with its last term having \( t'_i \) as a subterm. This implies that \( D(R) \) is nonterminating.

(b) \( R \) has a finite cycle:

Using the same argument as in (a), it can be shown that \( D(R) \) also has a finite cycle. \( \square \)

The above theorem can be generalized to handle rules having the following property: the rules use function symbols that have the following semantics and that do not appear in the left-hand side of any rules other than the rules giving their semantics:

\[
f(i, x_1, \cdots, x_n) \rightarrow x_i \quad \text{for } 1 \leq i \leq n.
\]

Note that if-then-else operator is a particular case of the above function symbol \( f \) when \( n = 2 \). So, for proving uniform termination of such rules, we derive from the rules using these function symbols on their right-hand side, a corresponding set of rules which do not use these function symbols and prove uniform termination of the new set.

**Appendix B. Proof of lemma used in Theorem 5.2.**

**Lemma B.1.** Suppose \( t_0 \rightarrow t_1 \) using \( r \rightarrow s \) applied at position \( i \), \( t_1 \rightarrow t_2 \) using \( t \rightarrow u \) applied at \( i, j \), and \( s[j] \) and \( t \) overlap determining the derived pair \( <p, q> = <\theta(r), \theta(s) \text{ with } \theta(u) \text{ at } j> \). Then \( t_0 \rightarrow t_2 \) using \( p \rightarrow q \) applied at \( i \). A similar result holds for the case in which \( s \) unifies with a subterm of \( t \).

**Proof.** Rename the variables of \( t \) and \( u \), if necessary, so that \( s \) and \( t \) have no variable in common. There is some subterm \( t_0[i] \) and a substitution \( \theta_1 \) such that \( \theta_1(r) = t_0[i] \) and \( t_1 = [t_0 \text{ with } \theta_1(s) \text{ at } j] \).

Again, there is some subterm \( t_1[(i, j)] \) and a substitution \( \theta_2 \) such that \( \theta_2(t) = t_1[(i, j)] \) and \( t_2 = [t_1 \text{ with } \theta_2(u) \text{ at } i, j] \).
Since the variables of $s$ and $t$ are disjoint, we have $(\theta_1 \cup \theta_2)(s[j]) = \theta_1(s[j]) = \theta_2(t) = (\theta_1 \cup \theta_2)(t)$. That is, $\theta_1 \cup \theta_2$ is a unifier of $s[j]$ and $t$ and therefore has $\theta$ as a factor:

$$\theta_1 \cup \theta_2 = \theta_3 \cdot \theta,$$

for some substitution $\theta_3$.

Thus $t_0[i] = \theta_1(r) = (\theta_1 \cup \theta_2)(r) = (\theta_3 \cdot \theta)(r) = \theta_3(\theta(r)) = \theta_3(p)$. That is, $t_0$ is matched by $p$ at $i$. Now consider $\theta_3(q)$; it is

$$\theta_3([\theta(s) \text{ with } \theta(u) \text{ at } j])$$

$$= [\theta_3(\theta(s)) \text{ with } \theta_3(\theta(u)) \text{ at } j]$$

$$= [\theta_1(s) \text{ with } \theta_2(u) \text{ at } j].$$

Thus $t_2 = [t_1 \text{ with } \theta_2(u) \text{ at } i \cdot j]$}

$$= [[t_0 \text{ with } \theta_1(s) \text{ at } i] \text{ with } \theta_2(u) \text{ at } i \cdot j]$$

$$= [t_0 \text{ with } [\theta_1(s) \text{ with } \theta_2(u) \text{ at } j] \text{ at } i]$$

$$= [t_0 \text{ with } \theta_3(q) \text{ at } i],$$

showing that $t_0 \rightarrow t_2$ using $p \rightarrow q$ applied at $i$.

We omit the proof of the case in which $s$ unifies with a subterm of $t$. $\square$

**Appendix C. Derived pair construction is (left, right) linearity preserving.**

**Theorem C.1.** If $r \rightarrow s$ and $t \rightarrow u$ are two right linear rules with disjoint variable sets, then each of their derived pairs, $< p, q >$ is also right linear.

**Proof.** There are three cases:

(i) $s$ unifies with the subterm $t[i]$ of $t$ by their m.g.u. $\theta$.

The corresponding derived pair $< p, q >$ has

$$p = [\theta(t) \text{ with } \theta(r) \text{ at } i],$$

$$q = \theta(u).$$

Since $s$ is linear, by the following lemma, substitutions for any two distinct variables in $t[i]$ in $\theta$ do not have a common variable. The variables in $t$ other than the ones in $t[i]$ do not play any role. So $\theta(u)$ is linear.

(ii) the subterm $s[i]$ of $s$ unifies with $t$ by their m.g.u. $\theta$.

The corresponding derived pair $< p, q >$ has

$$p = \theta(r),$$

$$q = [\theta(s) \text{ with } \theta(u) \text{ at } i].$$

Since $s[i]$ is linear, by the lemma, substitutions for any two distinct variables in $t$ in $\theta$ do not have a common variable. So, $\theta(s)$ and $\theta(u)$ are linear. And, $q$ is thus linear.

(iii) if subterms of $s$ do not unify with $t$ or $s$ does not unify with subterms of $t$, then there are no derived pairs of $r \rightarrow s$ and $t \rightarrow u$. $\square$

By a similar argument, it can also be proved that every derived pair of two left linear rules is left linear.

**Lemma C.2.** Let $t$ and $u$ be unifiable terms with disjoint variable sets, and $\theta$ be their most general unifier. Let $\theta^*$ be the restriction of $\theta$ to the variables of $u$, say $\theta^* = [e_1/\nu_1, \cdots, e_n/\nu_n]$. If $t$ is linear, then all variables in $e_1, \cdots, e_n$ are distinct.
Proof. For every variable \( x \) having \( k \) (> 1) occurrences in \( u \), replace different occurrences of \( x \) by distinct variables \( x_1, \ldots, x_k \) that do not appear in \( t \) and \( u \). Let \( u' \) be the resulting term which is linear.

By the following lemma, in the m.g.u. \( \theta' \) of \( t \) and \( u' \), substitutions for distinct variables in \( t \) and \( u' \) do not have a common variable. Let \( \sigma_x \) be the m.g.u. for the set of terms \( \theta'(x_i), 1 \leq i \leq k \), the substitutions for the variables used to replace multiple occurrences of \( x \) in \( u \). If these \( \sigma_x \) for every variable \( x \) having multiple occurrences in \( u \) are composed with \( \theta' \), we get a unifier of \( t \) and \( u \).

In this unifier, substitutions for variables in \( u \) do not have a common variable. From this, it is evident that the m.g.u. \( \theta \) of \( t \) and \( u \) cannot have substitutions for variables in \( u \) that share common variables. \( \square \)

Lemma C.3. For two unifiable terms \( t \) and \( u \), if \( t \) and \( u \) are linear, then the substitutions in their m.g.u. \( \theta \) for any two distinct variables of \( t \) or \( u \) do not have common variables.

Proof. By induction on the structure in term \( t \).

Basis. \( t \) is a variable.
Then \( \theta(t) = u \) and the statement trivially holds.

Inductive step. \( t \) is \( f(t_1, \ldots, t_n) \).

For \( t \) and \( u \) to be unifiable, either \( u \) is a variable or \( u \) is \( f(u_1, \ldots, u_n) \). The case of \( u \) being a variable is handled as in the basis step.

For the case \( u \) is \( f(u_1, \ldots, u_n) \), for each \( i, 1 \leq i \leq n, t_i \) must unify with \( u_i \) by their m.g.u. \( \theta \), say. By the inductive hypothesis, the statement holds for each of \( \theta_i \).

Since \( t \) and \( u \) are linear, the disjoint union of \( \theta_i, 1 \leq i \leq n \), is the m.g.u. \( \theta \) of \( t \) and \( u \).

It follows that the statement of the lemma holds for \( \theta \) also. \( \square \)

Acknowledgments. The basic idea of conducting a search for repeated terms (cycles) or subterms sprang from discussions in 1977 between one of the authors (Musser) and Dallas Lankford. We thank P. Gloess, G. Huet, and J. Levy for their interest and assistance in refuting some of our earlier conjectures, thus helping us arrive at the notion of the overlap closure and the theorems of § 5. We also thank P. Narendran for assistance in constructing the proof of the theorems in Appendix C, J. Goguen for discussions of the approach to term rewriting used in OBJ, and M. O'Donnell for helpful comments and pointing out the relationship between some of the definitions in § 5 to Klop’s work.

REFERENCES


